intel

Microsystem Components Handbook

Microprocessors Volume I

Order Number: 230843-003

intel Microsystem Components Handbook, Volume I 1986

LITERATURE

To order Intel Literature write or call:

Intel Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130

Intel Literature Sales:
(800) 548-4725
Other Inquiries:
(800) 538-1876

Use the order blank on the facing page or call our Toll Free Number listed above to order literature. Remember to add your local sales tax and a 10% handling charge for U.S. customers, 20% for Canadian customers.

1986 HANDBOOKS

Product Line handbooks contain data sheets, application notes, article reprints and other design information.

NAME	ORDER NUMBER	*PRICE IN U.S. DOLLARS
COMPLETE SET OF 9 HANDBOOKS Get a 30% discount off the retail price of $171.00	231003	**$120.00**
MEMORY COMPONENTS HANDBOOK	210830	**$18.00**
MICROCOMMUNICATIONS HANDBOOK	231658	**$18.00**
MICROCONTROLLER HANDBOOK	210918	**$18.00**
MICROSYSTEM COMPONENTS HANDBOOK Microprocessor and peripherals (2 Volume Set)	230843	**$25.00**
DEVELOPMENT SYSTEMS HANDBOOK	210940	**$18.00**
OEM SYSTEMS HANDBOOK	210941	**$18.00**
SOFTWARE HANDBOOK	230786	**$18.00**
MILITARY HANDBOOK	210461	**$18.00**
QUALITY/RELIABILITY HANDBOOK	210997	**$20.00**
PRODUCT GUIDE Overview of Intel's complete product lines	210846	**No charge**
LITERATURE GUIDE Listing of Intel Literature	210620	**No charge**
INTEL PACKAGING SPECIFICATIONS Listing of Packaging types, number of leads, and dimensions	231369	**No charge**

*These prices are for the U. S. and Canada only. In Europe and other international locations, please contact your local Intel Sales Office or Distributor for literature prices.

intel®

U.S. LITERATURE ORDER FORM

NAME: ______________________________

COMPANY: ______________________________

ADDRESS: ______________________________

CITY: ______________________ STATE: __________ ZIP: __________

COUNTRY: ______________________________

PHONE NO.: (____) ______________________________

ORDER NO.	TITLE	QTY.	PRICE	TOTAL
☐☐☐☐☐☐ - ☐☐☐	______	____ ×	____ =	____
☐☐☐☐☐☐ - ☐☐☐	______	____ ×	____ =	____
☐☐☐☐☐☐ - ☐☐☐	______	____ ×	____ =	____
☐☐☐☐☐☐ - ☐☐☐	______	____ ×	____ =	____
☐☐☐☐☐☐ - ☐☐☐	______	____ ×	____ =	____
☐☐☐☐☐☐ - ☐☐☐	______	____ ×	____ =	____
☐☐☐☐☐☐ - ☐☐☐	______	____ ×	____ =	____

Subtotal __________

Your Local Sales Tax __________

Add appropriate postage and handling to subtotal
10% U.S.
20% Canada

→ Postage & Handling __________

Allow 2-4 weeks for delivery

Total __________

Pay by Visa, MasterCard, Check or Money Order, payable to Intel Books. Purchase Orders have a $50.00 minimum

☐ Visa ☐ MasterCard Expiration Date __________

Account No. ______________________________

Signature: ______________________________

Mail To: Intel Literature Sales
P.O. Box 58130
Santa Clara, CA
95052-8130

Customers outside the U.S. and Canada should contact the local Intel Sales Office or Distributor listed in the back of most Intel literature.

Call Toll Free: (800) 548-4725 for phone orders

Prices good until 12/31/86.

Source HB

Mail To: Intel Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130

intel®

MICROSYSTEM COMPONENTS HANDBOOK

1986

About Our Cover:
The design on our front cover is an abstract portrayal of the unlimited interface linking options available with Intel microsystem components. Intel microprocessors and associated peripherals are the building blocks which provide total systems development solutions. Intel's superior technology, reliability and support provides easier solutions to specific development problems. Thereby, cutting "time-to-market" and creating a greater market share.

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

> Above, BITBUS, COMMputer, CREDIT, Data Pipeline, GENIUS, i, $\hat{\text{I}}$, ICE, iCEL, iCS, iDBP, iDIS, I²ICE, iLBX, i_m, iMDDX,, iMMX, Insite, Intel, $\text{int}_\text{e}\text{l}$, $\text{int}_\text{e}\text{lBOS}$, Intelevision, $\text{int}_\text{e}\text{ligent}$ Identifier, $\text{int}_\text{e}\text{ligent}$ Programming, Intellec, Intellink, iOSP, iPDS, iPSC, iRMX, iSBC, iSBX, iSDM, iSXM, KEPROM, Library Manager, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, ONCE, OpenNET, Plug-A-Bubble, PROMPT, Promware, QUEST, QueX, Ripplemode, RMX/80, RUPI, Seamless, SLD, UPI, and VLSiCEL, and the combination of ICE, iCS, iRMX, iSBC, iSBX, MCS, or UPI and a numerical suffix.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

*MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Distribution
Mail Stop SC6-59
3065 Bowers Avenue
Santa Clara, CA 95051

Table of Contents

—VOLUME 2—

CHAPTER 6

Numeric Index

CUSTOMER SUPPORT

CUSTOMER SUPPORT

Customer Support is Intel's complete support service that provides Intel customers with Customer Training, Software Support and Hardware Support.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. Intel's extensive customer support includes factory repair services as well as worldwide field service offices providing hardware repair services, software support services and customer training classes.

HARDWARE SUPPORT

Hardware Support Services provides maintenance on Intel supported products at board and system level. Both field and factory services are offered. Services include several types of field maintenance agreements, installation and warranty services, hourly contracted services (factory return for repair) and specially negotiated support agreements for system integrators and large volume end-users having unique service requirements. For more information contact your local Intel Sales Office.

SOFTWARE SUPPORT

Software Support Service provides maintenance on software packages via software support contracts which include subscription services, information phone support, and updates. Consulting services can be arranged for on-site assistance at the customer's location for both short-term and long-term needs. For complex products such as NDS II or I^2ICE, orientation/installation packages are available through membership in Insite User's Library, where customer-submitted programs are catalogued and made available for a minimum fee to members. For more information contact your local Intel Sales Office.

CUSTOMER TRAINING

Customer Training provides workshops at customer sites (by agreement) and on a regularly scheduled basis at Intel's facilities. Intel offers a breadth of workshops on microprocessors, operating systems and programming languages, etc. For more information on these classes contact the Training Center nearest you.

TRAINING CENTER LOCATIONS

To obtain a complete catalog of our workshops, call the nearest Training Center in your area.

Boston	(617) 692-1000	London	(0793) 696-000
Chicago	(312) 310-5700	Munich	(089) 5389-1
San Francisco	(415) 940-7800	Paris	(01) 687-22-21
Washington, D.C.	(301) 474-2878	Stockholm	(468) 734-01-00
Israel	(972) 349-491-099	Milan	39-2-82-44-071
Tokyo	03-437-6611	Benelux (Rotterdam)	(10) 21-23-77
Osaka (Call Tokyo)	03-437-6611	Copenhagen	(1) 198-033
Toronto, Canada	(416) 675-2105	Hong Kong	5-215311-7

Overview

1

INTRODUCTION

Intel microprocessors and peripherals provide a complete solution in increasingly complex application environments. Quite often, a single peripheral device will replace anywhere from 20 to 100 TTL devices (and the associated design time that goes with them).

Built-in functions and standard Intel microprocessor/peripheral interface deliver very real *time* and *performance* advantages to the designer of microprocessor-based systems.

REDUCED TIME TO MARKET

When you can purchase an off-the-shelf solution that replaces a number of discrete devices, you're also replacing all the design, testing, and debug *time* that goes with them.

INCREASED RELIABILITY

At Intel, the rate of failure for devices is carefully tracked. Highest reliability is a tangible goal that translates to higher reliability for your product, reduced downtime, and reduced repair costs. And as more and more functions are intergrated on a single VLSI device, the resulting system requires less power, produces less heat, and requires fewer mechanical connections—again resulting in greater system reliability.

LOWER PRODUCTION COST

By minimizing design time, increasing reliability, and replacing numerous parts, microprocessor and peripheral solutions can contribute dramatically to lower product costs.

HIGHER SYSTEM PERFORMANCE

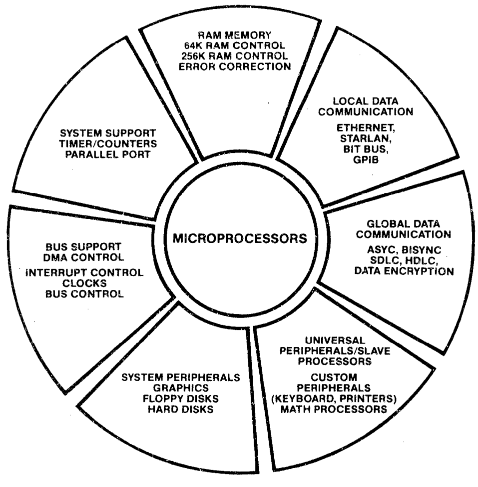
Intel microprocessors and peripherals provide the highest system performance for the demands of today's (and tomorrow's) microprocessor-based applications. For example, the 80386 32 bit offers the highest performance for multitasking, multiuser systems. Intel's peripheral products have been designed with the future in mind. They support all of Intel's 8, 16 and 32 bit processors.

HOW TO USE THE GUIDE

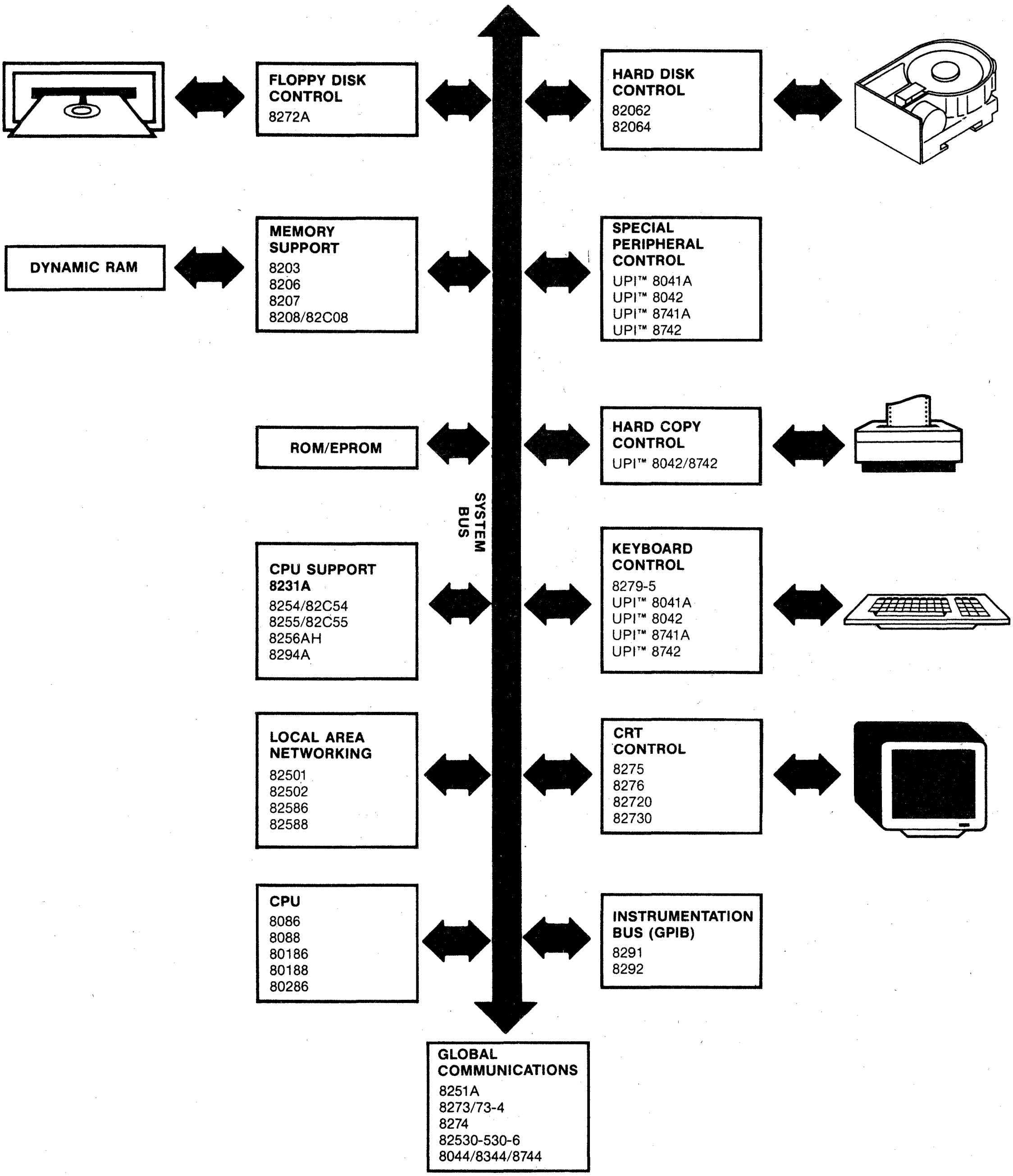
The following application guide illustrates the range of microprocessors and peripherals that can be used for the applictions in the vertical column of the left. The peripherals are grouped by the I/O function they control. CRT datacommunication, universal (user programmable), mass storage dynamic RAM controllers, and CPU/bus support.

An "X" in a horizontal application row indicates a potential peripheral or CPU, depending upon the features desired. For example, a conversational terminal could use either of the three display controllers, depending upon features like the number of characters per row or font capability. A "Y" indicates a likely candidate, for example, the 8272A Floppy Disk Controller in a small business computer.

The Intel microprocessor and peripherals family provides a broad range of time-saving, high performance solutions.



Intel's Microsystem Components Kit Solution



Get Your Kit Together!

APPLICATION CHART

POTENTIAL APPLICATION X—TYPICAL APPLICATION Y

APPLICATION	MICROPROCESSORS					DISPLAY			DATA COMMUNICATIONS								UPI	DISK		DRAM CONTROL				SUPPORT			
	8088	8086	80188	80186	80286	8275/76	82720	82730/731*	8251A	8273*	8274	8291A/92/94*	82530	82588*	82586/501/502*	8044/8744*	8042/8742	8272A	82062/064	8203*	8206	8207*	8208*/C08*	8254/C54	8255/C55	8256AH	8279/8279-5
PERIPHERALS																											
Printers	X		X	Y			X	X	X			X				X	X							X	X	X	
Plotters	X	X	X	Y					X			X				X	X								X		
Keyboards			X	Y					X							X	X								X		X
MASS STORAGE																											
Hard Disk	X	X	Y	Y															X						X		
Mini Winchester	X		Y	Y															Y								
Tape			X	Y												X	X								X		
Cassette			Y	Y													X								X		
Floppy/Mini			Y	Y														Y									
COMMUNICATIONS																											
PBX			X	X	Y					X	X		X			X				X	X	X	X	X	X	X	
LANS	X	X	Y	Y							X		X	X	Y	X								X			
Modems									X		X		X			X	X										X
Bisync									X		X			X										X			
SDLC/HDLC										X	X		X			X								X			
Serial Backplane										X	X		X		X	X											
Central Office		X		X	Y					X	X		X			X					X	X	X	X		X	
Network Control		Y		X	Y						X		X	X	X	X								X			
OFFICE/BUS																											
Copier/FAX	X		X	Y								X			X	X	X									X	X
Wordprocessor	X	X	X	X	Y		X	Y	X							X		Y		X	X	X	X	Y	Y	X	
Typewriter			X													X	X								Y		
Electronic Mail		X	X	X			X	Y			X		X	X	X						X						
Transaction System		Y	X	X	X				X							X								Y	X	X	
Data Entry	X	X	X	X		X	X	X	X							X	X						X	Y		X	X
COMPUTERS																											
SM Bus Computer		X	Y	X	X	X	Y	Y	X		X		X	X	X	X	X	Y	Y	X	X	X	X	Y	Y	X	
PC	Y	X		X	X	X	Y	Y	X		X	X	X	X	X	X	X	Y	Y	X	X	X	X	Y	Y	X	
Portable PC				X					X								X	Y		Y			X	Y	Y	X	
Home Computer	X	X	Y	X	X		X		X								X	Y		X		X	X	Y	Y	X	

APPLICATION	MICROPROCESSORS					DISPLAY			DATA COMMUNICATIONS								UPI	DISK		DRAM CONTROL				SUPPORT			
	8088	8086	80188	80186	80286	8275/76	82720	82730/731*	8251A	8273*	8274	8291A/92/94*	82530	82588*	82586/501/502*	8044/8744*	8042/8742	8272A	82062/064	8203*	8206	8207*	8208*/C08*	8254/C54	8255/C55	8256AH	8279/8279-5
TERMINALS																											
Conversational			Y			X	X	X	X	X												X	X	Y		X	
Graphics CRT		Y		Y	Y		Y	Y	X		X		X		X			X	X	X	X	X	X	Y	Y	X	
Editing	X	X	X	X	X		Y	Y	X		X		X		X	X		X		X		X	X	Y	Y	Y	
Intelligent	X	X	Y	Y				Y	X		X		X	X	X			X	X			X	X	Y	Y	X	
Videotex	X	X	X	X			X	X	X													X	X	Y	Y		
Printing, Laser, Impact	X	X	X	X			X	X	X						X	X	X					X	X	Y	Y		
Portable	X	X	Y				X	X	X									X		X		X	X	Y	Y		
INDUSTRIAL AUTO																											
Robotics			Y	Y	X						X		X			Y	X				X				Y		X
Network	X	X	X	X	X					X	X		X	X	X	Y	X				X						
Numeric Control		X	X	X	X		Y			X	X		X		X	Y	X				X						X
Process Control	X	X	X	Y	X		Y			X	X		X		X	Y	X			X	X		X	X	X		X
Instrumentation	X	X	X	X			Y					X				Y	X				X			X	X		X
Aviation/Navigation		X	X	X	X												X				X			X	X		X
INDUSTRIAL/DATA ACQ.																											
Laboratory Instrumentation	X	X	Y	X	X							Y				Y	X					X	X		X		X
Source Data	X		Y													Y						X	X		X		X
Auto Test	X	X	Y	X	Y											Y						X	X		X		X
Medical	X	X	Y	X	X		Y								X	Y	X				X	X	X		X		X
Test Instrumentation	X	X	Y	X	X						X		X		X	Y	X					X	X		X		X
Security		X	Y	X										X		Y	X				X	X	X		X		X
COMMERCIAL DATA PROCESSING																											
POS Terminal		X	X	Y		X	X	X	X		X		X		X	X	X			X	X	X	X	X	X	X	
Financial Transfer		X	X	Y		X	X	X	X					X		Y					X	X	X		X	X	
Automatic Teller		X	X	Y		X	X	X	X							Y	X				X	X	X		X	X	X
Document Processing	X	X	X	Y	X	X	X	X	X							Y						X	X			X	
WORKSTATIONS																											
Office	X	X	X	Y	X		Y	Y	X		X		X	X	X	X	X	Y	Y	X	X	X	X	X	Y	X	
Engineering	X	X		Y	X		Y	Y	X		X		X	X	X		X	Y	Y	X	X	X	X	X	Y	X	
CAD		X		Y	Y		Y	Y	X		X		X		X		X	Y	Y	X	X	X	X	X	Y		
MINI MAINFRAME																											
Processor & Control Store		X		Y	Y	Y					X		X								X	X					
Database Subsystems		X		Y	X	X								X							X	X					
I/O Subsystem			Y	Y	Y						X		X			X						X					
Comm. Subsystem		X		Y	Y				X		X		X	X		X						X					

*Single Source Product

8080, 8085 Microprocessors

2

intel® 8080A/8080A-1/8080A-2 8-BIT N-CHANNEL MICROPROCESSOR

- **TTL Drive Capability**
- **2 μs (—1:1.3 μs, —2:1.5 μs) Instruction Cycle**
- **Powerful Problem Solving Instruction Set**
- **6 General Purpose Registers and an Accumulator**
- **16-Bit Program Counter for Directly Addressing up to 64K Bytes of Memory**
- **16-Bit Stack Pointer and Stack Manipulation Instructions for Rapid Switching of the Program Environment**
- **Decimal, Binary, and Double Precision Arithmetic**
- **Ability to Provide Priority Vectored Interrupts**
- **512 Directly Addressed I/O Ports**
- **Available in EXPRESS — Standard Temperature Range**
- **Available in 40-Lead Cerdip and Plastic Packages**
  (See Packaging Spec, Order #231369)

The Intel® 8080A is a complete 8-bit parallel central processing unit (CPU). It is fabricated on a single LSI chip using Intel's n-channel silicon gate MOS process. This offers the user a high performance solution to control and processing applications.

The 8080A contains 6 8-bit general purpose working registers and an accumulator. The 6 general purpose registers may be addressed individually or in pairs providing both single and double precision operators. Arithmetic and logical instructions set or reset 4 testable flags. A fifth flag provides decimal arithmetic operation.

The 8080A has an external stack feature wherein any portion of memory may be used as a last in/first out stack to store/retrieve the contents of the accumulator, flags, program counter, and all of the 6 general purpose registers. The 16-bit stack pointer controls the addressing of this external stack. This stack gives the 8080A the ability to easily handle multiple level priority interrupts by rapidly storing and restoring processor status. It also provides almost unlimited subroutine nesting.

This microprocessor has been designed to simplify systems design. Separate 16-line address and 8-line bidirectional data busses are used to facilitate easy interface to memory and I/O. Signals to control the interface to memory and I/O are provided directly by the 8080A. Ultimate control of the address and data busses resides with the HOLD signal. It provides the ability to suspend processor operation and force the address and data busses into a high impedance state. This permits OR-tying these busses with other controlling devices for (DMA) direct memory access or multi-processor operation.

NOTE:
The 8080A is functionally and electrically compatible with the Intel® 8080.

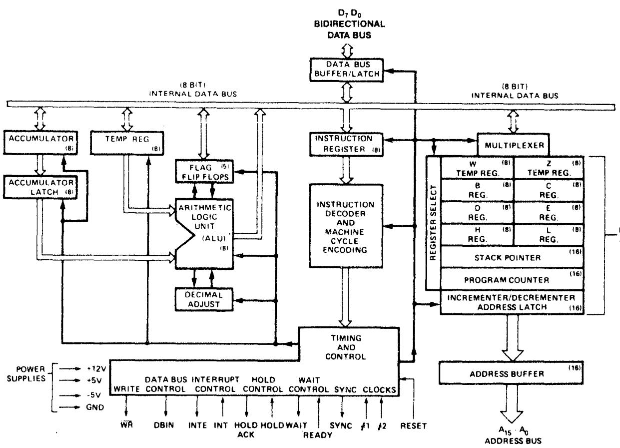


Figure 1. Block Diagram

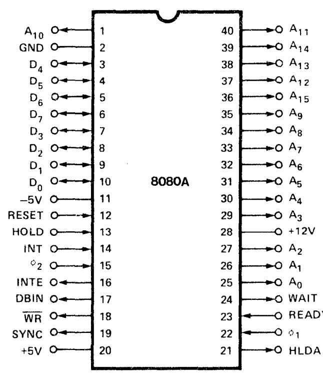


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Type	Name and Function
A_{15}-A_0	O	**Address Bus:** The address bus provides the address to memory (up to 64K 8-bit words) or denotes the I/O device number for up to 256 input and 256 output devices. A_0 is the least significant address bit.
D_7-D_0	I/O	**Data Bus:** The data bus provides bi-directional communication betweeen the CPU, memory, and I/O devices for instructions and data transfers. Also, during the first clock cycle of each machine cycle, the 8080A outputs a status word on the data bus that describes the current machine cycle. D_0 is the least significant bit.
SYNC	O	**Synchronizing Signal:** The SYNC pin provides a signal to indicate the beginning of each machine cycle.
DBIN	O	**Data Bus In:** The DBIN signal indicates to external circuits that the data bus is in the input mode. This signal should be used to enable the gating of data onto the 8080A data bus from memory or I/O.
READY	I	**Ready:** The READY signal indicates to the 8080A that valid memory or input data is available on the 8080A data bus. This signal is used to synchronize the CPU with slower memory or I/O devices. If after sending an address out the 8080A does not receive a READY input, the 8080A will enter a WAIT state for as long as the READY line is low. READY can also be used to single step the CPU.
WAIT	O	**Wait:** The WAIT signal acknowledges that the CPU is in a WAIT state.
$\overline{WR}$	O	**Write:** The $\overline{WR}$ signal is used for memory WRITE or I/O output control. The data on the data bus is stable while the $\overline{WR}$ signal is active low ($\overline{WR} = 0$).
HOLD	I	**Hold:** The HOLD signal requests the CPU to enter the HOLD state. The HOLD state allows an external device to gain control of the 8080A address and data bus as soon as the 8080A has completed its use of these busses for the current machine cycle. It is recognized under the following conditions: • the CPU is in the HALT state. • the CPU is in the T2 or TW state and the READY signal is active. As a result of entering the HOLD state the CPU ADDRESS BUS (A_{15}-A_0) and DATA BUS (D_7-D_0) will be in their high impedance state. The CPU acknowledges its state with the HOLD ACKNOWLEDGE (HLDA) pin.
HLDA	O	**Hold Acknowledge:** The HLDA signal appears in response to the HOLD signal and indicates that the data and address bus will go to the high impedance state. The HLDA signal begins at: • T3 for READ memory or input. • The Clock Period following T3 for WRITE memory or OUTPUT operation. In either case, the HLDA signal appears after the rising edge of ϕ_2.
INTE	O	**Interrupt Enable:** Indicates the content of the internal interrupt enable flip/flop. This flip/flop may be set or reset by the Enable and Disable Interrupt instructions and inhibits interrupts from being accepted by the CPU when it is reset. It is automatically reset (disabling further interrupts) at time T1 of the instruction fetch cycle (M1) when an interrupt is accepted and is also reset by the RESET signal.
INT	I	**Interrupt Request:** The CPU recognizes an interrupt request on this line at the end of the current instruction or while halted. If the CPU is in the HOLD state or if the Interrupt Enable flip/flop is reset it will not honor the request.
RESET[1]	I	**Reset:** While the RESET signal is activated, the content of the program counter is cleared. After RESET, the program will start at location 0 in memory. The INTE and HLDA flip/flops are also reset. Note that the flags, accumulator, stack pointer, and registers are not cleared.
V_{SS}		**Ground:** Reference.
V_{DD}		**Power:** $+12 \pm 5\%$ Volts.
V_{CC}		**Power:** $+5 \pm 5\%$ Volts.
V_{BB}		**Power:** $-5 \pm 5\%$ Volts.
ϕ_1, ϕ_2		**Clock Phases:** 2 externally supplied clock phases. (non TTL compatible)

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias . . . . 0°C to +70° C
Storage Temperature . . . . -65°C to +150°C
All Input or Output Voltages
With Respect to V_{BB} . . . . -0.3V to +20V
V_{CC}, V_{DD} and V_{SS} With Respect to V_{BB} -0.3V to +20V
Power Dissipation 1.5W

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{DD} = +12V ±5%, V_{CC} = +5V ±5%, V_{BB} = −5V ±5%, V_{SS} =0V; unless otherwise noted)

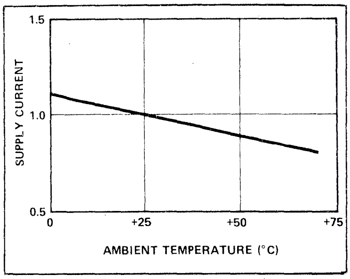
Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	$V_{SS}-1$		$V_{SS}+0.8$	V	
V_{IHC}	Clock Input High Voltage	9.0		$V_{DD}+1$	V	
V_{IL}	Input Low Voltage	$V_{SS}-1$		$V_{SS}+0.8$	V	
V_{IH}	Input High Voltage	3.3		$V_{CC}+1$	V	
V_{OL}	Output Low Voltage			0.45	V	I_{OL} = 1.9mA on all outputs, I_{OH} = -150μA.
V_{OH}	Output High Voltage	3.7			V	
$I_{DD\,(AV)}$	Avg. Power Supply Current (V_{DD})		40	70	mA	Operation T_{CY} = .48 μsec
$I_{CC\,(AV)}$	Avg. Power Supply Current (V_{CC})		60	80	mA	
$I_{BB\,(AV)}$	Avg. Power Supply Current (V_{BB})		.01	1	mA	
I_{IL}	Input Leakage			±10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{CL}	Clock Leakage			±10	μA	$V_{SS} \leq V_{CLOCK} \leq V_{DD}$
I_{DL}	Data Bus Leakage in Input Mode			-100 -2.0	μA mA	$V_{SS} \leq V_{IN} \leq V_{SS}+0.8V$ $V_{SS}+0.8V \leq V_{IN} \leq V_{CC}$
I_{FL}	Address and Data Bus Leakage During HOLD			+10 -100	μA	$V_{ADDR/DATA} = V_{CC}$ $V_{ADDR/DATA} = V_{SS} + 0.45V$

CAPACITANCE (T_A = 25°C, V_{CC} = V_{DD} = V_{SS} = 0V, V_{BB} = −5V)

Symbol	Parameter	Typ.	Max.	Unit	Test Condition
C_ϕ	Clock Capacitance	17	25	pf	f_c = 1 MHz
C_{IN}	Input Capacitance	6	10	pf	Unmeasured Pins
C_{OUT}	Output Capacitance	10	20	pf	Returned to V_{SS}

NOTES:

1. The RESET signal must be active for a minimum of 3 clock cycles.
2. ΔI supply / ΔT_A = -0.45%/°C.

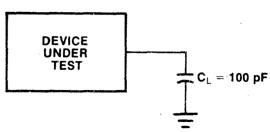


Typical Supply Current vs. Temperature, Normalized[2]

A.C. CHARACTERISTICS (8080A) (T_A = 0°C to 70°C, V_{DD} = +12V ±5%, V_{CC} = +5V ±5%, V_{BB} = -5V ±5%, V_{SS} = 0V; unless otherwise noted)

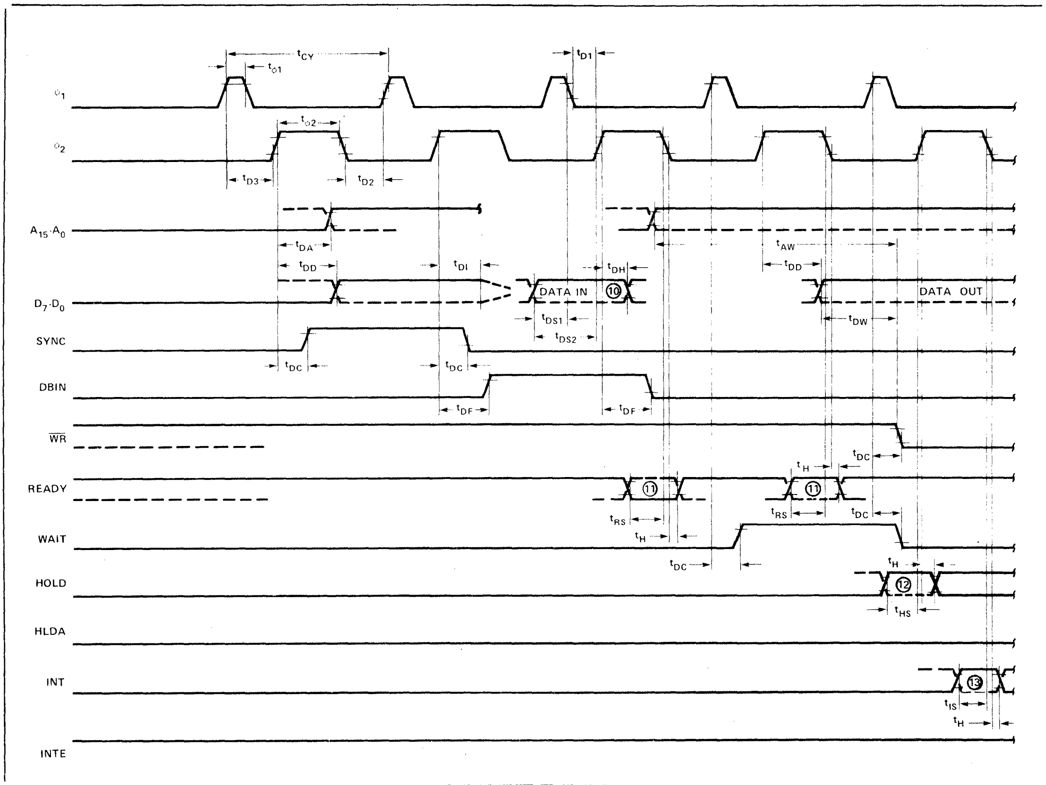
Symbol	Parameter	Min.	Max.	-1 Min.	-1 Max.	-2 Min.	-2 Max.	Unit	Test Condition
t_{CY}[3]	Clock Period	0.48	2.0	0.32	2.0	0.38	2.0	μsec	
t_r, t_f	Clock Rise and Fall Time	0	50	0	25	0	50	nsec	
$t_{\phi 1}$	ϕ_1 Pulse Width	60		50		60		nsec	
$t_{\phi 2}$	ϕ_2 Pulse Width	220		145		175		nsec	
t_{D1}	Delay ϕ_1 to ϕ_2	0		0		0		nsec	
t_{D2}	Delay ϕ_2 to ϕ_1	70		60		70		nsec	
t_{D3}	Delay ϕ_1 to ϕ_2 Leading Edges	80		60		70	nsec		
t_{DA}	Address Output Delay From ϕ_2		200		150		175	nsec	C_L = 100 pF
t_{DD}	Data Output Delay From ϕ_2		200		180		200	nsec	
t_{DC}	Signal Output Delay From ϕ_1 or ϕ_2 (SYNC, WR, WAIT, HLDA)		120		110		120	nsec	C_L = 50 pF
t_{DF}	DBIN Delay From ϕ_2	25	140	25	130	25	140	nsec	
t_{DI}[1]	Delay for Input Bus to Enter Input Mode		t_{DF}		t_{DF}		t_{DF}	nsec	
t_{DS1}	Data Setup Time During ϕ_1 and DBIN	30		10		20		nsec	
t_{DS2}	Data Setup Time to ϕ_2 During DBIN	150		120		130		nsec	
t_{DH}[1]	Data Holt time From ϕ_2 and DBIN	[1]		[1]		[1]		nsec	
t_{IE}	INTE Output Delay From ϕ_2		200		200		200	nsec	C_L = 50 pF
t_{RS}	READY Setup Time During ϕ_2	120		90		90		nsec	
t_{HS}	HOLD Setup Time During ϕ_2	140		120		120		nsec	
t_{IS}	INT Setup Time During ϕ_2	120		100		100		nsec	
t_H	Hold Time From ϕ_2 (READY, INT, HOLD)	0		0		0		nsec	
t_{FD}	Delay to Float During Hold (Address and Data Bus)		120		120		120	nsec	
t_{AW}	Address Stable Prior to WR	[5]		[5]		[5]		nsec	
t_{DW}	Output Data Stable Prior to WR	[6]		[6]		[6]		nsec	
t_{WD}	Output Data Stable From WR	[7]		[7]		[7]		nsec	
t_{WA}	Address Stable From WR	[7]		[7]		[7]		nsec	
t_{HF}	HLDA to Float Delay	[8]		[8]		[8]		nsec	
t_{WF}	WR to Float Delay	[9]		[9]		[9]		nsec	
t_{AH}	Address Hold Time After DBIN During HLDA	-20		-20		-20		nsec	

A.C. TESTING LOAD CIRCUIT



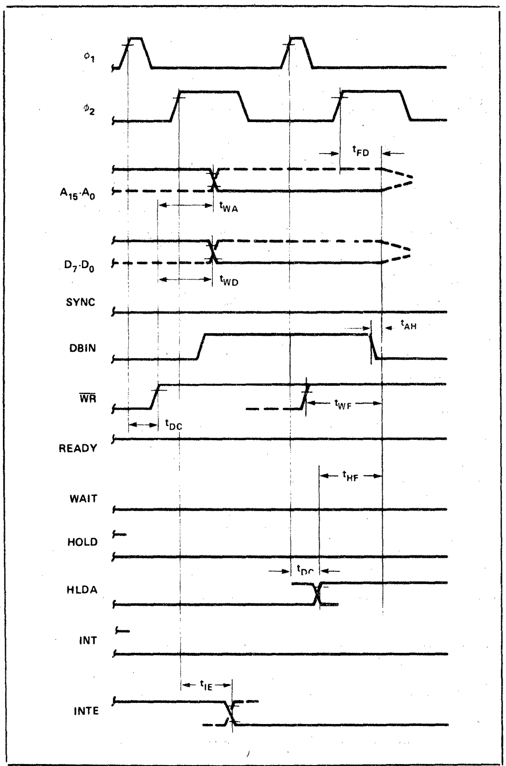
C_L = 100 pF
C_L INCLUDES JIG CAPACITANCE

WAVEFORMS



NOTE:
Timing measurements are made at the following reference voltages: CLOCK "1" = 8.0V, "0" = 1.0V; INPUTS "1" = 3.3V, "0" = 0.8V; OUTPUTS "1" = 2.0V, "0" = 0.8V.

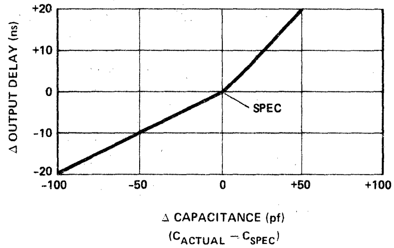
WAVEFORMS (Continued)



NOTES: (Parenthesis gives -1, -2 specifications, respectively)

1. Data input should be enabled with DBIN status. No bus conflict can then occur and data hold time is assured.
   $t_{DH} = 50$ ns or t_{DF}, whichever is less.
2. $t_{CY} = t_{D3} + t_{r\phi2} + t_{\phi2} + t_{f\phi2} + t_{D2} + t_{r\phi1} \geq 480$ ns ($-$ 1:320 ns, $-$ 2:380 ns).

TYPICAL Δ OUTPUT DELAY VS. Δ CAPACITANCE



3. The following are relevant when interfacing the 8080A to devices having $V_{IH} = 3.3V$:
   a) Maximum output rise time from .8V to 3.3V = 100ns @ C_L = SPEC.
   b) Output delay when measured to 3.0V = SPEC +60ns @ C_L = SPEC.
   c) If C_L = SPEC, add .6ns/pF if $C_L > C_{SPEC}$, subtract .3ns/pF (from modified delay) if $C_L < C_{SPEC}$.
4. $t_{AW} = 2\,t_{CY} - t_{D3} - t_{r\phi2} - 140$ ns ($-$ 1:110 ns, $-$ 2:130 ns).
5. $t_{DW} = t_{CY} - t_{D3} - t_{r\phi2} - 170$ ns ($-$ 1:150 ns, $-$ 2:170 ns).
6. If not HLDA, $t_{WD} = t_{WA} = t_{D3} + t_{r\phi2} + 10$ ns. If HLDA, $t_{WD} = t_{WA} = t_{WF}$.
7. $t_{HF} = t_{D3} + t_{r\phi2} - 50$ ns).
8. $t_{WF} = t_{D3} + t_{r\phi2} - 10$ns.
9. Data in must be stable for this period during DBIN T_3. Both t_{DS1} and t_{DS2} must be satisfied.
10. Ready signal must be stable for this period during T_2 or T_W. (Must be externally synchronized.)
11. Hold signal must be stable for this period during T_2 or T_W when entering hold mode, and during T_3, T_4, T_5 and T_{WH} when in hold mode. (External synchronization is not required.)
12. Interrupt signal must be stable during this period of the last clock cycle of any instruction in order to be recognized on the following instruction. (External synchronization is not required.)
13. This timing diagram shows timing relationships only; it does not represent any specific machine cycle.

INSTRUCTION SET

The accumulator group instructions include arithmetic and logical operators with direct, indirect, and immediate addressing modes.

Move, load, and store instruction groups provide the ability to move either 8 or 16 bits of data between memory, the six working registers and the accumulator using direct, indirect, and immediate addressing modes.

The ability to branch to different portions of the program is provided with jump, jump conditional, and computed jumps. Also the ability to call to and return from subroutines is provided both conditionally and unconditionally. The RESTART (or single byte call instruction) is useful for interrupt vector operation.

Double precision operators such as stack manipulation and double add instructions extend both the arithmetic and interrupt handling capability of the 8080A. The ability to increment and decrement memory, the six general registers and the accumulator is provided as well as extended increment and decrement instructions to operate on the register pairs and stack pointer. Further capability is provided by the ability to rotate the accumulator left or right through or around the carry bit.

Input and output may be accomplished using memory addresses as I/O ports or the directly addressed I/O provided for in the 8080A instruction set.

The following special instruction group completes the 8080A instruction set: the NOP instruction, HALT to stop processor execution and the DAA instructions provide decimal arithmetic capability. STC allows the carry flag to be directly set, and the CMC instruction allows it to be complemented. CMA complements the contents of the accumulator and XCHG exchanges the contents of two 16-bit register pairs directly.

Data and Instruction Formats

Data in the 8080A is stored in the form of 8-bit binary integers. All data transfers to the system data bus will be in the same format.

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0

DATA WORD

The program instructions may be one, two, or three bytes in length. Multiple byte instructions must be stored in successive words in program memory. The instruction formats then depend on the particular operation executed.

		TYPICAL INSTRUCTIONS
One Byte Instructions		
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	OP CODE	Register to register, memory reference, arithmetic or logical, rotate, return, push, pop, enable or disable Interrupt instructions
Two Byte Instructions		
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	OP CODE	
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	OPERAND	Immediate mode or I/O instructions
Three Byte Instructions		
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	OP CODE	Jump, call or direct load and store instructions
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	LOW ADDRESS OR OPERAND 1	
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	HIGH ADDRESS OR OPERAND 2	

For the 8080A a logic "1" is defined as a high level and a logic "0" is defined as a low level.

Table 2. Instruction Set Summary

Mnemonic	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Operations Description	Clock Cycles [2]
	Instruction Code [1]									
MOVE, LOAD, AND STORE										
MOVr1,r2	0	1	D	D	D	S	S	S	Move register to register	5
MOV M,r	0	1	1	1	0	S	S	S	Move register to memory	7
MOV r,M	0	1	D	D	D	1	1	0	Move memory to register	7
MVI r	0	0	D	D	D	1	1	0	Move immediate register	7
MVI M	0	0	1	1	0	1	1	0	Move immediate memory	10
LXI B	0	0	0	0	0	0	0	1	Load immediate register Pair B & C	10
LXI D	0	0	0	1	0	0	0	1	Load immediate register Pair D & E	10
LXI H	0	0	1	0	0	0	0	1	Load immediate register Pair H & L	10
STAX B	0	0	0	0	0	0	1	0	Store A indirect	7
STAX D	0	0	0	1	0	0	1	0	Store A indirect	7
LDAX B	0	0	0	0	1	0	1	0	Load A indirect	7
LDAX D	0	0	0	1	1	0	1	0	Load A indirect	7
STA	0	0	1	1	0	0	1	0	Store A direct	13
LDA	0	0	1	1	1	0	1	0	Load A direct	13
SHLD	0	0	1	0	0	0	1	0	Store H & L direct	16
LHLD	0	0	1	0	1	0	1	0	Load H & L direct	16
XCHG	1	1	1	0	1	0	1	1	Exchange D & E, H & L Registers	4
STACK OPS										
PUSH B	1	1	0	0	0	1	0	1	Push register Pair B & C on stack	11
PUSH D	1	1	0	1	0	1	0	1	Push register Pair D & E on stack	11
PUSH H	1	1	1	0	0	1	0	1	Push register Pair H & L on stack	11
PUSH PSW	1	1	1	1	0	1	0	1	Push A and Flags on stack	11
POP B	1	1	0	0	0	0	0	1	Pop register Pair B & C off stack	10
POP D	1	1	0	1	0	0	0	1	Pop register Pair D & E off stack	10
POP H	1	1	1	0	0	0	0	1	Pop register Pair H & L off stack	10
POP PSW	1	1	1	1	0	0	0	1	Pop A and Flags off stack	10
XTHL	1	1	1	0	0	0	1	1	Exchange top of stack, H & L	18
SPHL	1	1	1	1	1	0	0	1	H & L to stack pointer	5
LXI SP	0	0	1	1	0	0	0	1	Load immediate stack pointer	10
INX SP	0	0	1	1	0	0	1	1	Increment stack pointer	5
DCX SP	0	0	1	1	1	0	1	1	Decrement stack pointer	5
JUMP										
JMP	1	1	0	0	0	0	1	1	Jump unconditional	10
JC	1	1	0	1	1	0	1	0	Jump on carry	10
JNC	1	1	0	1	0	0	1	0	Jump on no carry	10
JZ	1	1	0	0	1	0	1	0	Jump on zero	10
JNZ	1	1	0	0	0	0	1	0	Jump on no zero	10
JP	1	1	1	1	0	0	1	0	Jump on positive	10
JM	1	1	1	1	1	0	1	0	Jump on minus	10
JPE	1	1	1	0	1	0	1	0	Jump on parity even	10

Mnemonic	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Operations Description	Clock Cycles [2]
	Instruction Code [1]									
JPO	1	1	1	0	0	0	1	0	Jump on parity odd	10
PCHL	1	1	1	0	1	0	0	1	H & L to program counter	5
CALL										
CALL	1	1	0	0	1	1	0	1	Call unconditional	17
CC	1	1	0	1	1	1	0	0	Call on carry	11/17
CNC	1	1	0	1	0	1	0	0	Call on no carry	11/17
CZ	1	1	0	0	1	1	0	0	Call on zero	11/17
CNZ	1	1	0	0	0	1	0	0	Call on no zero	11/17
CP	1	1	1	1	0	1	0	0	Call on positive	11/17
CM	1	1	1	1	1	1	0	0	Call on minus	11/17
CPE	1	1	1	0	1	1	0	0	Call on parity even	11/17
CPO	1	1	1	0	0	1	0	0	Call on parity odd	11/17
RETURN										
RET	1	1	0	0	1	0	0	1	Return	10
RC	1	1	0	1	1	0	0	0	Return on carry	5/11
RNC	1	1	0	1	0	0	0	0	Return on no carry	5/11
RZ	1	1	0	0	1	0	0	0	Return on zero	5/11
RNZ	1	1	0	0	0	0	0	0	Return on no zero	5/11
RP	1	1	1	1	0	0	0	0	Return on positive	5/11
RM	1	1	1	1	1	0	0	0	Return on minus	5/11
RPE	1	1	1	0	1	0	0	0	Return on parity even	5/11
RPO	1	1	1	0	0	0	0	0	Return on parity odd	5/11
RESTART										
RST	1	1	A	A	A	1	1	1	Restart	11
INCREMENT AND DECREMENT										
INR r	0	0	D	D	D	1	0	0	Increment register	5
DCR r	0	0	D	D	D	1	0	1	Decrement register	5
INR M	0	0	1	1	0	1	0	0	Increment memory	10
DCR M	0	0	1	1	0	1	0	1	Decrement memory	10
INX B	0	0	0	0	0	0	1	1	Increment B & C registers	5
INX D	0	0	0	1	0	0	1	1	Increment D & E registers	5
INX H	0	0	1	0	0	0	1	1	Increment H & L registers	5
DCX B	0	0	0	0	1	0	1	1	Decrement B & C	5
DCX D	0	0	0	1	1	0	1	1	Decrement D & E	5
DCX H	0	0	1	0	1	0	1	1	Decrement H & L	5
ADD										
ADD r	1	0	0	0	0	S	S	S	Add register to A	4
ADC r	1	0	0	0	1	S	S	S	Add register to A with carry	4
ADD M	1	0	0	0	0	1	1	0	Add memory to A	7
ADC M	1	0	0	0	1	1	1	0	Add memory to A with carry	7
ADI	1	1	0	0	0	1	1	0	Add immediate to A	7
ACI	1	1	0	0	1	1	1	0	Add immediate to A with carry	7
DAD B	0	0	0	0	1	0	0	1	Add B & C to H & L	10
DAD D	0	0	0	1	1	0	0	1	Add D & E to H & L	10
DAD H	0	0	1	0	1	0	0	1	Add H & L to H & L	10
DAD SP	0	0	1	1	1	0	0	1	Add stack pointer to H & L	10

Summary of Processor Instructions (Cont.)

Mnemonic	Instruction Code [1] D7	D6	D5	D4	D3	D2	D1	D0	Operations Description	Clock Cycles [2]
SUBTRACT										
SUB r	1	0	0	1	0	S	S	S	Subtract register from A	4
SBB r	1	0	0	1	1	S	S	S	Subtract register from A with borrow	4
SUB M	1	0	0	1	0	1	1	0	Subtract memory from A	7
SBB M	1	0	0	1	1	1	1	0	Subtract memory from A with borrow	7
SUI	1	1	0	1	0	1	1	0	Subtract immediate from A	7
SBI	1	1	0	1	1	1	1	0	Subtract immediate from A with borrow	7
LOGICAL										
ANA r	1	0	1	0	0	S	S	S	And register with A	4
XRA r	1	0	1	0	1	S	S	S	Exclusive Or register with A	4
ORA r	1	0	1	1	0	S	S	S	Or register with A	4
CMP r	1	0	1	1	1	S	S	S	Compare register with A	4
ANA M	1	0	1	0	0	1	1	0	And memory with A	7
XRA M	1	0	1	0	1	1	1	0	Exclusive Or memory with A	7
ORA M	1	0	1	1	0	1	1	0	Or memory with A	7
CMP M	1	0	1	1	1	1	1	0	Compare memory with A	7
ANI	1	1	1	0	0	1	1	0	And immediate with A	7
XRI	1	1	1	0	1	1	1	0	Exclusive Or immediate with A	7
ORI	1	1	1	1	0	1	1	0	Or immediate with A	7
CPI	1	1	1	1	1	1	1	0	Compare immediate with A	7

Mnemonic	Instruction Code [1] D7	D6	D5	D4	D3	D2	D1	D0	Operations Description	Clock Cycles [2]
ROTATE										
RLC	0	0	0	0	0	1	1	1	Rotate A left	4
RRC	0	0	0	0	1	1	1	1	Rotate A right	4
RAL	0	0	0	1	0	1	1	1	Rotate A left through carry	4
RAR	0	0	0	1	1	1	1	1	Rotate A right through carry	4
SPECIALS										
CMA	0	0	1	0	1	1	1	1	Complement A	4
STC	0	0	1	1	0	1	1	1	Set carry	4
CMC	0	0	1	1	1	1	1	1	Complement carry	4
DAA	0	0	1	0	0	1	1	1	Decimal adjust A	4
INPUT/OUTPUT										
IN	1	1	0	1	1	0	1	1	Input	10
OUT	1	1	0	1	0	0	1	1	Output	10
CONTROL										
EI	1	1	1	1	1	0	1	1	Enable Interrupts	4
DI	1	1	1	1	0	0	1	1	Disable Interrupt	4
NOP	0	0	0	0	0	0	0	0	No-operation	4
HLT	0	1	1	1	0	1	1	0	Halt	7

NOTES:

1. DDD or SSS: B=000, C=001, D=010, E=011, H=100, L=101, Memory=110, A=111.
2. Two possible cycle times (6/12) indicate instruction cycles dependent on condition flags.

intel®

8085AH/8085AH-2/8085AH-1
8-BIT HMOS MICROPROCESSORS

- **Single +5V Power Supply with 10% Voltage Margins**
- **3 MHz, 5 MHz and 6 MHz Selections Available**
- **20% Lower Power Consumption than 8085A for 3 MHz and 5 MHz**
- **1.3 μs Instruction Cycle (8085AH); 0.8 μs (8085AH-2); 0.67 μs (8085AH-1)**
- **100% Software Compatible with 8080A**
- **On-Chip Clock Generator (with External Crystal, LC or RC Network)**
- **On-Chip System Controller; Advanced Cycle Status Information Available for Large System Control**
- **Four Vectored Interrupt Inputs (One is Non-Maskable) Plus an 8080A-Compatible Interrupt**
- **Serial In/Serial Out Port**
- **Decimal, Binary and Double Precision Arithmetic**
- **Direct Addressing Capability to 64K Bytes of Memory**
- **Available in 40-Lead Cerdip and Plastic Packages**
  (See Packaging Spec, Order #231369)

The Intel® 8085AH is a complete 8 bit parallel Central Processing Unit (CPU) implemented in N-channel, depletion load, silicon gate technology (HMOS). Its instruction set is 100% software compatible with the 8080A microprocessor, and it is designed to improve the present 8080A's performance by higher system speed. Its high level of system integration allows a minimum system of three IC's [8085AH (CPU), 8156H (RAM/IO) and 8755A (EPROM/IO)] while maintaining total system expandability. The 8085AH-2 and 8085AH-1 are faster versions of the 8085 AH.

The 8085AH incorporates all of the features that the 8224 (clock generator) and 8228 (system controller) provided for the 8080A, thereby offering a higher level of system integration.

The 8085AH uses a multiplexed data bus. The address is split between the 8 bit address bus and the 8 bit data bus. The on-chip address latches of 8155H/8156H/8755A memory products allow a direct interface with the 8085AH.

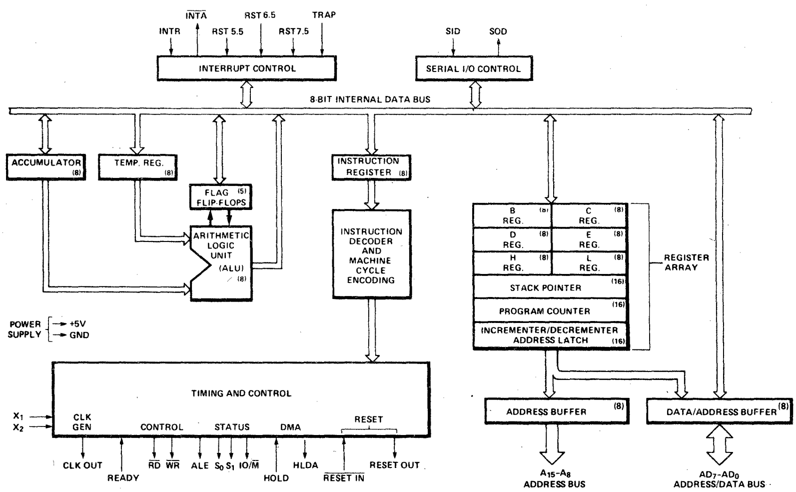


Figure 1. 8085AH CPU Functional Block Diagram

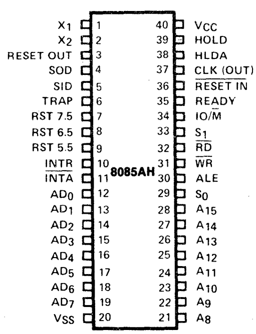


Figure 2. 8085AH Pin Configuration

Table 1. Pin Description

Symbol	Type	Name and Function
A_8–A_{15}	O	**Address Bus:** The most significant 8 bits of the memory address or the 8 bits of the I/O address, 3-stated during Hold and Halt modes and during RESET.
AD_{0-7}	I/O	**Multiplexed Address/Data Bus:** Lower 8 bits of the memory address (or I/O address) appear on the bus during the first clock cycle (T state) of a machine cycle. It then becomes the data bus during the second and third clock cycles.
ALE	O	**Address Latch Enable:** It occurs during the first clock state of a machine cycle and enables the address to get latched into the on-chip latch of peripherals. The falling edge of ALE is set to guarantee setup and hold times for the address information. The falling edge of ALE can also be used to strobe the status information. ALE is never 3-stated.
S_0, S_1, and IO/$\overline{M}$	O	**Machine Cycle Status:** **IO/$\overline{M}$ S_1 S_0 Status** 0 0 1 Memory write 0 1 0 Memory read 1 0 1 I/O write 1 1 0 I/O read 0 1 1 Opcode fetch 1 1 1 Interrupt Acknowledge * 0 0 Halt * X X Hold * X X Reset * = 3-state (high impedance) X = unspecified S_1 can be used as an advanced R/$\overline{W}$ status. IO/$\overline{M}$, S_0 and S_1 become valid at the beginning of a machine cycle and remain stable throughout the cycle. The falling edge of ALE may be used to latch the state of these lines.
$\overline{RD}$	O	**Read Control:** A low level on $\overline{RD}$ indicates the selected memory or I/O device is to be read and that the Data Bus is available for the data transfer, 3-stated during Hold and Halt modes and during RESET.
$\overline{WR}$	O	**Write Control:** A low level on $\overline{WR}$ indicates the data on the Data Bus is to be written into the selected memory or I/O location. Data is set up at the trailing edge of $\overline{WR}$. 3-stated during Hold and Halt modes and during RESET.
READY	I	**Ready:** If READY is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data. If READY is low, the cpu will wait an integral number of clock cycles for READY to go high before completing the read or write cycle. READY must conform to specified setup and hold times.
HOLD	I	**Hold:** Indicates that another master is requesting the use of the address and data buses. The cpu, upon receiving the hold request, will relinquish the use of the bus as soon as the completion of the current bus transfer. Internal processing can continue. The processor can regain the bus only after the HOLD is removed. When the HOLD is acknowledged, the Address, Data $\overline{RD}$, $\overline{WR}$, and IO/$\overline{M}$ lines are 3-stated.
HLDA	O	**Hold Acknowledge:** Indicates that the cpu has received the HOLD request and that it will relinquish the bus in the next clock cycle. HLDA goes low after the Hold request is removed. The cpu takes the bus one half clock cycle after HLDA goes low.
INTR	I	**Interrupt Request:** Is used as a general purpose interrupt. It is sampled only during the next to the last clock cycle of an instruction and during Hold and Halt states. If it is active, the Program Counter (PC) will be inhibited from incrementing and an $\overline{INTA}$ will be issued. During this cycle a RESTART or CALL instruction can be inserted to jump to the interrupt service routine. The INTR is enabled and disabled by software. It is disabled by Reset and immediately after an interrupt is accepted.
$\overline{INTA}$	O	**Interrupt Acknowledge:** Is used instead of (and has the same timing as) $\overline{RD}$ during the Instruction cycle after an INTR is accepted. It can be used to activate an 8259A Interrupt chip or some other interrupt port.
RST 5.5 RST 6.5 RST 7.5	I	**Restart Interrupts:** These three inputs have the same timing as INTR except they cause an internal RESTART to be automatically inserted. The priority of these interrupts is ordered as shown in Table 2. These interrupts have a higher priority than INTR. In addition, they may be individually masked out using the SIM instruction.

Table 1. Pin Description (Continued)

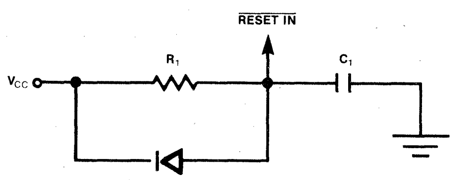
Symbol	Type	Name and Function
TRAP	I	**Trap:** Trap interrupt is a non-maskable RESTART interrupt. It is recognized at the same time as INTR or RST 5.5-7.5. It is unaffected by any mask or Interrupt Enable. It has the highest priority of any interrupt. (See Table 2.)
$\overline{\text{RESET IN}}$	I	**Reset In:** Sets the Program Counter to zero and resets the Interrupt Enable and HLDA flip-flops. The data and address buses and the control lines are 3-stated during RESET and because of the asynchronous nature of RESET, the processor's internal registers and flags may be altered by RESET with unpredictable results. $\overline{\text{RESET IN}}$ is a Schmitt-triggered input, allowing connection to an R-C network for power-on RESET delay (see Figure 3). Upon power-up, $\overline{\text{RESET IN}}$ must remain low for at least 10 ms after minimum V_{CC} has been reached. For proper reset operation after the power-up duration, $\overline{\text{RESET IN}}$ should be kept low a minimum of three clock periods. The CPU is held in the reset condition as long as $\overline{\text{RESET IN}}$ is applied.

Symbol	Type	Name and Function
RESET OUT	O	**Reset Out:** Reset Out indicates cpu is being reset. Can be used as a system reset. The signal is synchronized to the processor clock and lasts an integral number of clock periods.
X_1, X_2	I	**X_1 and X_2:** Are connected to a crystal, LC, or RC network to drive the internal clock generator. X_1 can also be an external clock input from a logic gate. The input frequency is divided by 2 to give the processor's internal operating frequency.
CLK	O	**Clock:** Clock output for use as a system clock. The period of CLK is twice the X_1, X_2 input period.
SID	I	**Serial Input Data Line:** The data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.
SOD	O	**Serial Output Data Line:** The output SOD is set or reset as specified by the SIM instruction.
V_{CC}		**Power:** +5 volt supply.
V_{SS}		**Ground:** Reference.

Table 2. Interrupt Priority, Restart Address, and Sensitivity

Name	Priority	Address Branched To (1) When Interrupt Occurs	Type Trigger
TRAP	1	24H	Rising edge AND high level until sampled.
RST 7.5	2	3CH	Rising edge (latched).
RST 6.5	3	34H	High level until sampled.
RST 5.5	4	2CH	High level until sampled.
INTR	5	See Note (2).	High level until sampled.

NOTES:
1. The processor pushes the PC on the stack before branching to the indicated address.
2. The address branched to depends on the instruction provided to the cpu when the interrupt is acknowledged.



TYPICAL POWER-ON RESET RC VALUES*
R_1 = 75 KΩ
C_1 = 1 μF
*VALUES MAY HAVE TO VARY DUE TO APPLIED POWER SUPPLY RAMP UP TIME.

Figure 3. Power-On Reset Circuit

FUNCTIONAL DESCRIPTION

The 8085AH is a complete 8-bit parallel central processor. It is designed with N-channel, depletion load, silicon gate technology (HMOS), and requires a single +5 volt supply. Its basic clock speed is 3 MHz (8085AH), 5 MHz (8085AH-2), or 6 MHz (8085AH-1), thus improving on the present 8080A's performance with higher system speed. Also it is designed to fit into a minimum system of three IC's: The CPU (8085AH), a RAM/IO (8156H), and an EPROM/IO chip (8755A).

The 8085AH has twelve addressable 8-bit registers. Four of them can function only as two 16-bit register pairs. Six others can be used interchangeably as 8-bit registers or as 16-bit register pairs. The 8085AH register set is as follows:

Mnemonic	Register	Contents
ACC or A	Accumulator	8 bits
PC	Program Counter	16-bit address
BC,DE,HL	General-Purpose Registers; data pointer (HL)	8 bits x 6 or 16 bits x 3
SP	Stack Pointer	16-bit address
Flags or F	Flag Register	5 flags (8-bit space)

The 8085AH uses a multiplexed Data Bus. The address is split between the higher 8-bit Address Bus and the lower 8-bit Address/Data Bus. During the first T state (clock cycle) of a machine cycle the low order address is sent out on the Address/Data bus. These lower 8 bits may be latched externally by the Address Latch Enable signal (ALE). During the rest of the machine cycle the data bus is used for memory or I/O data.

The 8085AH provides $\overline{RD}$, $\overline{WR}$, S_0, S_1, and IO/$\overline{M}$ signals for bus control. An Interrupt Acknowledge signal ($\overline{INTA}$) is also provided. HOLD and all Interrupts are synchronized with the processor's internal clock. The 8085AH also provides Serial Input Data (SID) and Serial Output Data (SOD) lines for simple serial interface.

In addition to these features, the 8085AH has three maskable, vector interrupt pins, one nonmaskable TRAP interrupt, and a bus vectored interrupt, INTR.

INTERRUPT AND SERIAL I/O

The 8085AH has 5 interrupt inputs: INTR, RST 5.5, RST 6.5, RST 7.5, and TRAP. INTR is identical in function to the 8080A INT. Each of the three RESTART inputs, 5.5, 6.5, and 7.5, has a programmable mask. TRAP is also a RESTART interrupt but it is nonmaskable.

The three maskable interrupts cause the internal execution of RESTART (saving the program counter in the stack and branching to the RESTART address) if the interrupts are enabled and if the interrupt mask is not set. The nonmaskable TRAP causes the internal execution of a RESTART vector independent of the state of the interrupt enable or masks. (See Table 2.)

There are two different types of inputs in the restart interrupts. RST 5.5 and RST 6.5 are *high level-sensitive* like INTR (and INT on the 8080) and are recognized with the same timing as INTR. RST 7.5 is *rising edge-sensitive.*

For RST 7.5, only a pulse is required to set an internal flip-flop which generates the internal interrupt request (a normally high level signal with a low going pulse is recommended for highest system noise immunity). The RST 7.5 request flip-flop remains set until the request is serviced. Then it is reset automatically. This flip-flop may also be reset by using the SIM instruction or by issuing a $\overline{RESET\ IN}$ to the 8085AH. The RST 7.5 internal flip-flop will be set by a pulse on the RST 7.5 pin even when the RST 7.5 interrupt is masked out.

The status of the three RST interrupt masks can only be affected by the SIM instruction and $\overline{RESET\ IN}$. (See SIM, Chapter 5 of the MCS-80/85 User's Manual.)

The interrupts are arranged in a fixed priority that determines which interrupt is to be recognized if more than one is pending as follows: TRAP—highest priority, RST 7.5, RST 6.5, RST 5.5, INTR—lowest priority. This priority scheme does not take into account the priority of a routine that was started by a higher priority interrupt. RST 5.5 can interrupt an RST 7.5 routine if the interrupts are re-enabled before the end of the RST 7.5 routine.

The TRAP interrupt is useful for catastrophic events such as power failure or bus error. The TRAP input is recognized just as any other interrupt but has the highest priority. It is not affected by any flag or mask. The TRAP input is both *edge and level sensitive*. The TRAP input must go high and remain high until it is acknowledged. It will not be recognized again until it goes low, then high again. This avoids any false triggering due to noise or logic glitches. Figure 4 illustrates the TRAP interrupt request circuitry within the 8085AH. Note that the servicing of any interrupt (TRAP, RST 7.5, RST 6.5, RST 5.5, INTR) disables all future interrupts (except TRAPs) until an EI instruction is executed.

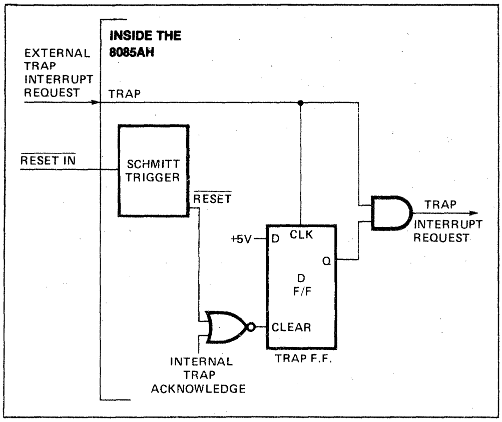


Figure 4. TRAP and RESET IN Circuit

The TRAP interrupt is special in that it disables interrupts, but preserves the previous interrupt enable status. Performing the first RIM instruction following a TRAP interrupt allows you to determine whether interrupts were enabled or disabled prior to the TRAP. All subsequent RIM instructions provide current interrupt enable status. Performing a RIM instruction following INTR, or RST 5.5–7.5 will provide current Interrupt Enable status, revealing that Interrupts are disabled. See the description of the RIM instruction in the MCS-80/85 Family User's Manual.

The serial I/O system is also controlled by the RIM and SIM instructions. SID is read by RIM, and SIM sets the SOD data.

DRIVING THE X_1 AND X_2 INPUTS

You may drive the clock inputs of the 8085AH, 8085AH-2, or 8085AH-1 with a crystal, an LC tuned circuit, an RC network, or an external clock source. The crystal frequency must be at least 1 MHz, and must be twice the desired internal clock frequency; hence, the 8085AH is operated with a 6 MHz crystal (for 3 MHz clock), the 8085AH-2 operated with a 10 MHz crystal (for 5 MHz clock), and the 8085AH-1 can be operated with a 12 MHz crystal (for 6 MHz clock). If a crystal is used, it must have the following characteristics:

Parallel resonance at twice the clock frequency desired
C_L (load capacitance) $\leq$ 30 pF
C_S (shunt capacitance) $\leq$ 7 pF
R_S (equivalent shunt resistance) $\leq$ 75 Ohms
Drive level: 10 mW
Frequency tolerance: $\pm$.005% (suggested)

Note the use of the 20 pF capacitor between X_2 and ground. This capacitor is required with crystal frequencies below 4 MHz to assure oscillator startup at the correct frequency. A parallel-resonant LC circuit may be used as the frequency-determining network for the 8085AH, providing that its frequency tolerance of approximately $\pm$10% is acceptable. The components are chosen from the formula:

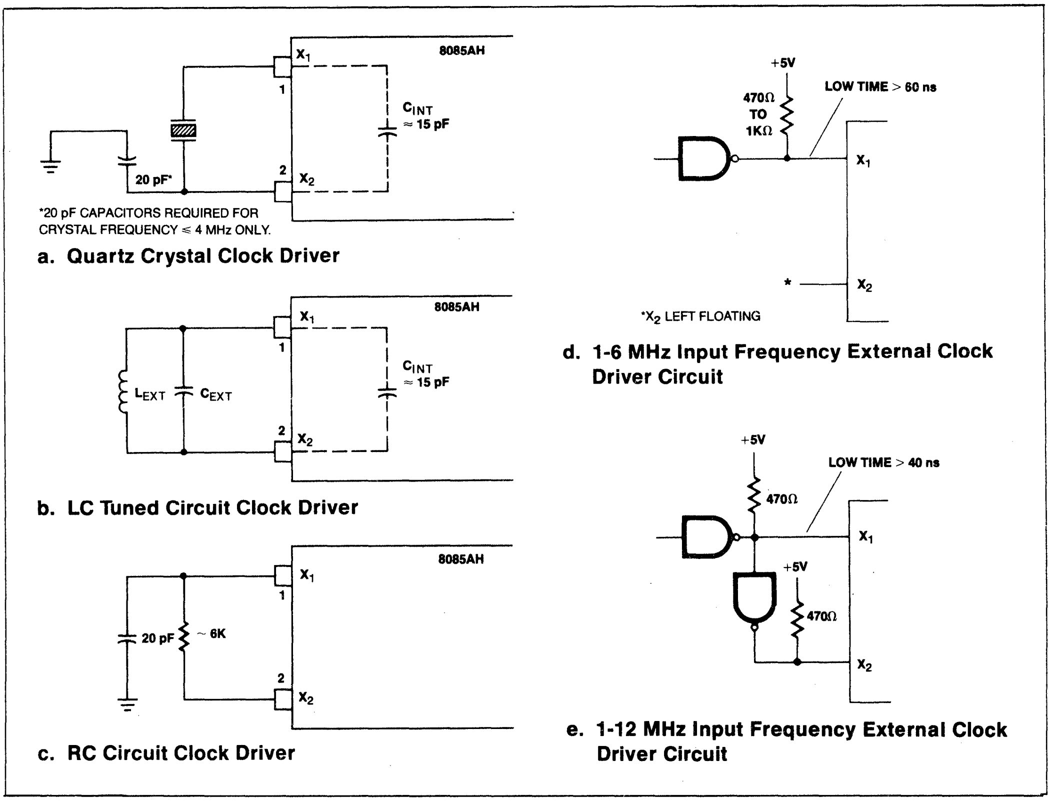
$$f = \frac{1}{2\pi\sqrt{L(C_{ext} + C_{int})}}$$

To minimize variations in frequency, it is recommended that you choose a value for C_{ext} that is at least twice that of C_{int}, or 30 pF. The use of an LC circuit is not recommended for frequencies higher than approximately 5 MHz.

An RC circuit may be used as the frequency-determining network for the 8085AH if maintaining a precise clock frequency is of no importance. Variations in the on-chip timing generation can cause a wide variation in frequency when using the RC mode. Its advantage is its low component cost. The driving frequency generated by the circuit shown is approximately 3 MHz. It is not recommended that frequencies greatly higher or lower than this be attempted.

Figure 5 shows the recommended clock driver circuits. Note in D and E that pullup resistors are required to assure that the high level voltage of the input is at least 4V and maximum low level voltage of 0.8V.

For driving frequencies up to and including 6 MHz you may supply the driving signal to X_1 and leave X_2 open-circuited (Figure 5D). If the driving frequency is from 6 MHz to 12 MHz, stability of the clock generator will be improved by driving both X_1 and X_2 with a push-pull source (Figure 5E). To prevent self-oscillation of the 8085AH, be sure that X_2 is not coupled back to X_1 through the driving circuit.



a. Quartz Crystal Clock Driver

b. LC Tuned Circuit Clock Driver

c. RC Circuit Clock Driver

d. 1-6 MHz Input Frequency External Clock Driver Circuit

e. 1-12 MHz Input Frequency External Clock Driver Circuit

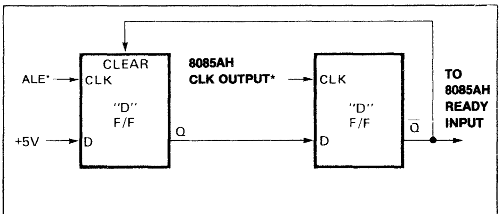
Figure 5. Clock Driver Circuits

GENERATING AN 8085AH WAIT STATE

If your system requirements are such that slow memories or peripheral devices are being used, the circuit shown in Figure 6 may be used to insert one WAIT state in each 8085AH machine cycle.

The D flip-flops should be chosen so that

- CLK is rising edge-triggered
- CLEAR is low-level active.



*ALE AND CLK (OUT) SHOULD BE BUFFERED IF CLK INPUT OF LATCH EXCEEDS 8085AH IOL OR IOH.

Figure 6. Generation of a Wait State for 8085AH CPU

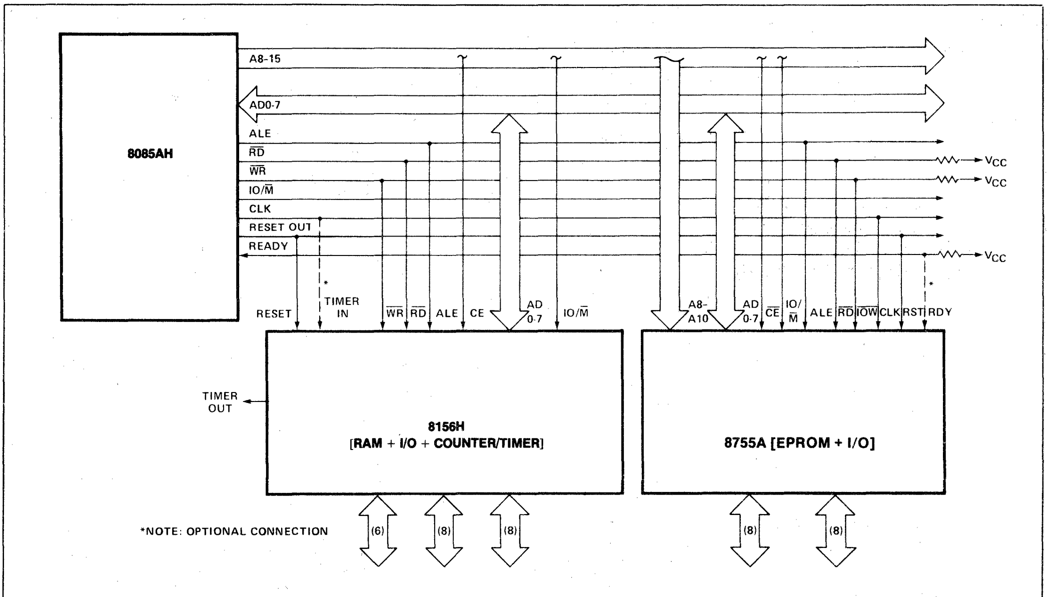


Figure 8. MCS-85® Minimum System (Memory Mapped I/O)

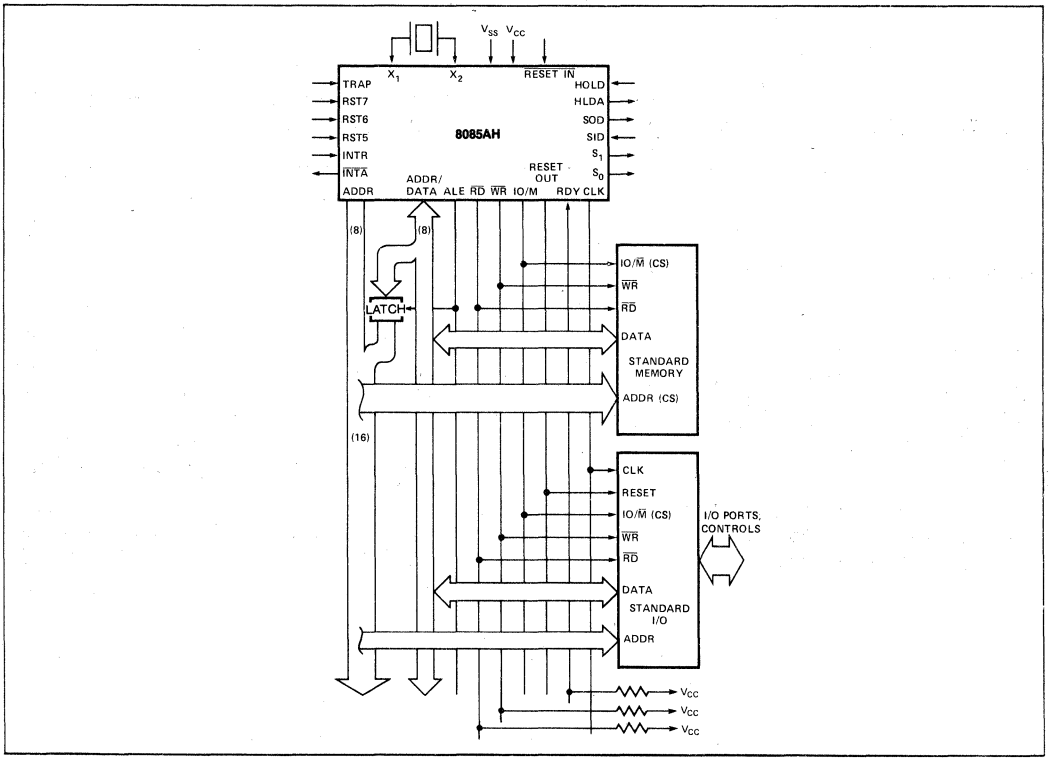


Figure 9. MCS-85® System (Using Standard Memories)

As in the 8080, the READY line is used to extend the read and write pulse lengths so that the 8085AH can be used with slow memory. HOLD causes the CPU to relinquish the bus when it is through with it by floating the Address and Data Buses.

SYSTEM INTERFACE

The 8085AH family includes memory components, which are directly compatible to the 8085AH CPU. For example, a system consisting of the three chips, 8085AH, 8156H, and 8755A will have the following features:

- 2K Bytes EPROM
- 256 Bytes RAM
- 1 Timer/Counter
- 4 8-bit I/O Ports
- 1 6-bit I/O Port
- 4 Interrupt Levels
- Serial In/Serial Out Ports

This minimum system, using the standard I/O technique is as shown in Figure 7.

In addition to standard I/O, the memory mapped I/O offers an efficient I/O addressing technique. With this technique, an area of memory address space is assigned for I/O address, thereby, using the memory address for I/O manipulation. Figure 8 shows the system configuration of Memory Mapped I/O using 8085AH.

The 8085AH CPU can also interface with the standard memory that does *not* have the multiplexed address/data bus. It will require a simple 8-bit latch as shown in Figure 9.

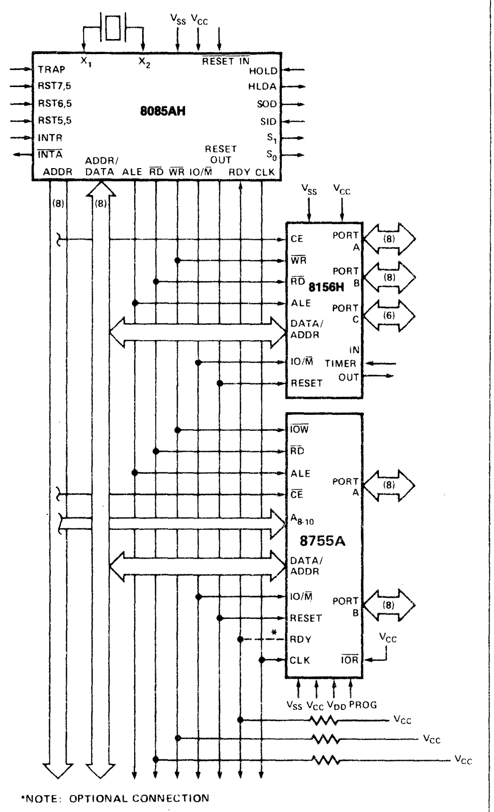


Figure 7. 8085AH Minimum System (Standard I/O Technique)

BASIC SYSTEM TIMING

The 8085AH has a multiplexed Data Bus. ALE is used as a strobe to sample the lower 8-bits of address on the Data Bus. Figure 10 shows an instruction fetch, memory read and I/O write cycle (as would occur during processing of the OUT instruction). Note that during the I/O write and read cycle that the I/O port address is copied on both the upper and lower half of the address.

There are seven possible types of machine cycles. Which of these seven takes place is defined by the status of the three status lines ($IO/\overline{M}$, S_1, S_0) and the three control signals ($\overline{RD}$, $\overline{WR}$, and $\overline{INTA}$). (See Table 3.) The status lines can be used as advanced controls (for device selection, for example), since they become active at the T_1 state, at the outset of each machine cycle. Control lines $\overline{RD}$ and $\overline{WR}$ become active later, at the time when the transfer of data is to take place, so are used as command lines.

A machine cycle normally consists of three T states, with the exception of OPCODE FETCH, which normally has either four or six T states (unless WAIT or HOLD states are forced by the receipt of $\overline{READY}$ or HOLD inputs). Any T state must be one of ten possible states, shown in Table 4.

Table 3. 8085AH Machine Cycle Chart

MACHINE CYCLE		STATUS			CONTROL		
		$IO/\overline{M}$	S1	S0	$\overline{RD}$	$\overline{WR}$	$\overline{INTA}$
OPCODE FETCH	(OF)	0	1	1	0	1	1
MEMORY READ	(MR)	0	1	0	0	1	1
MEMORY WRITE	(MW)	0	0	1	1	0	1
I/O READ	(IOR)	1	1	0	0	1	1
I/O WRITE	(IOW)	1	0	1	1	0	1
ACKNOWLEDGE OF INTR	(INA)	1	1	1	1	1	0
BUS IDLE	(BI): DAD	0	1	0	1	1	1
	ACK. OF RST,TRAP	1	1	1	1	1	1
	HALT	TS	0	0	TS	TS	1

Table 4. 8085AH Machine State Chart

Machine State	Status & Buses				Control		
	S1,S0	$IO/\overline{M}$	A_8-A_{15}	AD_0-AD_7	$\overline{RD}$,$\overline{WR}$	$\overline{INTA}$	ALE
T_1	X	X	X	X	1	1	1*
T_2	X	X	X	X	X	X	0
T_{WAIT}	X	X	X	X	X	X	0
T_3	X	X	X	X	X	X	0
T_4	1	0†	X	TS	1	1	0
T_5	1	0†	X	TS	1	1	0
T_6	1	0†	X	TS	1	1	0
T_{RESET}	X	TS	TS	TS	TS	1	0
T_{HALT}	0	TS	TS	TS	TS	1	0
T_{HOLD}	X	TS	TS	TS	TS	1	0

0 = Logic "0" TS = High Impedance
1 = Logic "1" X = Unspecified

* ALE not generated during 2nd and 3rd machine cycles of DAD instruction.
† IO/M = 1 during T_4-T_6 of INA machine cycle.

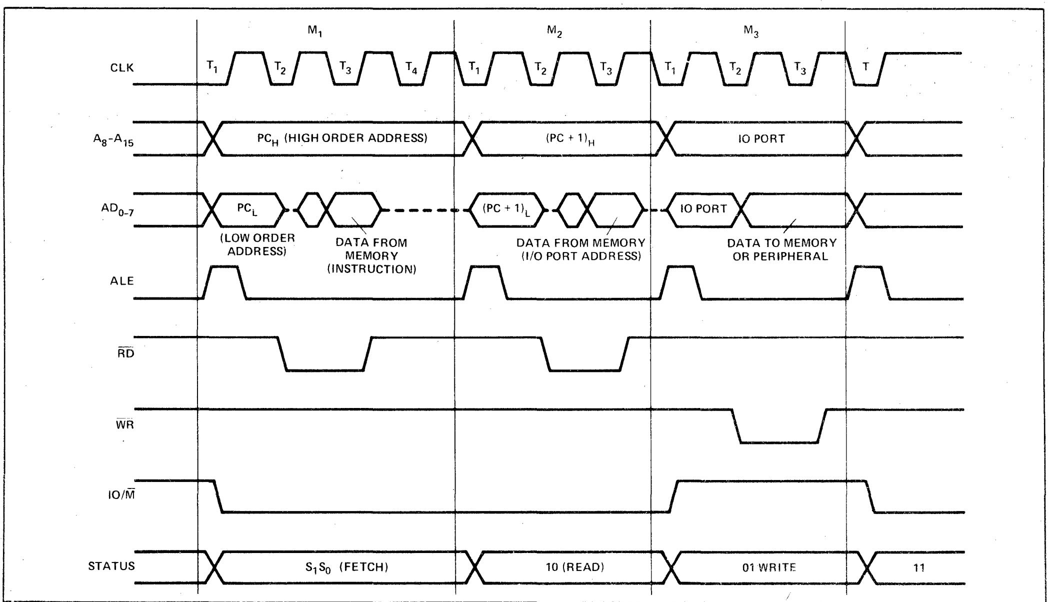


Figure 10. 8085AH Basic System Timing

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .... 0°C to 70°C
Storage Temperature .... −65°C to +150°C
Voltage on Any Pin
With Respect to Ground .... −0.5V to +7V
Power Dissipation .... 1.5 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

8085AH, 8085AH-2: (T_A = 0°C to 70°C, V_{CC} = 5V ±10%, V_{SS} =0V; unless otherwise specified)*
8085AH-1: (T_A = 0°C to 70°C, V_{CC} = 5V ±5%, V_{SS} = 0V; unless otherwise specified)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} +0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400μA
I_{CC}	Power Supply Current		135	mA	8085AH, 8085AH-2
			200	mA	8085AH-1 (Preliminary)
I_{IL}	Input Leakage		±10	μA	$0 \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage		±10	μA	$0.45V \leq V_{OUT} \leq V_{CC}$
V_{ILR}	Input Low Level, RESET	−0.5	+0.8	V	
V_{IHR}	Input High Level, RESET	2.4	V_{CC} +0.5	V	
V_{HY}	Hysteresis, RESET	0.15		V	

A.C. CHARACTERISTICS

8085AH, 8085AH-2: (T_A = 0°C to 70°C, V_{CC} = 5V ±10%, V_{SS} = OV)*
8085AH-1: (T_A = 0°C to 70°C, V_{CC} = 5V ±5%, V_{SS} = 0V)

Symbol	Parameter	8085AH[2] (Final)		8085AH-2[2] (Final)		8085AH-1 (Preliminary)		Units
		Min.	Max.	Min.	Max.	Min.	Max.	
t_{CYC}	CLK Cycle Period	320	2000	200	2000	167	2000	ns
t_1	CLK Low Time (Standard CLK Loading)	80		40		20		ns
t_2	CLK High Time (Standard CLK Loading)	120		70		50		ns
t_r, t_f	CLK Rise and Fall Time		30		30		30	ns
t_{XKR}	X_1 Rising to CLK Rising	20	120	20	100	20	100	ns
t_{XKF}	X_1 Rising to CLK Falling	20	150	20	110	20	110	ns
t_{AC}	A_{8-15} Valid to Leading Edge of Control[1]	270		115		70		ns
t_{ACL}	A_{0-7} Valid to Leading Edge of Control	240		115		60		ns
t_{AD}	A_{0-15} Valid to Valid Data In		575		350		225	ns
t_{AFR}	Address Float After Leading Edge of $\overline{READ}$ ($\overline{INTA}$)		0		0		0	ns
t_{AL}	A_{8-15} Valid Before Trailing Edge of ALE [1]	115		50		25		ns

***Note:** For Extended Temperature EXPRESS use M8085AH Electricals Parameters.

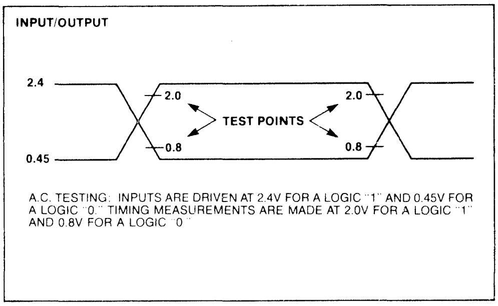
A.C. CHARACTERISTICS (Continued)

Symbol	Parameter	8085AH[2] (Final)		8085AH-2[2] (Final)		8085AH-1 (Preliminary)		Units
		Min.	Max.	Min.	Max.	Min.	Max.	
t_{ALL}	A_{0-7} Valid Before Trailing Edge of ALE	90		50		25		ns
t_{ARY}	READY Valid from Address Valid		220		100		40	ns
t_{CA}	Address (A_{8-15}) Valid After Control	120		60		30		ns
t_{CC}	Width of Control Low ($\overline{RD}$, $\overline{WR}$, $\overline{INTA}$) Edge of ALE	400		230		150		ns
t_{CL}	Trailing Edge of Control to Leading Edge of ALE	50		25		0		ns
t_{DW}	Data Valid to Trailing Edge of $\overline{WRITE}$	420		230		140		ns
t_{HABE}	HLDA to Bus Enable		210		150		150	ns
t_{HABF}	Bus Float After HLDA		210		150		150	ns
t_{HACK}	HLDA Valid to Trailing Edge of CLK	110		40		0		ns
t_{HDH}	HOLD Hold Time	0		0		0		ns
t_{HDS}	HOLD Setup Time to Trailing Edge of CLK	170		120		120		ns
t_{INH}	INTR Hold Time	0		0		0		ns
t_{INS}	INTR, RST, and TRAP Setup Time to Falling Edge of CLK	160		150		150		ns
t_{LA}	Address Hold Time After ALE	100		50		20		ns
t_{LC}	Trailing Edge of ALE to Leading Edge of Control	130		60		25		ns
t_{LCK}	ALE Low During CLK High	100		50		15		ns
t_{LDR}	ALE to Valid Data During Read		460		270		175	ns
t_{LDW}	ALE to Valid Data During Write		200		140		110	ns
t_{LL}	ALE Width	140		80		50		ns
t_{LRY}	ALE to READY Stable		110		30		10	ns
t_{RAE}	Trailing Edge of $\overline{READ}$ to Re-Enabling of Address	150		90		50		ns
t_{RD}	$\overline{READ}$ (or $\overline{INTA}$) to Valid Data		300		150		75	ns
t_{RV}	Control Trailing Edge to Leading Edge of Next Control	400		220		160		ns
t_{RDH}	Data Hold Time After $\overline{READ}$ $\overline{INTA}$	0		0		0		ns
t_{RYH}	READY Hold Time	0		0		5		ns
t_{RYS}	READY Setup Time to Leading Edge of CLK	110		100		100		ns
t_{WD}	Data Valid After Trailing Edge of $\overline{WRITE}$	100		60		30		ns
t_{WDL}	LEADING Edge of $\overline{WRITE}$ to Data Valid		40		20		30	ns

NOTES:

1. A_8–A_{15} address Specs apply IO/$\overline{M}$, S_0, and S_1 except A_8–A_{15} are undefined during T_4–T_6 of OF cycle whereas IO/$\overline{M}$, S_0, and S_1 are stable.
2. *Test Conditions:* t_{CYC} = 320 ns (8085AH)/200 ns (8085AH-2);/ 167 ns (8085AH-1); C_L = 150 pF.
3. For all output timing where $C_L \neq 150$ pF use the following correction factors:
   $25 \text{ pF} \leq C_L < 150 \text{ pF}$: −0.10 ns/pF
   $150 \text{ pF} < C_L \leq 300 \text{ pF}$: +0.30 ns/pF
4. Output timings are measured with purely capacitive load.
5. To calculate timing specifications at other values of t_{CYC} use Table 5.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING LOAD CIRCUIT

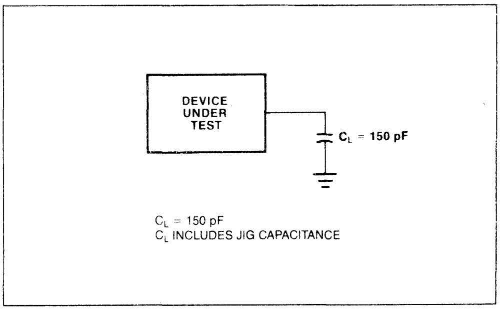


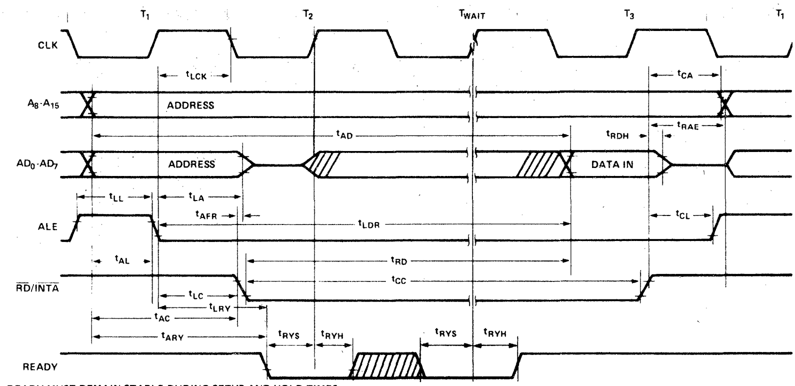
Table 5. Bus Timing Specification as a T_{CYC} Dependent

Symbol	8085AH	8085AH-2	8085AH-1	
t_{AL}	(1/2) T − 45	(1/2) T − 50	(1/2) T − 58	Minimum
t_{LA}	(1/2) T − 60	(1/2) T − 50	(1/2) T − 63	Minimum
t_{LL}	(1/2) T − 20	(1/2) T − 20	(1/2) T − 33	Minimum
t_{LCK}	(1/2) T − 60	(1/2) T − 50	(1/2) T − 68	Minimum
t_{LC}	(1/2) T − 30	(1/2) T − 40	(1/2) T − 58	Minimum
t_{AD}	(5/2 + N) T − 225	(5/2 + N) T − 150	(5/2 + N) T − 192	Maximum
t_{RD}	(3/2 + N) T − 180	(3/2 + N) T − 150	(3/2 + N) T − 175	Maximum
t_{RAE}	(1/2) T − 10	(1/2) T − 10	(1/2) T − 33	Minimum
t_{CA}	(1/2) T − 40	(1/2) T − 40	(1/2) T − 53	Minimum
t_{DW}	(3/2 + N) T − 60	(3/2 + N) T − 70	(3/2 + N) T − 110	Minimum
t_{WD}	(1/2) T − 60	(1/2) T − 40	(1/2) T − 53	Minimum
t_{CC}	(3/2 + N) T − 80	(3/2 + N) T − 70	(3/2 + N) T − 100	Minimum
t_{CL}	(1/2) T − 110	(1/2) T − 75	(1/2) T − 83	Minimum
t_{ARY}	(3/2) T − 260	(3/2) T − 200	(3/2) T − 210	Maximum
t_{HACK}	(1/2) T − 50	(1/2) T − 60	(1/2) T − 83	Minimum
t_{HABF}	(1/2) T + 50	(1/2) T + 50	(1/2) T + 67	Maximum
t_{HABE}	(1/2) T + 50	(1/2) T + 50	(1/2) T + 67	Maximum
t_{AC}	(2/2) T − 50	(2/2) T − 85	(2/2) T − 97	Minimum
t_1	(1/2) T − 80	(1/2) T − 60	(1/2) T − 63	Minimum
t_2	(1/2) T − 40	(1/2) T − 30	(1/2) T − 33	Minimum
t_{RV}	(3/2) T − 80	(3/2) T− 80	(3/2) T − 90	Minimum
t_{LDR}	(4/2) T − 180	(4/2) T − 130	(4/2) T − 159	Maximum

NOTE: N is equal to the total WAIT states. T = t_{CYC}.

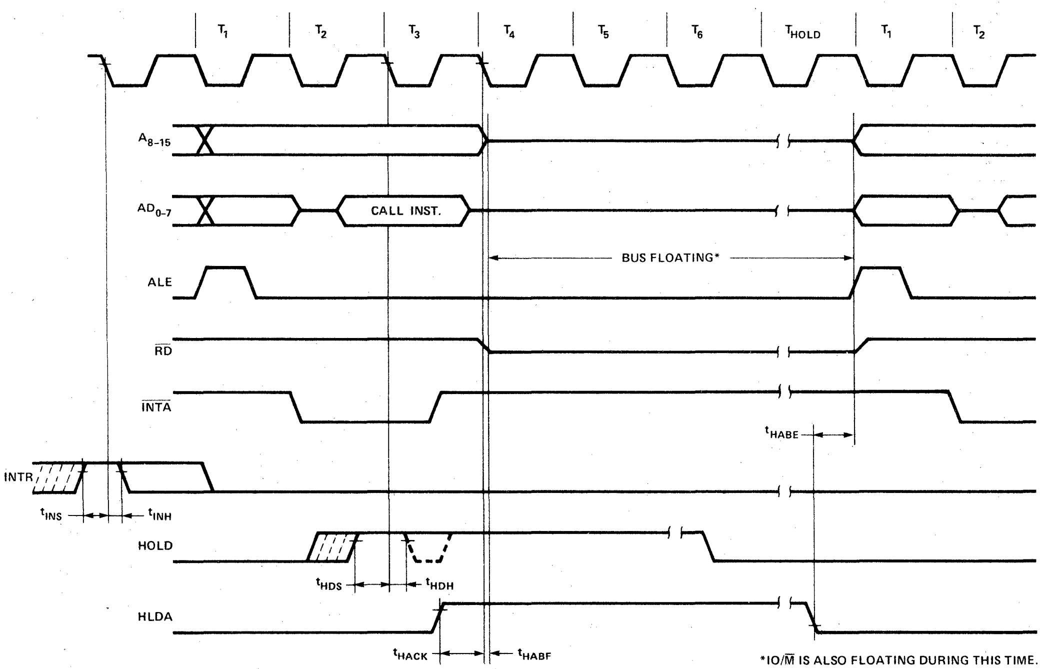
WAVEFORMS (Continued)

READ OPERATION WITH WAIT CYCLE (TYPICAL) — SAME READY TIMING APPLIES TO WRITE



NOTE 1: READY MUST REMAIN STABLE DURING SETUP AND HOLD TIMES.

INTERRUPT AND HOLD



*IO/M IS ALSO FLOATING DURING THIS TIME.

Table 6. Instruction Set Summary

Mnemonic	Instruction Code D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	Operations Description
MOVE, LOAD, AND STORE		
MOVr1 r2	0 1 D D D S S S	Move register to register
MOV M.r	0 1 1 1 0 S S S	Move register to memory
MOV r.M	0 1 D D D 1 1 0	Move memory to register
MVI r	0 0 D D D 1 1 0	Move immediate register
MVI M	0 0 1 1 0 1 1 0	Move immediate memory
LXI B	0 0 0 0 0 0 0 1	Load immediate register Pair B & C
LXI D	0 0 0 1 0 0 0 1	Load immediate register Pair D & E
LXI H	0 0 1 0 0 0 0 1	Load immediate register Pair H & L
STAX B	0 0 0 0 0 0 1 0	Store A indirect
STAX D	0 0 0 1 0 0 1 0	Store A indirect
LDAX B	0 0 0 0 1 0 1 0	Load A indirect
LDAX D	0 0 0 1 1 0 1 0	Load A indirect
STA	0 0 1 1 0 0 1 0	Store A direct
LDA	0 0 1 1 1 0 1 0	Load A direct
SHLD	0 0 1 0 0 0 1 0	Store H & L direct
LHLD	0 0 1 0 1 0 1 0	Load H & L direct
XCHG	1 1 1 0 1 0 1 1	Exchange D & E, H & L Registers
STACK OPS		
PUSH B	1 1 0 0 0 1 0 1	Push register Pair B & C on stack
PUSH D	1 1 0 1 0 1 0 1	Push register Pair D & E on stack
PUSH H	1 1 1 0 0 1 0 1	Push register Pair H & L on stack
PUSH PSW	1 1 1 1 0 1 0 1	Push A and Flags on stack
POP B	1 1 0 0 0 0 0 1	Pop register Pair B & C off stack
POP D	1 1 0 1 0 0 0 1	Pop register Pair D & E off stack
POP H	1 1 1 0 0 0 0 1	Pop register Pair H & L off stack
POP PSW	1 1 1 1 0 0 0 1	Pop A and Flags off stack
XTHL	1 1 1 0 0 0 1 1	Exchange top of stack, H & L
SPHL	1 1 1 1 1 0 0 1	H & L to stack pointer
LXI SP	0 0 1 1 0 0 0 1	Load immediate stack pointer
INX SP	0 0 1 1 0 0 1 1	Increment stack pointer
DCX SP	0 0 1 1 1 0 1 1	Decrement stack pointer
JUMP		
JMP	1 1 0 0 0 0 1 1	Jump unconditional
JC	1 1 0 1 1 0 1 0	Jump on carry
JNC	1 1 0 1 0 0 1 0	Jump on no carry
JZ	1 1 0 0 1 0 1 0	Jump on zero
JNZ	1 1 0 0 0 0 1 0	Jump on no zero
JP	1 1 1 1 0 0 1 0	Jump on positive
JM	1 1 1 1 1 0 1 0	Jump on minus
JPE	1 1 1 0 1 0 1 0	Jump on parity even
JPO	1 1 1 0 0 0 1 0	Jump on parity odd
PCHL	1 1 1 0 1 0 0 1	H & L to program counter
CALL		
CALL	1 1 0 0 1 1 0 1	Call unconditional
CC	1 1 0 1 1 1 0 0	Call on carry
CNC	1 1 0 1 0 1 0 0	Call on no carry

Mnemonic	Instruction Code D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	Operations Description
CZ	1 1 0 0 1 1 0 0	Call on zero
CNZ	1 1 0 0 0 1 0 0	Call on no zero
CP	1 1 1 1 0 1 0 0	Call on positive
CM	1 1 1 1 1 1 0 0	Call on minus
CPE	1 1 1 0 1 1 0 0	Call on parity even
CPO	1 1 1 0 0 1 0 0	Call on parity odd
RETURN		
RET	1 1 0 0 1 0 0 1	Return
RC	1 1 0 1 1 0 0 0	Return on carry
RNC	1 1 0 1 0 0 0 0	Return on no carry
RZ	1 1 0 0 1 0 0 0	Return on zero
RNZ	1 1 0 0 0 0 0 0	Return on no zero
RP	1 1 1 1 0 0 0 0	Return on positive
RM	1 1 1 1 1 0 0 0	Return on minus
RPE	1 1 1 0 1 0 0 0	Return on parity even
RPO	1 1 1 0 0 0 0 0	Return on parity odd
RESTART		
RST	1 1 A A A 1 1 1	Restart
INPUT/OUTPUT		
IN	1 1 0 1 1 0 1 1	Input
OUT	1 1 0 1 0 0 1 1	Output
INCREMENT AND DECREMENT		
INR r	0 0 D D D 1 0 0	Increment register
DCR r	0 0 D D D 1 0 1	Decrement register
INR M	0 0 1 1 0 1 0 0	Increment memory
DCR M	0 0 1 1 0 1 0 1	Decrement memory
INX B	0 0 0 0 0 0 1 1	Increment B & C registers
INX D	0 0 0 1 0 0 1 1	Increment D & E registers
INX H	0 0 1 0 0 0 1 1	Increment H & L registers
DCX B	0 0 0 0 1 0 1 1	Decrement B & C
DCX D	0 0 0 1 1 0 1 1	Decrement D & E
DCX H	0 0 1 0 1 0 1 1	Decrement H & L
ADD		
ADD r	1 0 0 0 0 S S S	Add register to A
ADC r	1 0 0 0 1 S S S	Add register to A with carry
ADD M	1 0 C 0 0 1 1 0	Add memory to A
ADC M	1 0 0 0 1 1 1 0	Add memory to A with carry
ADI	1 1 0 0 0 1 1 0	Add immediate to A
ACI	1 1 0 0 1 1 1 0	Add immediate to A with carry
DAD B	0 0 0 0 1 0 0 1	Add B & C to H & L
DAD D	0 0 0 1 1 0 0 1	Add D & E to H & L
DAD H	0 0 1 0 1 0 0 1	Add H & L to H & L
DAD SP	0 0 1 1 1 0 0 1	Add stack pointer to H & L
SUBTRACT		
SUB r	1 0 0 1 0 S S S	Subtract register from A
SBB r	1 0 0 1 1 S S S	Subtract register from A with borrow
SUB M	1 0 0 1 0 1 1 0	Subtract memory from A
SBB M	1 0 0 1 1 1 1 0	Subtract memory from A with borrow
SUI	1 1 0 1 0 1 1 0	Subtract immediate from A
SBI	1 1 0 1 1 1 1 0	Subtract immediate from A with borrow

Table 6. Instruction Set Summary (Continued)

Mnemonic	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Operations Description
LOGICAL									
ANA r	1	0	1	0	0	S	S	S	And register with A
XRA r	1	0	1	0	1	S	S	S	Exclusive OR register with A
ORA r	1	0	1	1	0	S	S	S	OR register with A
CMP r	1	0	1	1	1	S	S	S	Compare register with A
ANA M	1	0	1	0	0	1	1	0	And memory with A
XRA M	1	0	1	0	1	1	1	0	Exclusive OR memory with A
ORA M	1	0	1	1	0	1	1	0	OR memory with A
CMP M	1	0	1	1	1	1	1	0	Compare memory with A
ANI	1	1	1	0	0	1	1	0	And immediate with A
XRI	1	1	1	0	1	1	1	0	Exclusive OR immediate with A
ORI	1	1	1	1	0	1	1	0	OR immediate with A
CPI	1	1	1	1	1	1	1	0	Compare immediate with A
ROTATE									
RLC	0	0	0	0	0	1	1	1	Rotate A left
RRC	0	0	0	0	1	1	1	1	Rotate A right
RAL	0	0	0	1	0	1	1	1	Rotate A left through carry
RAR	0	0	0	1	1	1	1	1	Rotate A right through carry

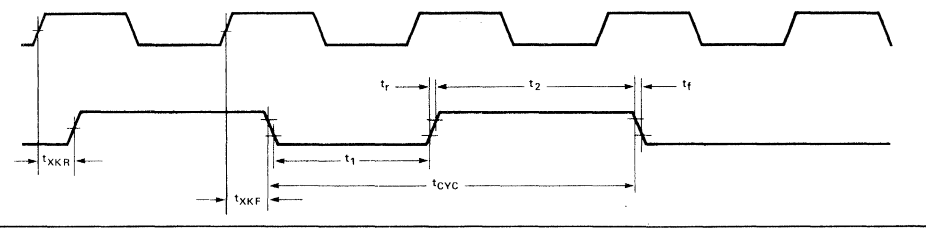
Mnemonic	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Operations Description
SPECIALS									
CMA	0	0	1	0	1	1	1	1	Complement A
STC	0	0	1	1	0	1	1	1	Set carry
CMC	0	0	1	1	1	1	1	1	Complement carry
DAA	0	0	1	0	0	1	1	1	Decimal adjust A
CONTROL									
EI	1	1	1	1	1	0	1	1	Enable Interrupts
DI	1	1	1	1	0	0	1	1	Disable Interrupt
NOP	0	0	0	0	0	0	0	0	No-operation
HLT	0	1	1	1	0	1	1	0	Halt
NEW 8085AH INSTRUCTIONS									
RIM	0	0	1	0	0	0	0	0	Read Interrupt Mask
SIM	0	0	1	1	0	0	0	0	Set Interrupt Mask

NOTES:

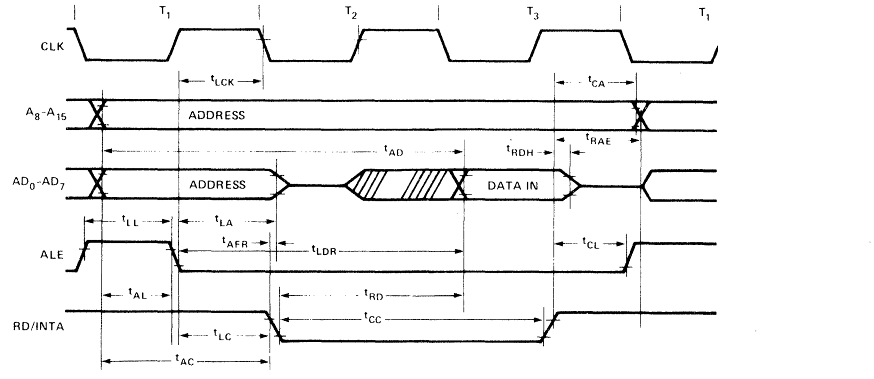
1. DDS or SSS: B 000, C 001, D 010, E011, H 100, L 101, Memory 110, A 111.
2. Two possible cycle times (6/12) indicate instruction cycles dependent on condition flags.

WAVEFORMS

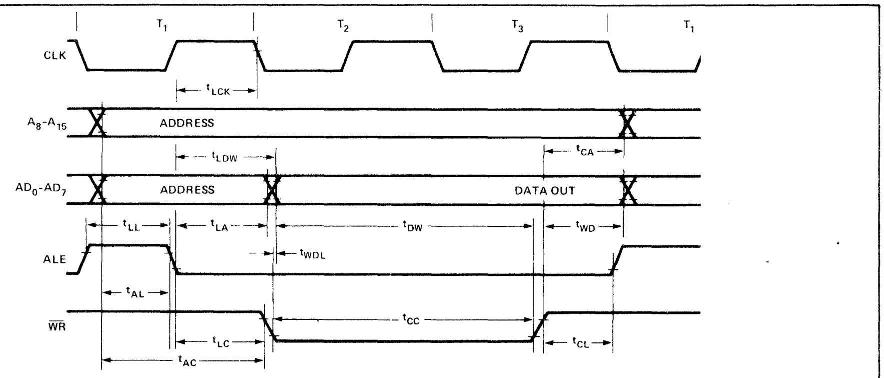
CLOCK



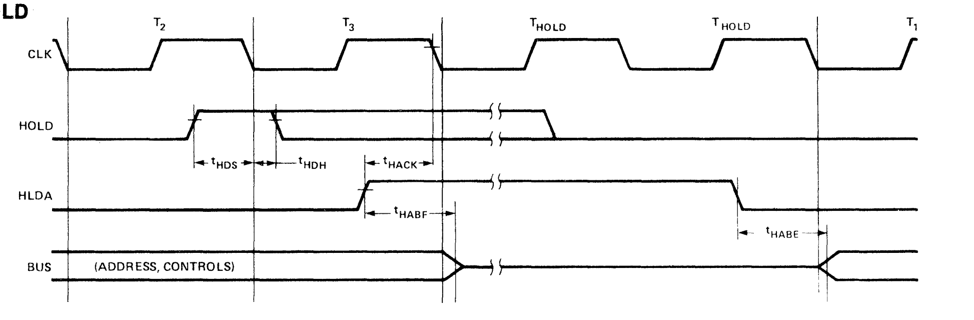
READ



WRITE



HOLD



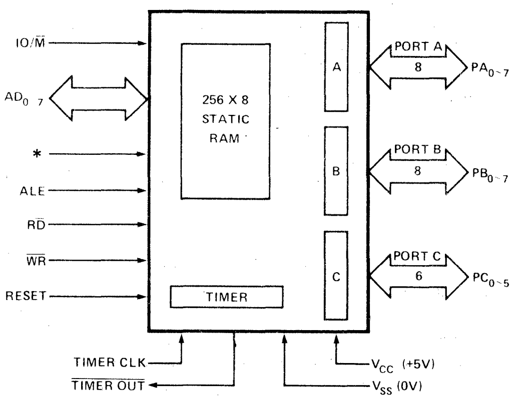
8155H/8156H/8155H-2/8156H-2
2048-BIT STATIC HMOS RAM WITH I/O PORTS AND TIMER

- **Single +5V Power Supply with 10% Voltage Margins**
- **30% Lower Power Consumption than the 8155 and 8156**
- **256 Word x 8 Bits**
- **Completely Static Operation**
- **Internal Address Latch**
- **2 Programmable 8-Bit I/O Ports**
- **1 Programmable 6-Bit I/O Port**
- **Programmable 14-Bit Binary Counter/Timer**
- **Compatible with 8085AH and 8088 CPU**
- **Multiplexed Address and Data Bus**
- **Available in EXPRESS**
  - **Standard Temperature Range**
  - **Extended Temperature Range**

The Intel® 8155H and 8156H are RAM and I/O chips implemented in N-Channel, depletion load, silicon gate technology (HMOS), to be used in the 8085AH and 8088 microprocessor systems. The RAM portion is designed with 2048 static cells organized as 256 x 8. They have a maximum access time of 400 ns to permit use with no wait states in 8085AH CPU. The 8155H-2 and 8156H-2 have maximum access times of 330 ns for use with the 8085AH-2 and the 5 MHz 8088 CPU.

The I/O portion consists of three general purpose I/O ports. One of the three ports can be programmed to be status pins, thus allowing the other two ports to operate in handshake mode.

A 14-bit programmable counter/timer is also included on chip to provide either a square wave or terminal count pulse for the CPU system depending on timer mode.



*8155H/8155H-2 = C̄Ē, 8156H/8156H-2 = CE

Figure 1. Block Diagram

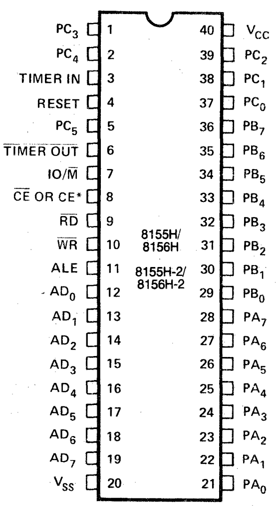


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Type	Name and Function
RESET	I	**Reset:** Pulse provided by the 8085AH to initialize the system (connect to 8085AH RESET OUT). Input high on this line resets the chip and initializes the three I/O ports to input mode. The width of RESET pulse should typically be two 8085AH clock cycle times.
AD_{0-7}	I/O	**Address/Data:** 3-state Address/Data lines that interface with the CPU lower 8-bit Address/Data Bus. The 8-bit address is latched into the address latch inside the 8155H/56H on the falling edge of ALE. The address can be either for the memory section or the I/O section depending on the $IO/\overline{M}$ input. The 8-bit data is either written into the chip or read from the chip, depending on the $\overline{WR}$ or $\overline{RD}$ input signal.
CE or $\overline{CE}$	I	**Chip Enable:** On the 8155H, this pin is $\overline{CE}$ and is ACTIVE LOW. On the 8156H, this pin is CE and is ACTIVE HIGH.
$\overline{RD}$	I	**Read Control:** Input low on this line with the Chip Enable active enables and AD_{0-7} buffers. If $IO/\overline{M}$ pin is low, the RAM content will be read out to the AD bus. Otherwise the content of the selected I/O port or command/status registers will be read to the AD bus.
$\overline{WR}$	I	**Write Control:** Input low on this line with the Chip Enable active causes the data on the Address/Data bus to be written to the RAM or I/O ports and command/status register, depending on $IO/\overline{M}$.
ALE	I	**Address Latch Enable:** This control signal latches both the address on the AD_{0-7} lines and the state of the Chip Enable and $IO/\overline{M}$ into the chip at the falling edge of ALE.
$IO/\overline{M}$	I	**I/O Memory:** Selects memory if low and I/O and command/status registers if high.
PA_{0-7}(8)	I/O	**Port A:** These 8 pins are general purpose I/O pins. The in/out direction is selected by programming the command register.
PB_{0-7}(8)	I/O	**Port B:** These 8 pins are general purpose I/O pins. The in/out direction is selected by programming the command register.
PC_{0-5}(6)	I/O	**Port C:** These 6 pins can function as either input port, output port, or as control signals for PA and PB. Programming is done through the command register. When PC_{0-5} are used as control signals, they will provide the following: PC_0 — A INTR (Port A Interrupt) PC_1 — ABF (Port A Buffer Full) PC_2 — $\overline{A\ STB}$ (Port A Strobe) PC_3 — B INTR (Port B Interrupt) PC_4 — B BF (Port B Buffer Full) PC_5 — $\overline{B\ STB}$ (Port B Strobe)
TIMER IN	I	**Timer Input:** Input to the counter-timer.
$\overline{TIMER\ OUT}$	O	**Timer Output:** This output can be either a square wave or a pulse, depending on the timer mode.
V_{CC}		**Voltage:** +5 volt supply.
V_{SS}		**Ground:** Ground reference.

FUNCTIONAL DESCRIPTION

The 8155H/8156H contains the following:

- 2k Bit Static RAM organized as 256 x 8
- Two 8-bit I/O ports (PA & PB) and one 6-bit I/O port (PC)
- 14-bit timer-counter

The $IO/\overline{M}$ (IO/Memory Select) pin selects either the five registers (Command, Status, PA_{0-7}, PB_{0-7}, PC_{0-5}) or the memory (RAM) portion.

The 8-bit address on the Address/Data lines, Chip Enable input CE or $\overline{CE}$, and $IO/\overline{M}$ are all latched on-chip at the falling edge of ALE.

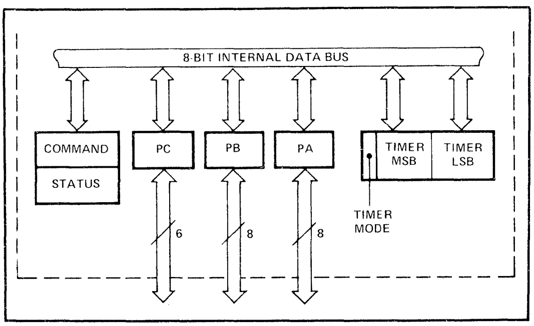


Figure 3. 8155H/8156H Internal Registers

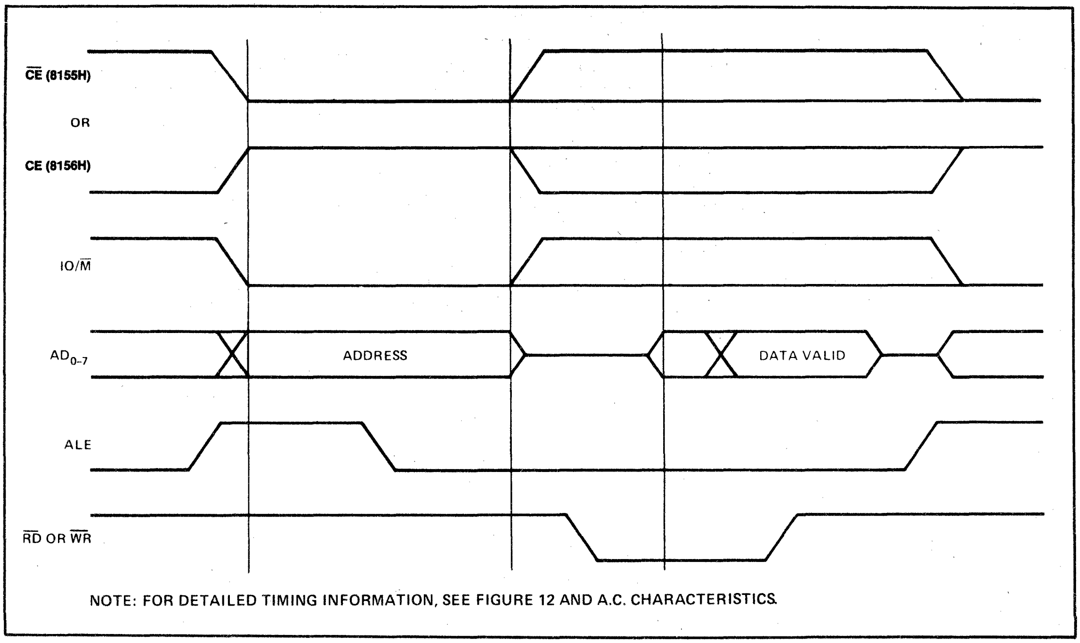


Figure 4. 8155H/8156H On-Board Memory Read/Write Cycle

PROGRAMMING OF THE COMMAND REGISTER

The command register consists of eight latches. Four bits (0-3) define the mode of the ports, two bits (4-5) enable or disable the interrupt from port C when it acts as control port, and the last two bits (6-7) are for the timer.

The command register contents can be altered at any time by using the I/O address XXXXX000 during a WRITE operation with the Chip Enable active and IO/$\overline{M}$ = 1. The meaning of each bit of the command byte is defined in Figure 5. The contents of the command register may never be read.

READING THE STATUS REGISTER

The status register consists of seven latches, one for each bit; six (0-5) for the status of the ports and one (6) for the status of the timer.

The status of the timer and the I/O section can be polled by reading the Status Register (Address XXXXX000). Status word format is shown in Figure 6. Note that you may never write to the status register since the command register shares the same I/O address and the command register is selected when a write to that address is issued.

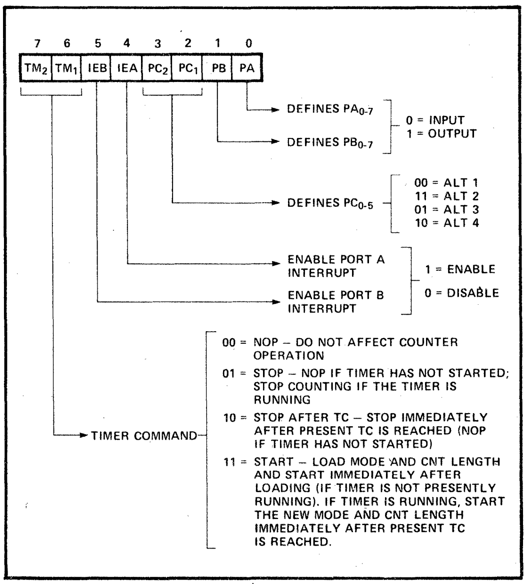


Figure 5. Command Register Bit Assignment

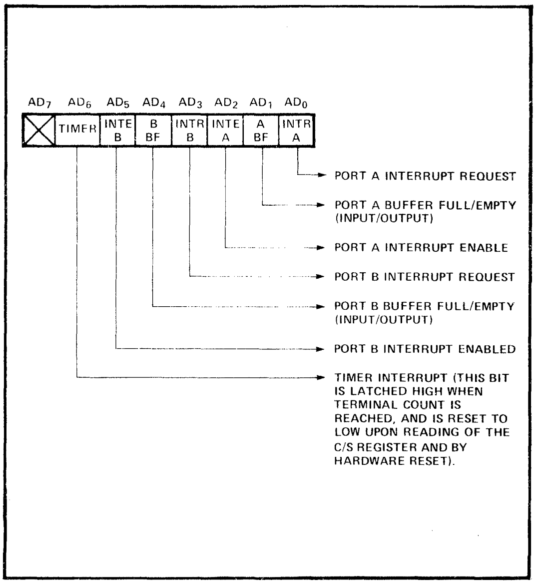


Figure 6. Status Register Bit Assignment

INPUT/OUTPUT SECTION

The I/O section of the 8155H/8156H consists of five registers: (See Figure 7.)

- **Command/Status Register (C/S)** — Both registers are assigned the address XXXXX000. The C/S address serves the dual purpose.

  When the C/S registers are selected during WRITE operation, a command is written into the command register. The contents of this register are *not* accessible through the pins.

  When the C/S (XXXXX000) is selected during a READ operation, the status information of the I/O ports and the timer becomes available on the AD_{0-7} lines.

- **PA Register** — This register can be programmed to be either input or output ports depending on the status of the contents of the C/S Register. Also depending on the command, this port can operate in either the basic mode or the strobed mode (See timing diagram). The I/O pins assigned in relation to this register are PA_{0-7}. The address of this register is XXXXX001.

- **PB Register** — This register functions the same as PA Register. The I/O pins assigned are PB_{0-7}. The address of this register is XXXXX010.

- **PC Register** — This register has the address XXXXX011 and contains only 6 bits. The 6 bits can be programmed to be either input ports, output ports or as control signals for PA and PB by properly programming the AD_2 and AD_3 bits of the C/S register.

  When PC_{0-5} is used as a control port, 3 bits are assigned for Port A and 3 for Port B. The first bit is an interrupt that the 8155H sends out. The second is an output signal indicating whether the buffer is full or empty, and the third is an input pin to accept a strobe for the strobed input mode. (See Table 2.)

When the 'C' port is programmed to either ALT3 or ALT4, the control signals for PA and PB are initialized as follows:

CONTROL	INPUT MODE	OUTPUT MODE
BF	Low	Low
INTR	Low	High
STB	Input Control	Input Control

I/O ADDRESS†								SELECTION
A7	A6	A5	A4	A3	A2	A1	A0	
X	X	X	X	X	0	0	0	Interval Command/Status Register
X	X	X	X	X	0	0	1	General Purpose I/O Port A
X	X	X	X	X	0	1	0	General Purpose I/O Port B
X	X	X	X	X	0	1	1	Port C — General Purpose I/O or Control
X	X	X	X	X	1	0	0	Low-Order 8 bits of Timer Count
X	X	X	X	X	1	0	1	High 6 bits of Timer Count and 2 bits of Timer Mode

X: Don't Care.

†: I/O Address must be qualified by CE = 1 (8156H) or $\overline{CE}$ = 0 (8155H) and IO/$\overline{M}$ = 1 in order to select the appropriate register.

Figure 7. I/O Port and Timer Addressing Scheme

Figure 8 shows how I/O PORTS A and B are structured within the 8155H and 8156H:

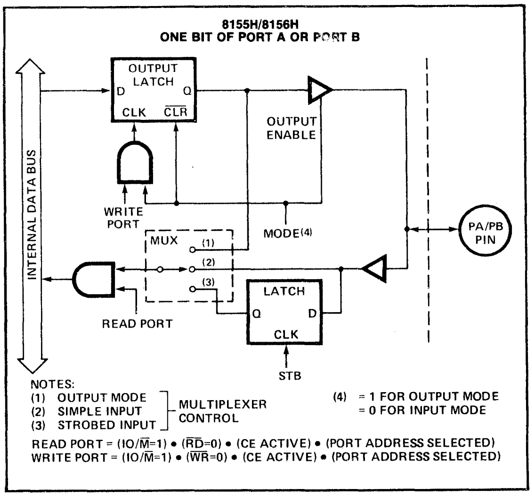


Figure 8. 8155H/8156H Port Functions

Table 2. Port Control Assignment

Pin	ALT 1	ALT 2	ALT 3	ALT 4
PC0	Input Port	Output Port	A INTR (Port A Interrupt)	A INTR (Port A Interrupt)
PC1	Input Port	Output Port	A BF (Port A Buffer Full)	A BF (Port A Buffer Full)
PC2	Input Port	Output Port	A $\overline{STB}$ (Port A Strobe)	A $\overline{STB}$ (Port A Strobe)
PC3	Input Port	Output Port	Output Port	B INTR (Port B Interrupt)
PC4	Input Port	Output Port	Output Port	B BF (Port B Buffer Full)
PC5	Input Port	Output Port	Output Port	B $\overline{STB}$ (Port B Strobe)

Note in the diagram that when the I/O ports are programmed to be output ports, the contents of the output ports can still be read by a READ operation when appropriately addressed.

The outputs of the 8155H/8156H are "glitch-free" meaning that you can write a "1" to a bit position that was previsouly "1" and the level at the output pin will not change.

Note also that the output latch is cleared when the port enters the input mode. The output latch cannot be loaded by writing to the port if the port is in the input mode. The result is that each time a port mode is changed from input to output, the output pins witll go low. When the 8155H/56H is RESET, the output latches are all cleared and all 3 ports enter the input mode.

When inthe ALT 1 or ALT 2 modes, the bits of PORT C are structured like the diagram above in the simple input or output mode, respectively.

Reading from an input port with nothing connected to the pins will provide unpredictable results.

Figure 9 shows how the 8155H/8156H I/O ports might be configured in a typical MCS-85 system.

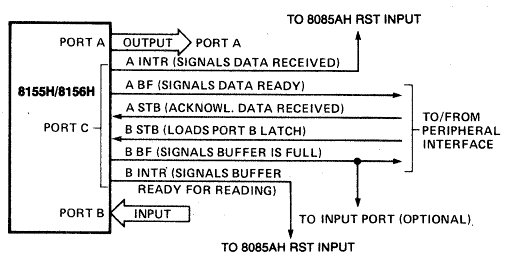


Figure 9. Example: Command Register = 00111001

TIMER SECTION

The time is a 14-bit down-counter that counts the TIMER IN pulses and provides either a square wave or pulse when terminal count (TC) is reached.

The timer has the I/O address XXXXX100 for the low order byte of the register and the I/O address XXXXX101 for the high order byte of the register. (See Figure 7).

To program the timer, the COUNT LENGTH REG is loaded first, one byte at a time, by selecting the timer addresses. Bits 0-13 of the high order count register will specify the length of the next count and bits 14-15 of the high order register will specify the timer output mode (see Figure 10). The value loaded into the count length register can have any value from 2H through 3FFFH in Bits 0-13.

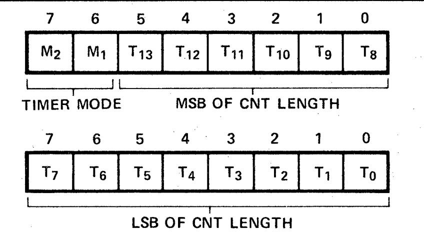


Figure 10. Timer Format

There are four modes to choose from: M2 and M1 define the timer mode, as shown in Figure 11.

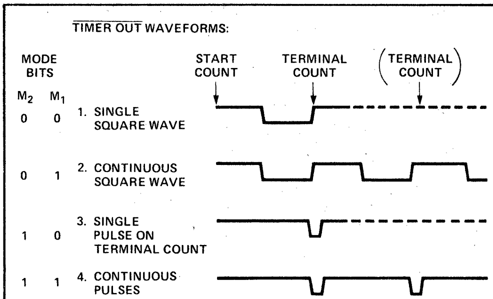


Figure 11. Timer Modes

Bits 6-7 (TM_2 and TM_1) of command register contents are used to start and stop the counter. There are four commands to choose from:

TM_2	TM_1	
0	0	NOP — Do not affect counter operation.
0	1	STOP — NOP if timer has not started; stop counting if the timer is running.
1	0	STOP AFTER TC — Stop immediately after present TC is reached (NOP if timer has not started)
1	1	START — Load mode and CNT length and start immediately after loading (if timer is not presently running). If timer is running, start the new mode and CNT length immediately after present TC is reached.

Note that while the counter is counting, you may load a new count and mode into the count length registers. Before the new count and mode will be used by the counter, you must issue a START command to the counter. This applies even though you may only want to change the count and use the previous mode.

In case of an odd-numbered count, the first half-cycle of the squarewave output, which is high, is one count longer than the second (low) half-cycle, as shown in Figure 12.

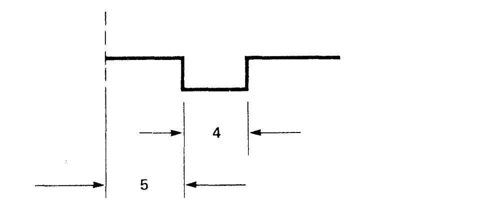


Figure 12. Asymmetrical Square-Wave Output Resulting from Count of 9

The counter in the 8155H is not initialized to any particular mode or count when hardware RESET occurs, but RESET does *stop* the counting. Therefore, counting cannot begin following RESET until a START command is issued via the C/S register.

Please note that the timer circuit on the 8155H/8156H chip is designed to be a square-wave timer, not an event counter. To achieve this, it counts down by twos twice in completing one cycle. Thus, its registers do not contain values directly representing the number of TIMER IN pulses received. You cannot load an initial value of 1 into the count register and cause the timer to operate, as its terminal count value is 10 (binary) or 2 (decimal). (For the detection of single pulses, it is suggested that one of the hardware interrupt pins on the 8085AH be used.) After the timer has started counting down, the values residing in the count registers can be used to calculate the actual number of TIMER IN pulses required to complete the timer cycle if desired. To obtain the remaining count, perform the following operations in order:

1. Stop the count
2. Read in the 16-bit value from the count length registers
3. Reset the upper two mode bits
4. Reset the carry and rotate right one position all 16 bits through carry
5. If carry is set, add 1/2 of the full original count (1/2 full count — 1 if full count is odd).

Note: If you started with an odd count and you read the count length register before the third count pulse occurs, you will not be able to discern whether one or two counts has occurred. Regardless of this, the 8155H/56H always counts out the right number of pulses in generating the $\overline{\text{TIMER OUT}}$ waveforms.

8085A MINIMUM SYSTEM CONFIGURATION

Figure 13a shows a minimum system using three chips, containing:

- 256 Bytes RAM
- 2K Bytes EPROM
- 38 I/O Pins
- 1 Interval Timer
- 4 Interrupt Levels

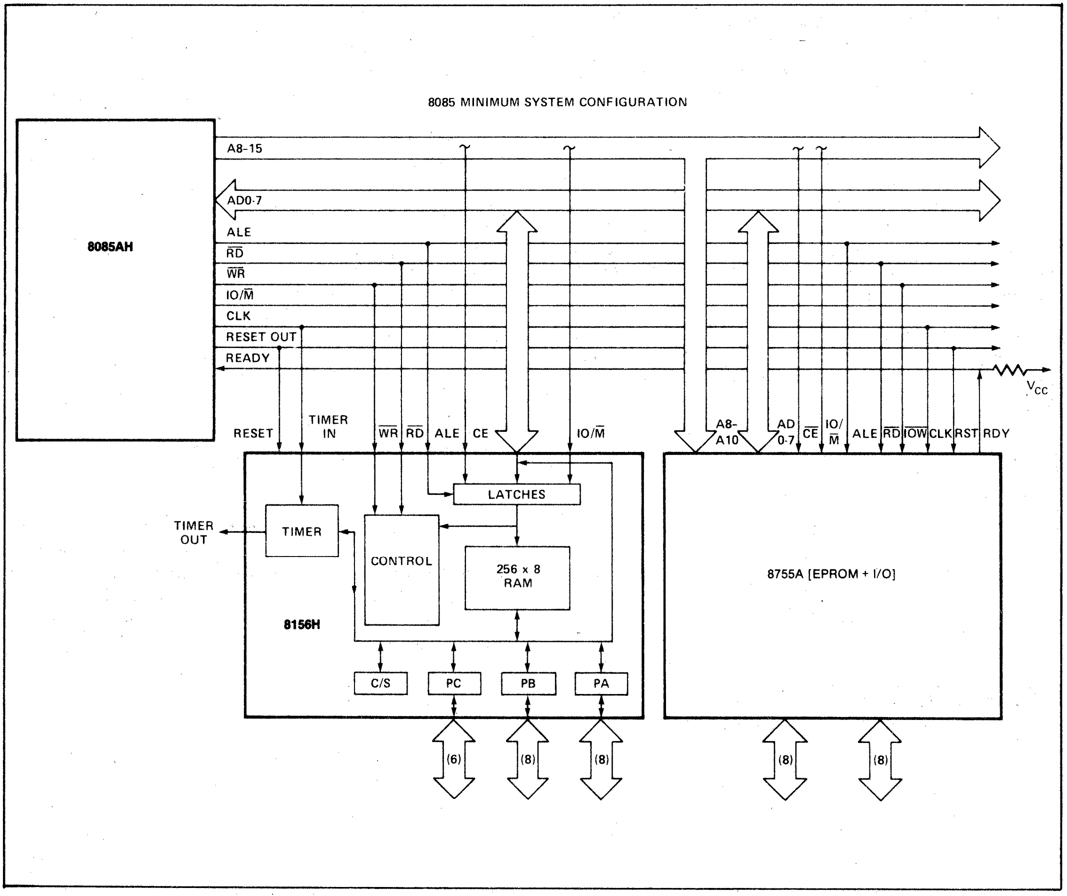


Figure 13a. 8085AH Minimum System Configuration (Memory Mapped I/O)

8088 FIVE CHIP SYSTEM

Figure 13b shows a five chip system containing:

- 1.25K Bytes RAM
- 2K Bytes EPROM
- 38 I/O Pins
- 1 Interval Timer
- 2 Interrupt Levels

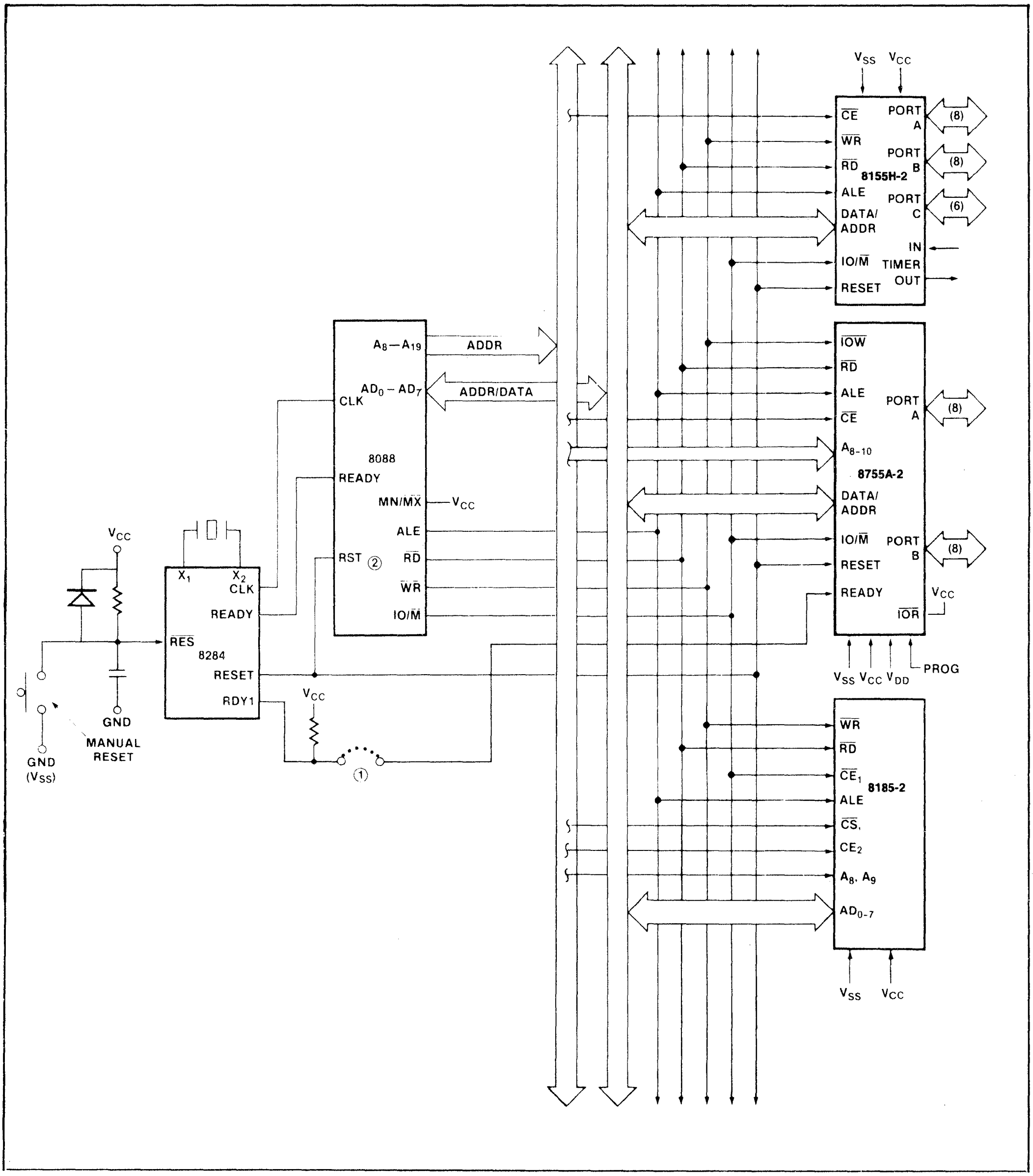


Figure 13b. 8088 Five Chip System Configuration

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... 0°C to +70°C
Storage Temperature .... -65°C to +150°C
Voltage on Any Pin
  With Respect to Ground .... -0.5V to +7V
Power Dissipation 1.5W

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ± 10%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	-0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -400μA
I_{IL}	Input Leakage		±10	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current		± 10	μA	$0.45V \leq V_{OUT} \leq V_{CC}$
I_{CC}	V_{CC} Supply Current		125	mA	
I_{IL} (CE)	Chip Enable Leakage 8155H 8156H		 +100 -100	 μA μA	$0V \leq V_{IN} \leq V_{CC}$

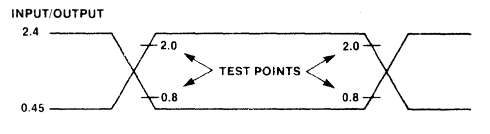
A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±10%)

		8155H/8156H		8155H-2/8156H-2		
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**
t_{AL}	Address to Latch Set Up Time	50		30		ns
t_{LA}	Address Hold Time after Latch	80		30		ns
t_{LC}	Latch to READ/WRITE Control	100		40		ns
t_{RD}	Valid Data Out Delay from READ Control		170		140	ns
t_{LD}	Latch to Data Out Valid		350		270	ns
t_{AD}	Address Stable to Data Out Valid		400		330	ns
t_{LL}	Latch Enable Width	100		70		ns
t_{RDF}	Data Bus Float After READ	0	100	0	80	ns
t_{CL}	READ/WRITE Control to Latch Enable	20		10		ns
t_{CC}	READ/WRITE Control Width	250		200		ns
t_{DW}	Data In to WRITE Set Up Time	150		100		ns
t_{WD}	Data In Hold Time After WRITE	25		25		ns
t_{RV}	Recovery Time Between Controls	300		200		ns
t_{WP}	WRITE to Port Output		400		300	ns
t_{PR}	Port Input Setup Time	70		50		ns
t_{RP}	Port Input Hold Time	50		10		ns
t_{SBF}	Strobe to Buffer Full		400		300	ns
t_{SS}	Strobe Width	200		150		ns
t_{RBE}	READ to Buffer Empty		400		300	ns
t_{SI}	Strobe to INTR On		400		300	ns

A.C. CHARACTERISTICS (Continued) (T_A = 0°C to 70°C, V_{CC} = 5V ±10%)

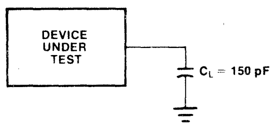
		8155H/8156H		8155H-2/8156H-2		
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**
t_{RDI}	READ to INTR Off		400		300	ns
t_{PSS}	Port Setup Time to Strobe Strobe	50		0		ns
t_{PHS}	Port Hold Time After Strobe	120		100		ns
t_{SBE}	Strobe to Buffer Empty		400		300	ns
t_{WBF}	WRITE to Buffer Full		400		300	ns
t_{WI}	WRITE to INTR Off		400		300	ns
t_{TL}	TIMER-IN to $\overline{\text{TIMER-OUT}}$ Low		400		300	ns
t_{TH}	TIMER-IN to $\overline{\text{TIMER-OUT}}$ High		400		300	ns
t_{RDE}	Data Bus Enable from READ Control	10		10		ns
t_1	TIMER-IN Low Time	80		40		ns
t_2	TIMER-IN High Time	120		70		ns
t_{WT}	WRITE to TIMER-IN (for writes which start counting)	360		200		ns

A.C. TESTING INPUT, OUTPUT WAVEFORM



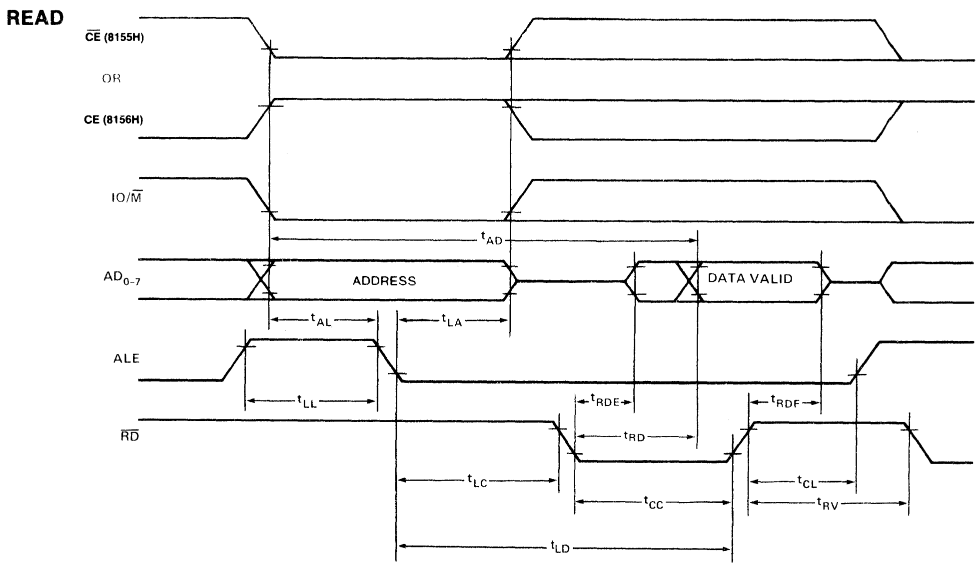
A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0.8V FOR A LOGIC "0."

A.C. TESTING LOAD CIRCUIT



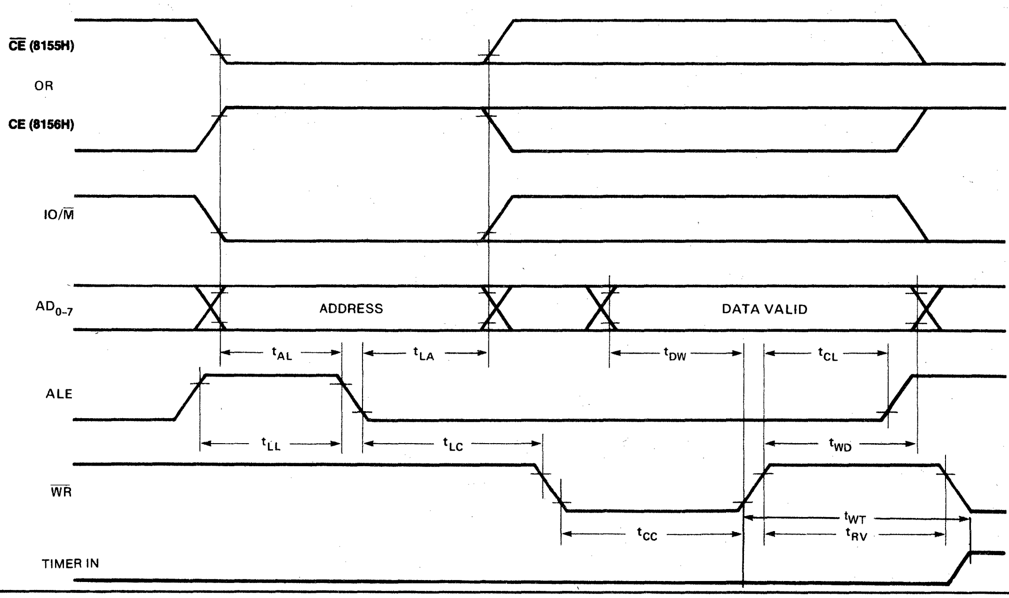
C_L = 150 pF
C_L INCLUDES JIG CAPACITANCE

WAVEFORMS

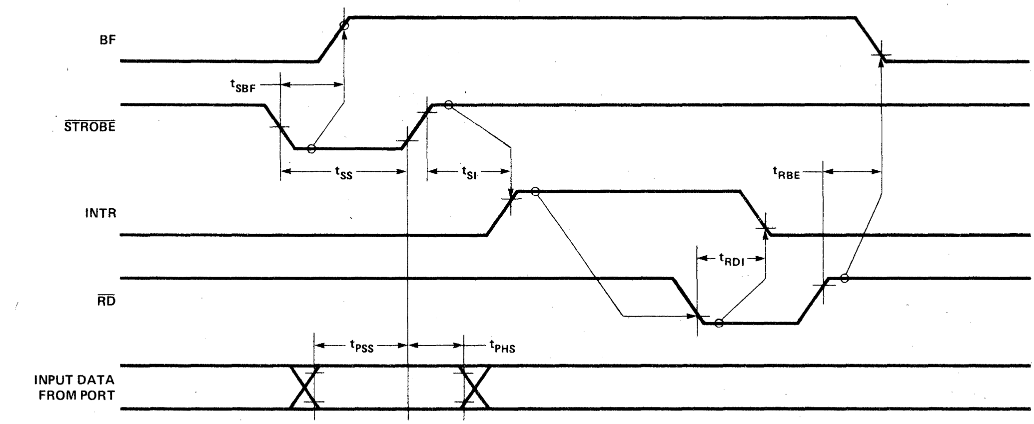


WAVEFORMS (Continued)

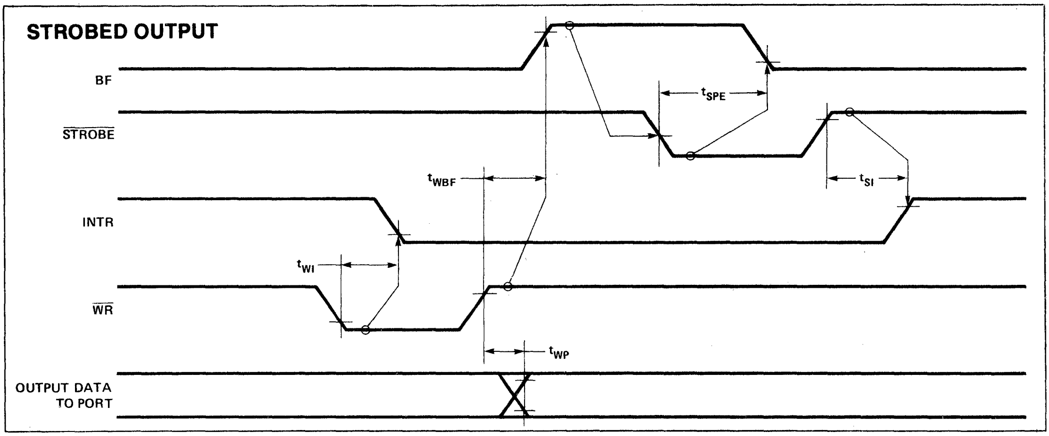
WRITE

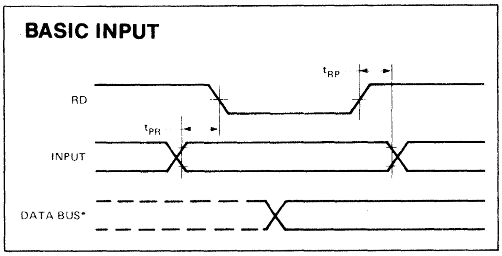


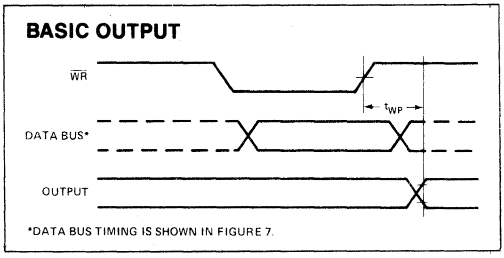
STROBED INPUT

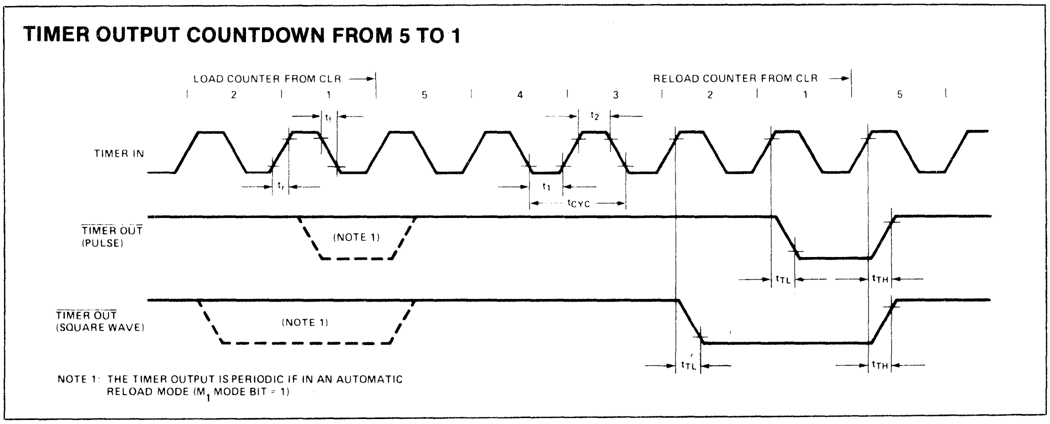


WAVEFORMS (Continued)









intel®

8185/8185-2
1024 x 8-BIT STATIC RAM FOR MCS-85®

- **Multiplexed Address and Data Bus**
- **Directly Compatible with 8085AH and iAPX 88 Microprocessors**
- **Low Operating Power Dissipation**
- **Low Standby Power Dissipation**
- **Single +5V Supply**
- **High Density 18-Pin Package**

The Intel® 8185 is an 8192-bit static random access memory (RAM) organized as 1024 words by 8-bits using N-channel Silicon-Gate MOS technology. The multiplexed address and data bus allows the 8185 to interface directly to the 8085A and iAPX 88 microprocessors to provide a maximum level of system integration.

The low standby power dissipation minimizes system power requirements when the 8185 is disabled.

The 8185-2 is a high-speed selected version of the 8185 that is compatible with the 5 MHz 8085AH-2 and the 5 MHz iAPX 88.

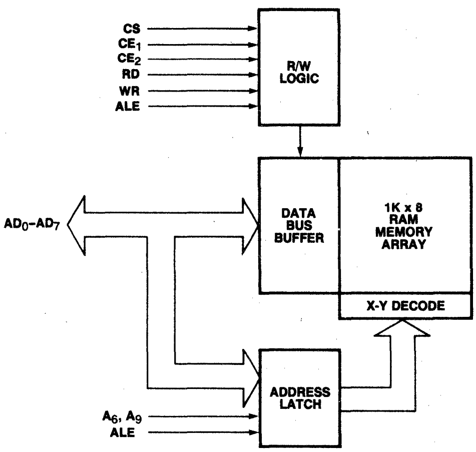
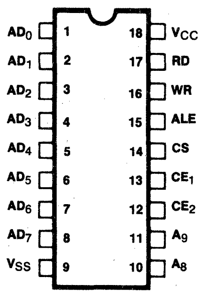


Figure 1. Block Diagram



AD_0-AD_7	ADDRESS/DATA LINES
A_8, A_9	ADDRESS LINES
CS	CHIP SELECT
CE_1	CHIP ENABLE (IO/M)
CE_2	CHIP ENABLE
ALE	ADDRESS LATCH ENABLE
WR	WRITE ENABLE

Figure 2. Pin Configuration

FUNCTIONAL DESCRIPTION

The 8185 has been designed to provide for direct interface to the multiplexed bus structure and bus timing of the 8085A microprocessor.

At the beginning of an 8185 memory access cycle, the 8-bit address on AD_{0-7}, A_8 and A_9, and the status of $\overline{CE}_1$ and CE_2 are all latched internally in the 8185 by the falling edge of ALE. If the latched status of both $\overline{CE}_1$ and CE_2 are active, the 8185 powers itself up, but no action occurs until the $\overline{CS}$ line goes low and the appropriate $\overline{RD}$ or $\overline{WR}$ control signal input is activated.

The $\overline{CS}$ input is not latched by the 8185 in order to allow the maximum amount of time for address decoding in selecting the 8185 chip. Maximum power consumption savings will occur, however, only when $\overline{CE}_1$ and CE_2 are activated selectively to power down the 8185 when it is not in use. A possible connection would be to wire the 8085A's IO/$\overline{M}$ line to the 8185's $\overline{CE}_1$ input, thereby keeping the 8185 powered down during I/O and interrupt cycles.

Table 1. Truth Table for Power Down and Function Enable

$\overline{CE}_1$	CE_2	$\overline{CS}$	(CS*)[2]	8185 Status
1	X	X	0	Power Down and Function Disable[1]
X	0	X	0	Power Down and Function Disable[1]
0	1	1	0	Powered Up and Function Disable[1]
0	1	0	1	Powered Up and Enabled

NOTES:
X: Don't Care.
1: Function Disable implies Data Bus in high impedance state and not writing.
2: CS* = ($\overline{CE}_1$ = 0) • (CE_2 = 1) • ($\overline{CS}$ = 0)
CS* = 1 signifies all chip enables and chip select active

Table 2. Truth Table for Control and Data Bus Pin Status

(CS*)	$\overline{RD}$	$\overline{WR}$	AD_{0-7} During Data Portion of Cycle	8185 Function
0	X	X	Hi-Impedance	No Function
1	0	1	Data from Memory	Read
1	1	0	Data to Memory	Write
1	1	1	Hi-Impedance	Reading, but not Driving Data Bus

NOTE:
X: Don't Care.

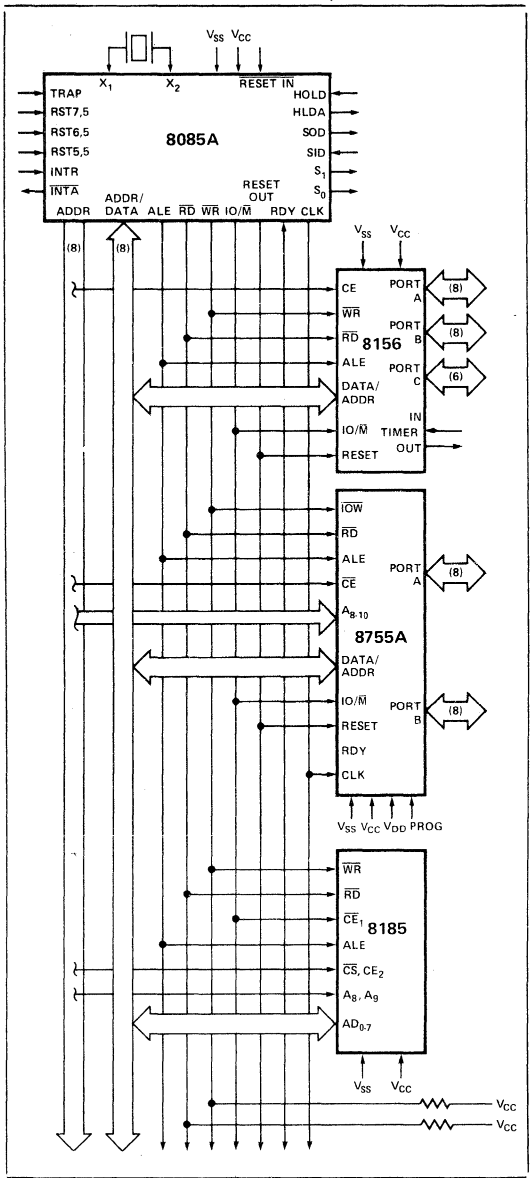


Figure 3. 8185 in an MCS-85 System

4 Chips:
- 2K Bytes EPROM
- 1.25K Bytes RAM
- 38 I/O Lines
- 1 Counter/Timer
- 2 Serial I/O Lines
- 5 Interrupt Inputs

iAPX 88 FIVE CHIP SYSTEM:

- 1.25 K Bytes RAM
- 2K Bytes EPROM
- 38 I/O Pins
- 1 Internal Timer
- 2 Interrupt Levels

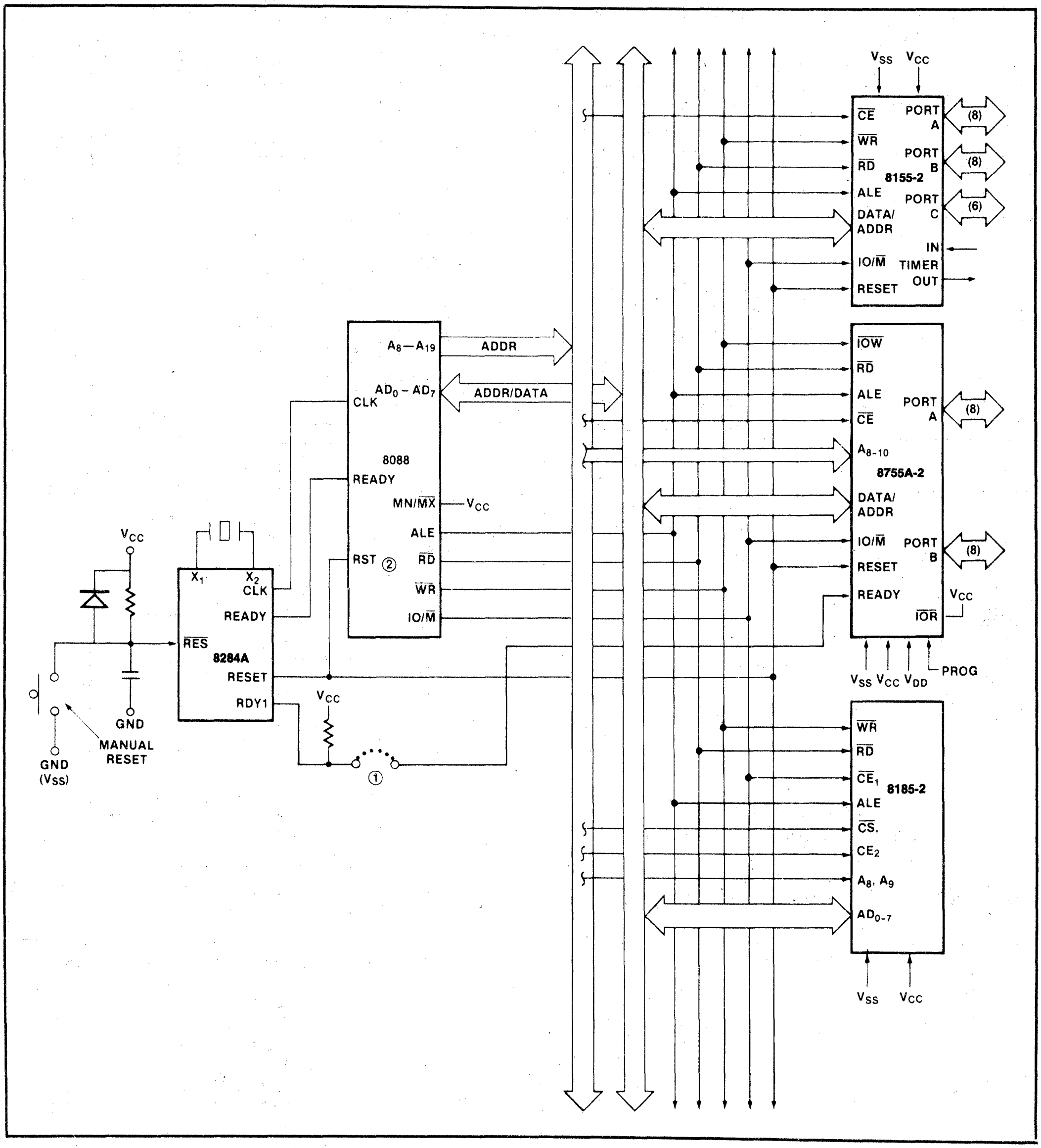


Figure 4. iAPX 88 Five Chip System Configuration

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... 0°C to +70°C
Storage Temperature .... -65°C to +150°C
Voltage on Any Pin
with Respect to Ground .... -0.5V to +7V
Power Dissipation 1.5W

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

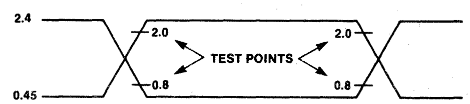
D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	-0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2mA
V_{OH}	Output High Voltage	2.4			I_{OH} = − 400μA
I_{IL}	Input Leakage		±10	μA	0V ≤V_{IN} ≤V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}
I_{CC}	V_{CC} Supply Current Powered Up		100	mA	
	Powered Down		35	mA	

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)

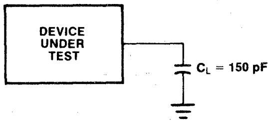
		8185		8185-2		
Symbol	Parameter	Min.	Max.	Min.	Max.	Units
t_{AL}	Address to Latch Set Up Time	50		30		ns
t_{LA}	Address Hold Time After Latch	80		30		ns
t_{LC}	Latch to READ/WRITE Control	100		40		ns
t_{RD}	Valid Data Out Delay from READ Control		170		140	ns
t_{LD}	ALE to Data Out Valid		300		200	ns
t_{LL}	Latch Enable Width	100		70		ns
t_{RDF}	Data Bus Float After READ	0	100	0	80	ns
t_{CL}	READ/WRITE Control to Latch Enable	20		10		ns
t_{CC}	READ/WRITE Control Width	250		200		ns
t_{DW}	Data In to WRITE Set Up Time	150		150		ns
t_{WD}	Data In Hold Time After WRITE	20		20		ns
t_{SC}	Chip Select Set Up to Control Line	10		10		ns
t_{CS}	Chip Select Hold Time After Control	10		10		ns
t_{ALCE}	Chip Enable Set Up to ALE Falling	30		10		ns
t_{LACE}	Chip Enable Hold Time After ALE	50		30		ns

A.C. TESTING INPUT, OUTPUT WAVEFORM



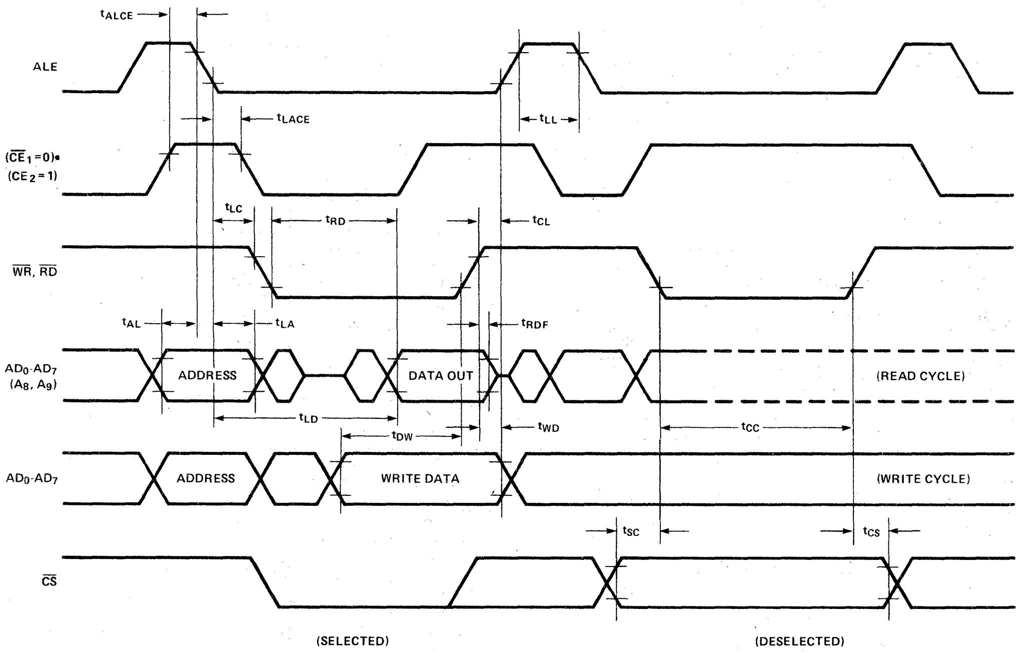
A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0.8V FOR A LOGIC "0."

A.C. TESTING LOAD CIRCUIT



C_L = 150 pF
C_L INCLUDES JIG CAPACITANCE

WAVEFORM



intel®

8224
CLOCK GENERATOR AND DRIVER FOR 8080A CPU

- **Single Chip Clock Generator/Driver for 8080A CPU**
- **Power-Up Reset for CPU**
- **Ready Synchronizing Flip-Flop**
- **Advanced Status Strobe**
- **Oscillator Output for External System Timing**
- **Crystal Controlled for Stable System Operation**
- **Reduces System Package Count**
- **Available in EXPRESS**
  - Standard Temperature Range
- **Available in 16-Lead Cerdip Package**
  (See Packaging Spec, Order #231369)

The Intel® 8224 is a single chip clock generator/driver for the 8080A CPU. It is controlled by a crystal, selected by the designer to meet a variety of system speed requirements.

Also included are circuits to provide power-up reset, advance status strobe, and synchronization of ready.

The 8224 provides the designer with a significant reduction of packages used to generate clocks and timing for 8080A.

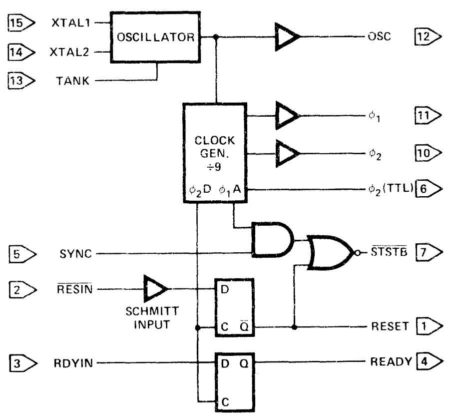
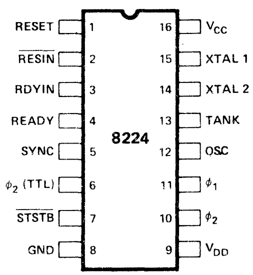


Figure 1. Block Diagram



$\overline{RESIN}$	RESET INPUT
RESET	RESET OUTPUT
RDYIN	READY INPUT
READY	READY OUTPUT
SYNC	SYNC INPUT
$\overline{STSTB}$	STATUS STB (ACTIVE LOW)
ϕ_1	} 8080
ϕ_2	} CLOCKS

XTAL 1	} CONNECTIONS
XTAL 2	} FOR CRYSTAL
TANK	USED WITH OVERTONE XTAL
OSC	OSCILLATOR OUTPUT
ϕ_2 (TTL)	ϕ_2 CLK (TTL LEVEL)
V_{CC}	+5V
V_{DD}	+12V
GND	0V

Figure 2. Pin Configuration

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias . . . . 0°C to 70°C
Storage Temperature . . . . −65°C to 150°C
Supply Voltage, V_{CC} . . . . −0.5V to +7V
Supply Voltage, V_{DD} . . . . −0.5V to +13.5V
Input Voltage . . . . −1.5V to +7V
Output Current 100mA

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5.0V ±5%, V_{DD} = +12V ±5%)

Symbol	Parameter	Limits Min.	Limits Typ.	Limits Max.	Units	Test Conditions
I_F	Input Current Loading			-.25	mA	V_F = .45V
I_R	Input Leakage Current			10	μA	V_R = 5.25V
V_C	Input Forward Clamp Voltage			1.0	V	I_C = -5mA
V_{IL}	Input "Low" Voltage			.8	V	V_{CC} = 5.0V
V_{IH}	Input "High" Voltage	2.6 2.0			V	Reset Input All Other Inputs
V_{IH}-V_{IL}	RESIN Input Hysteresis	.25			V	V_{CC} = 5.0V
V_{OL}	Output "Low" Voltage			.45 .45	V V	(ϕ_1,ϕ_2), Ready, Reset, $\overline{STSTB}$ I_{OL} =2.5mA All Other Outputs I_{OL} = 15mA
V_{OH}	Output "High" Voltage ϕ_1 , ϕ_2 READY, RESET All Other Outputs	 9.4 3.6 2.4			 V V V	 I_{OH} = -100μA I_{OH} = -100μA I_{OH} = -1mA
I_{CC}	Power Supply Current			115	mA	
I_{DD}	Power Supply Current			12	mA	

Note: 1. For crystal frequencies of 18 MHz connect 510 Ω registers between the X1 input and ground as well as the X2 input and ground to prevent oscillation at harmonic frequencies.

Crystal Requirements

Tolerance: 0.005% at 0°C-70°C
Resonance: Series (Fundamental)*
Load Capacitance: 20-35 pF
Equivalent Resistance: 75-20 ohms
Power Dissipation (Min): 4 mW

*With tank circuit use 3rd overtone mode.

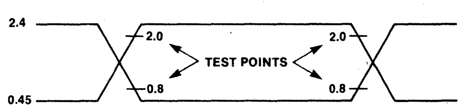
A.C. CHARACTERISTICS (V_{CC} = +5.0V ±5%, V_{DD} = +12.0V ±5%, T_A = 0°C to 70°C)

Symbol	Parameter	Limits Min.	Limits Typ.	Limits Max.	Units	Test Conditions
$t_{\phi 1}$	ϕ_1 Pulse Width	$\frac{2tcy}{9}$ - 20ns			ns	C_L = 20pF to 50pF
$t_{\phi 2}$	ϕ_2 Pulse Width	$\frac{5tcy}{9}$ - 35ns				
t_{D1}	ϕ_1 to ϕ_2 Delay	0				
t_{D2}	ϕ_2 to ϕ_1 Delay	$\frac{2tcy}{9}$ - 14ns				
t_{D3}	ϕ_1 to ϕ_2 Delay	$\frac{2tcy}{9}$		$\frac{2tcy}{9}$ + 20ns		
t_R	ϕ_1 and ϕ_2 Rise Time			20		
t_F	ϕ_1 and ϕ_2 Fall Time			20		
$t_{D\phi 2}$	ϕ_2 to ϕ_2 (TTL) Delay	-5		+15	ns	ϕ_2TTL,CL=30 R_1=300Ω R_2=600Ω
t_{DSS}	ϕ_2 to $\overline{STSTB}$ Delay	$\frac{6tcy}{9}$ - 30ns		$\frac{6tcy}{9}$		
t_{PW}	$\overline{STSTB}$ Pulse Width	$\frac{tcy}{9}$ - 15ns				$\overline{STSTB}$, CL=15pF R_1 = 2K R_2 = 4K
t_{DRS}	RDYIN Setup Time to Status Strobe	50ns - $\frac{4tcy}{9}$				
t_{DRH}	RDYIN Hold Time After $\overline{STSTB}$	$\frac{4tcy}{9}$				
t_{DR}	RDYIN or RESIN to ϕ_2 Delay	$\frac{4tcy}{9}$ - 25ns				Ready & Reset CL=10pF R_1=2K R_2=4K
t_{CLK}	CLK Period		$\frac{tcy}{9}$			
f_{max}	Maximum Oscillating Frequency			27	MHz	
C_{in}	Input Capacitance			8	pF	V_{CC}=+5.0V V_{DD}=+12V V_{BIAS}=2.5V f=1MHz

A.C. CHARACTERISTICS (Continued) (For t_{CY} = 488.28 ns) (T_A = 0°C to 70°C, V_{DD} = +5V ±5%, V_{DD} = +12V ±5%)

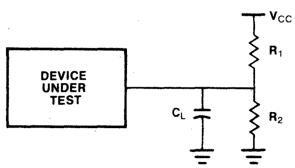
Symbol	Parameter	Limits Min.	Typ.	Max.	Units	Test Conditions
$t_{\phi 1}$	ϕ_1 Pulse Width	89			ns	t_{CY}=488.28ns
$t_{\phi 2}$	ϕ_2 Pulse Width	236			ns	
t_{D1}	Delay ϕ_1 to ϕ_2	0			ns	
t_{D2}	Delay ϕ_2 to ϕ_1	95			ns	ϕ_1 & ϕ_2 Loaded to C_L = 20 to 50pF
t_{D3}	Delay ϕ_1 to ϕ_2 Leading Edges	109		129	ns	
t_r	Output Rise Time			20	ns	
t_f	Output Fall Time			20	ns	
t_{DSS}	ϕ_2 to $\overline{STSTB}$ Delay	296		326	ns	
$t_{D\phi 2}$	ϕ_2 to ϕ_2 (TTL) Delay	-5		+15	ns	
t_{PW}	Status Strobe Pulse Width	40			ns	Ready & Reset Loaded to 2mA/10pF
t_{DRS}	RDYIN Setup Time to $\overline{STSTB}$	-167			ns	
t_{DRH}	RDYIN Hold Time after $\overline{STSTB}$	217			ns	All measurements referenced to 1.5V unless specified otherwise.
t_{DR}	READY or RESET to ϕ_2 Delay	192			ns	
f_{MAX}	Oscillator Frequency			18.432	MHz	

A.C. TESTING INPUT, OUTPUT WAVEFORM



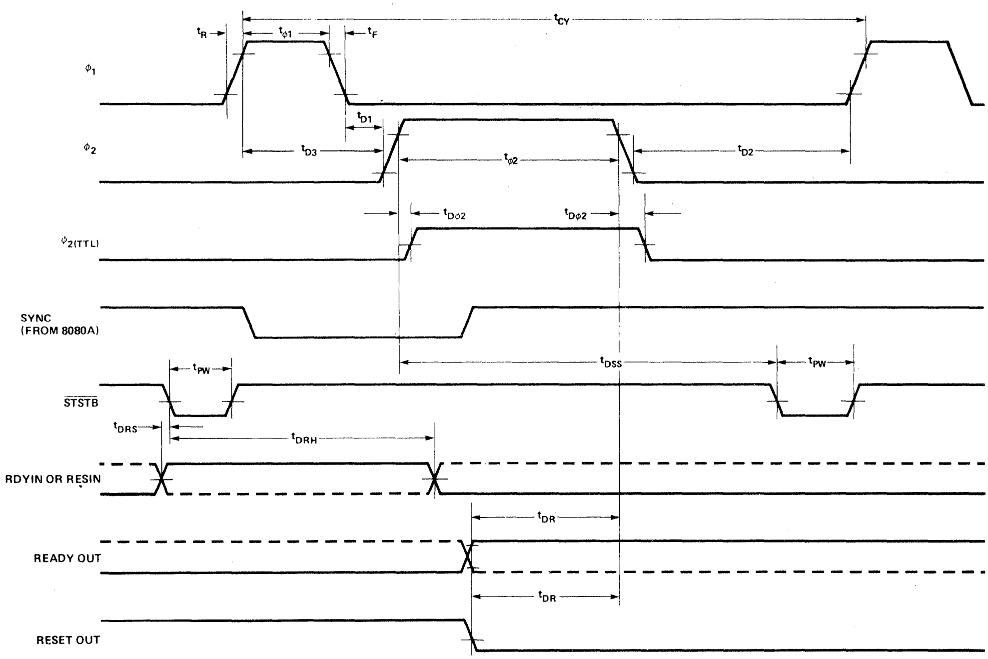
A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0.8V FOR A LOGIC "0" (UNLESS OTHERWISE NOTED).

A.C. TESTING LOAD CIRCUIT



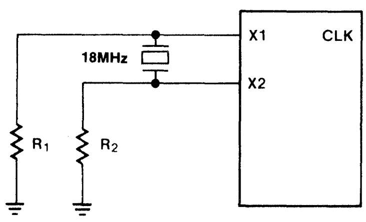
C_L INCLUDES JIG CAPACITANCE

WAVEFORMS



VOLTAGE MEASUREMENT POINTS: ϕ_1, ϕ_2 Logic "0" = 1.0V, Logic "1" = 8.0V. All other signals measured at 1.5V.

CLOCK HIGH AND LOW TIME (USING X1, X2)



![intel logo]

8228/8238
SYSTEM CONTROLLER AND BUS DRIVER FOR 8080A CPU

- **Single Chip System Control for MCS-80® Systems**
- **Built-In Bidirectional Bus Driver for Data Bus Isolation**
- **Allows the Use of Multiple Byte Instructions (e.g. CALL) for Interrupt Acknowledge**
- **Reduces System Package Count**
- **User Selected Single Level Interrupt Vector (RST 7)**
- **8283 Has Advanced IOW/MEMW for Large System Timing Control**
- **Available in EXPRESS**
  - **— Standard Temperature Range**
- **Available in 28-Lead Cerdip and Plastic Packages**

(See Packaging Spec, Order #231369)

The Intel® 8228 is a single chip system controller and bus driver for MCS-80. It generates all signals required to directly interface MCS-80 family RAM, ROM, and I/O components.

A bidirectional bus driver is included to provide high system TTL fan-out. It also provides isolation of the 8080 data bus from memory and I/O. This allows for the optimization of control signals, enabling the systems designer to use slower memory and I/O. The isolation of the bus driver also provides for enhanced system noise immunity.

A user selected single level interrupt vector (RST 7) is provided to simplify real time, interrupt driven, small system requirements. The 8228 also generates the correct control signals to allow the use of multiple byte instructions (e.g., CALL) in response to an interrupt acknowledge by the 8080A. This feature permits large, interrupt driven systems to have an unlimited number of interrupt levels.

The 8228 is designed to support a wide variety of system bus structures and also reduce system package count for cost effective, reliable design of the MCS-80 systems.

Note: The specifications for the 3228/3238 are identical with those for the 8228/8238

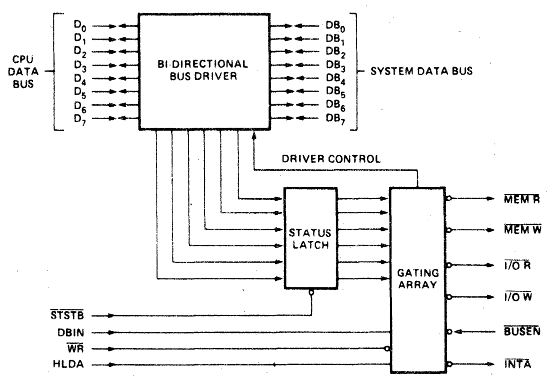
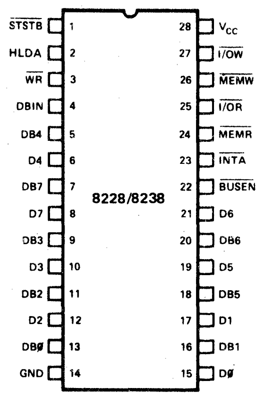


Figure 1. Block Diagram



D7-D0	DATA BUS (8080 SIDE)	INTA	INTERRUPT ACKNOWLEDGE
DB7-DB0	DATA BUS (SYSTEM SIDE)	HLDA	HLDA (FROM 8080)
I/OR	I/O READ	WR	WR (FROM 8080)
I/OW	I/O WRITE	BUSEN	BUS ENABLE INPUT
MEMR	MEMORY READ	STSTB	STATUS STROBE (FROM 8224)
MEMW	MEMORY WRITE	Vcc	+5V
DBIN	DBIN (FROM 8080)	GND	0 VOLTS

Figure 2. Pin Configuration

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... – 0°C to 70°C
Storage Temperature .... – 65°C to 150°C
Supply Voltage, V_{CC} .... – 0.5V to +7V
Input Voltage .... – 1.5V to +7V
Output Current .... 100 mA

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not limited. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±5%)

Symbol	Parameter	Limits Min.	Limits Typ.[1]	Limits Max.	Unit	Test Conditions
V_C	Input Clamp Voltage, All Inputs		.75	-1.0	V	V_{CC}=4.75V; I_C=-5mA
I_F	Input Load Current, $\overline{STSTB}$			500	µA	V_{CC} = 5.25V
	D_2 & D_6			750	µA	V_F = 0.45V
	D_0, D_1, D_4, D_5, & D_7			250	µA	
	All Other Inputs			250	µA	
I_R	Input Leakage Current $\overline{STSTB}$			100	µA	V_{CC} = 5.25V
	DB_0-DB_7			20	µA	V_R = 5.25V
	All Other Inputs			100	µA	
V_{TH}	Input Threshold Voltage, All Inputs	0.8		2.0	V	V_{CC} = 5V
I_{CC}	Power Supply Current		140	190	mA	V_{CC}=5.25V
V_{OL}	Output Low Voltage, D_0-D_7			.45	V	V_{CC}=4.75V; I_{OL}=2mA
	All Other Outputs			.45	V	I_{OL} = 10mA
V_{OH}	Output High Voltage, D_0-D_7	3.6	3.8		V	V_{CC}=4.75V; I_{OH}=-10µA
	All Other Outputs	2.4			V	I_{OH} = -1mA
I_{OS}	Short Circuit Current, All Outputs	15		90	mA	V_{CC}=5V
$I_{O(off)}$	Off State Output Current, All Control Outputs			100	µA	V_{CC}=5.25V; V_O=5.25
				-100	µA	V_O=.45V
I_{INT}	INTA Current			5	mA	(See INTA Test Circuit)

Note 1: Typical values are for T_A = 25°C and nominal supply voltages.

CAPACITANCE (V_{BIAS} = 2.5V, V_{CC} = 5.0V, T_A = 25°C, f = 1 MHz)

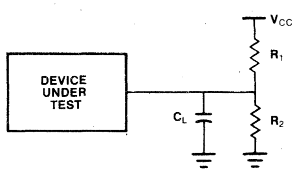
This parameter is periodically sampled and not 100% tested.

Symbol	Parameter	Limits Min.	Limits Typ.[1]	Limits Max.	Unit
C_{IN}	Input Capacitance		8	12	pF
C_{OUT}	Output Capacitance Control Signals		7	15	pF
I/O	I/O Capacitance (D or DB)		8	15	pF

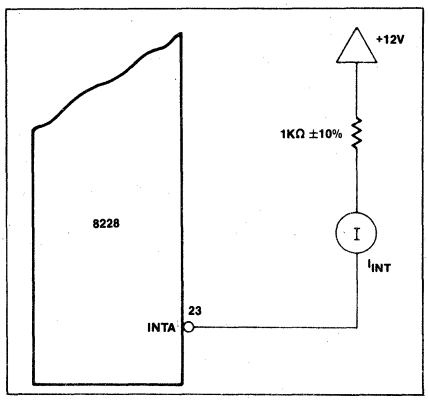
A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±5%)

Symbol	Parameter	Limits Min.	Limits Max.	Units	Condition
t_{PW}	Width of Status Strobe	22		ns	
t_{SS}	Setup Time, Status Inputs D_0-D_7	8		ns	
t_{SH}	Hold Time, Status Inputs D_0-D_7	5		ns	
t_{DC}	Delay from $\overline{STSTB}$ to any Control Signal	20	60	ns	C_L = 100pF
t_{RR}	Delay from DBIN to Control Outputs		30	ns	C_L = 100pF
t_{RE}	Delay from DBIN to Enable/Disable 8080 Bus		45	ns	C_L = 25pF
t_{RD}	Delay from System Bus to 8080 Bus during Read		30	ns	C_L = 25pF
t_{WR}	Delay from $\overline{WR}$ to Control Outputs	5	45	ns	C_L = 100pF
t_{WE}	Delay to Enable System Bus DB_0-DB_7 after $\overline{STSTB}$		30	ns	C_L = 100pF
t_{WD}	Delay from 8080 Bus D_0-D_7 to System Bus DB_0-DB_7 during Write	5	40	ns	C_L = 100pF
t_E	Delay from $\overline{\text{System Bus Enable}}$ to System Bus DB_0-DB_7		30	ns	C_L = 100pF
t_{HD}	HLDA to Read Status Outputs		25	ns	
t_{DS}	Setup Time, System Bus Inputs to HLDA	10		ns	
t_{DH}	Hold Time, System Bus Inputs to HLDA	20		ns	C_L = 100pF

A.C. TESTING LOAD CIRCUIT

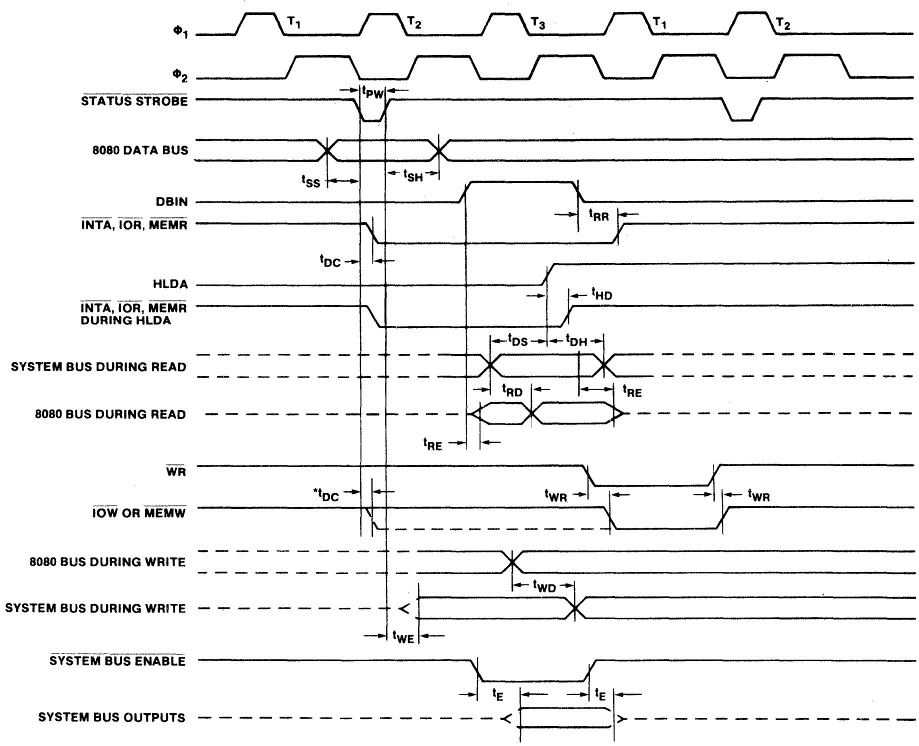


For D_0-D_7: R_1 = 4KΩ, R_2 = ∞Ω, C_L = 25pF. For all other outputs: R_1 = 500Ω, R_2 = 1KΩ, C_L = 100pF.



INTA Test Circuit (for RST 7)

WAVEFORM



VOLTAGE MEASUREMENT POINTS: D_0-D_7 (when outputs) Logic "0" = 0.8V, Logic "1" = 3.0V. All other signals measured at 1.5V.

*ADVANCED IOW/MEMW **FOR 8283 ONLY.**

8237A/8237A-4/8237A-5
HIGH PERFORMANCE PROGRAMMABLE DMA CONTROLLER

- Enable/Disable Control of Individual DMA Requests
- Four Independent DMA Channels
- Independent Autoinitialization of all Channels
- Memory-to-Memory Transfers
- Memory Block Initialization
- Address Increment or Decrement
- High performance: Transfers up to 1.6M Bytes/Second with 5 MHz 8237A-5
- Directly Expandable to any Number of Channels
- End of Process Input for Terminating Transfers
- Software DMA Requests
- Independent Polarity Control for DREQ and DACK Signals
- Available in EXPRESS
  - Standard Temperature Range
- Available in 40-Lead Cerdip and Plastic Packages
  (See Packaging Spec, Order #231369)

The 8237A Multimode Direct Memory Access (DMA) Controller is a peripheral interface circuit for microprocessor systems. It is designed to improve system performance by allowing external devices to directly transfer information from the system memory. Memory-to-memory transfer capability is also provided. The 8237A offers a wide variety of programmable control features to enhance data throughput and system optimization and to allow dynamic reconfiguration under program control.

The 8237A is designed to be used in conjunction with an external 8-bit address register such as the 8282. It contains four independent channels and may be expanded to any number of channels by cascading additional controller chips.

The three basic transfer modes allow programmability of the types of DMA service by the user. Each channel can be individually programmed to Autoinitialize to its original condition following an End of Process (EOP).

Each channel has a full 64K address and word count capability.

The 8237A-4 and 8237A-5 are 4 MHz and 5 MHz selected versions of the standard 3 MHz 8237A respectively.

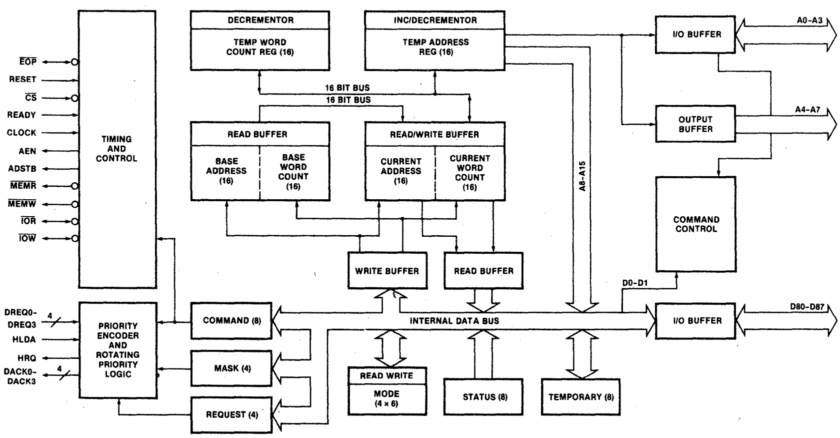


Figure 1. Block Diagram

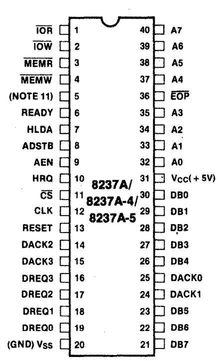


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Type	Name and Function
V_{CC}		**Power:** +5 volt supply.
V_{SS}		**Ground:** Ground.
CLK	I	**Clock Input:** Clock Input controls the internal operations of the 8237A and its rate of data transfers. The input may be driven at up to 3 MHz for the standard 8237A and up to 5 MHz for the 8237A-5.
$\overline{CS}$	I	**Chip Select:** Chip Select is an active low input used to select the 8237A as an I/O device during the Idle cycle. This allows CPU communication on the data bus.
RESET	I	**Reset:** Reset is an active high input which clears the Command, Status, Request and Temporary registers. It also clears the first/last flip/flop and sets the Mask register. Following a Reset the device is in the Idle cycle.
READY	I	**Ready:** Ready is an input used to extend the memory read and write pulses from the 8237A to accommodate slow memories or I/O peripheral devices. Ready must not make transitions during its specified setup/hold time.
HLDA	I	**Hold Acknowledge:** The active high Hold Acknowledge from the CPU indicates that it has relinquished control of the system busses.
DREQ0-DREQ3	I	**DMA Request:** The DMA Request lines are individual asynchronous channel request inputs used by peripheral circuits to obtain DMA service. In fixed Priority, DREQ0 has the highest priority and DREQ3 has the lowest priority. A request is generated by activating the DREQ line of a channel. DACK will acknowledge the recognition of DREQ signal. Polarity of DREQ is programmable. Reset intializes these lines to active high. DREQ must be maintained until the corresponding DACK goes active.
DB0-DB7	I/O	**Data Bus:** The Data Bus lines are bidirectional three-state signals connected to the system data bus. The outputs are enabled in the Program condition during the I/O Read to output the contents of an Address register, a Status register, the Temporary register or a Word Count register to the CPU. The outputs are disabled and the inputs are read during an I/O Write cycle when the CPU is programming the 8237A control registers. During DMA cycles the most significant 8 bits of the address are output onto the data bus to be strobed into an external latch by ADSTB. In memory-to-memory operations, data from the memory comes into the 8237A on the data bus during the read-from-memory transfer. In the write-to-memory transfer, the data bus outputs place the data into the new memory location.
$\overline{IOR}$	I/O	**I/O Read:** I/O Read is a bidirectional active low three-state line. In the Idle cycle, it is an input control signal used by the CPU to read the control registers. In the Active cycle, it is an output control signal used by the 8237A to access data from a peripheral during a DMA Write transfer.
$\overline{IOW}$	I/O	**I/O Write:** I/O Write is a bidirectional active low three-state line. In the Idle cycle, it is an input control signal used by the CPU to load information into the 8237A. In the Active cycle, it is an output control signal used by the 8237A to load data to the peripheral during a DMA Read transfer.
$\overline{EOP}$	I/O	**End of Process:** End of Process is an active low bidirectional signal. Information concerning the completion of DMA services is available at the bidirectional $\overline{EOP}$ pin. The 8237A allows an external signal to terminate an active DMA service. This is accomplished by pulling the $\overline{EOP}$ input low with an external $\overline{EOP}$ signal. The 8237A also generates a pulse when the terminal count (TC) for any channel is reached. This generates an $\overline{EOP}$ signal which is output through the $\overline{EOP}$ Line. The reception of $\overline{EOP}$, either internal or external, will cause the 8237A to terminate the service, reset the request, and, if Autoinitialize is enabled, to write the base registers to the current registers of that channel. The mask bit and TC bit in the status word will be set for the currently active channel by $\overline{EOP}$ unless the channel is programmed for Autoinitialize. In that case, the mask bit remains unchanged. During memory-to-memory transfers, $\overline{EOP}$ will be output when the TC for channel 1 occurs. $\overline{EOP}$ should be tied high with a pull-up resistor if it is not used to prevent erroneous end of process inputs.
A0-A3	I/O	**Address:** The four least significant address lines are bidirectional three-state signals. In the Idle cycle they are inputs and are used by the CPU to address the register to be loaded or read. In the Active cycle they are outputs and provide the lower 4 bits of the output address.

Table 1. Pin Description (Continued)

Symbol	Type	Name and Function
A4-A7	O	**Address:** The four most significant address lines are three-state outputs and provide 4 bits of address. These lines are enabled only during the DMA service.
HRQ	O	**Hold Request:** This is the Hold Request to the CPU and is used to request control of the system bus. If the corresponding mask bit is clear, the presence of any valid DREQ causes 8237A to issue the HRQ.
DACK0-DACK3	O	**DMA Acknowledge:** DMA Acknowledge is used to notify the individual peripherals when one has been granted a DMA cycle. The sense of these lines is programmable. Reset initializes them to active low.
AEN	O	**Address Enable:** Address Enable enables the 8-bit latch containing the upper 8 address bits onto the system address bus. AEN can also be used to disable other system bus drivers during DMA transfers. AEN is active HIGH.
ADSTB	O	**Address Strobe:** The active high, Address Strobe is used to strobe the upper address byte into an external latch.
$\overline{MEMR}$	O	**Memory Read:** The Memory Read signal is an active low three-state output used to access data from the selected memory location during a DMA Read or a memory-to-memory transfer.
$\overline{MEMW}$	O	**Memory Write:** The Memory Write is an active low three-state output used to write data to the selected memory location during a DMA Write or a memory-to-memory transfer.

FUNCTIONAL DESCRIPTION

The 8237A block diagram includes the major logic blocks and all of the internal registers. The data interconnection paths are also shown. Not shown are the various control signals between the blocks. The 8237A contains 344 bits of internal memory in the form of registers. Figure 3 lists these registers by name and shows the size of each. A detailed description of the registers and their functions can be found under Register Description.

Name	Size	Number
Base Address Registers	16 bits	4
Base Word Count Registers	16 bits	4
Current Address Registers	16 bits	4
Current Word Count Registers	16 bits	4
Temporary Address Register	16 bits	1
Temporary Word Count Register	16 bits	1
Status Register	8 bits	1
Command Register	8 bits	1
Temporary Register	8 bits	1
Mode Registers	6 bits	4
Mask Register	4 bits	1
Request Register	4 bits	1

Figure 3. 8237A Internal Registers

The 8237A contains three basic blocks of control logic. The Timing Control block generates internal timing and external control signals for the 8237A. The Program Command Control block decodes the various commands given to the 8237A by the microprocessor prior to servicing a DMA Request. It also decodes the Mode Control word used to select the type of DMA during the servicing. The Priority Encoder block resolves priority contention between DMA channels requesting service simultaneously.

The Timing Control block derives internal timing from the clock input. In 8237A systems this input will usually be the $\phi 2$ TTL clock from an 8224 or CLK from an 8085AH or 8284A. For 8085AH-2 systems above 3.9 MHz, the 8085 CLK(OUT) does not satisfy 8237A-5 clock LOW and HIGH time requirements. In this case, an external clock should be used to drive the 8237A-5.

DMA Operation

The 8237A is designed to operate in two major cycles. These are called Idle and Active cycles. Each device cycle is made up of a number of states. The 8237A can assume seven separate states, each composed of one full clock period. State I (SI) is the inactive state. It is entered when the 8237A has no valid DMA requests pending. While in SI, the DMA controller is inactive but may be in the Program Condition, being programmed by the processor. State S0 (S0) is the first state of a DMA service. The 8237A has requested a hold but the processor has not yet returned an acknowledge. The 8237A may still be programmed until it receives HLDA from the CPU. An acknowledge from the CPU will signal that DMA transfers may begin. S1, S2, S3 and S4 are the working states of the DMA service. If more time is needed to complete a transfer than is available with normal timing, wait states (SW) can be inserted between S2 or S3 and S4 by the use of the Ready line on the 8237A. Note that the data is transferred directly from the I/O device to memory (or vice versa) with $\overline{IOR}$ and $\overline{MEMW}$ (or $\overline{MEMR}$ and $\overline{IOW}$) being active at the same time. The data is not read into or driven out of the 8237A in I/O-to-memory or memory-to-I/O DMA transfers.

Memory-to-memory transfers require a read-from and a write-to-memory to complete each transfer. The states, which resemble the normal working states, use two digit numbers for identification. Eight states are required for a single transfer. The first four states (S11, S12, S13, S14) are used for the read-from-memory half and the last four states (S21, S22, S23, S24) for the write-to-memory half of the transfer.

IDLE CYCLE

When no channel is requesting service, the 8237A will enter the Idle cycle and perform "SI" states. In this cycle the 8237A will sample the DREQ lines every clock cycle to determine if any channel is requesting a DMA service. The device will also sample $\overline{CS}$, looking for an attempt by the microprocessor to write or read the internal registers of the 8237A. When $\overline{CS}$ is low and HLDA is low, the 8237A enters the Program Condition. The CPU can now establish, change or inspect the internal definition of the part by reading from or writing to the internal registers. Address lines A0–A3 are inputs to the device and select which registers will be read or written. The $\overline{IOR}$ and $\overline{IOW}$ lines are used to select and time reads or writes. Due to the number and size of the internal registers, an internal flip-flop is used to generate an additional bit of address. This bit is used to determine the upper or lower byte of the 16-bit Address and Word Count registers. The flip-flop is reset by Master Clear or Reset. A separate software command can also reset this flip-flop.

Special software commands can be executed by the 8237A in the Program Condition. These commands are decoded as sets of addresses with the $\overline{CS}$ and $\overline{IOW}$. The commands do not make use of the data bus. Instructions include Clear First/Last Flip-FLop and Master Clear.

ACTIVE CYCLE

When the 8237A is in the Idle cycle and a non-masked channel requests a DMA service, the device will output an HRQ to the microprocessor and enter the Active cycle. It is in this cycle that the DMA service will take place, in one of four modes:

Single Transfer Mode — In Single Transfer mode the device is programmed to make one transfer only. The word count will be decremented and the address decremented or incremented following each transfer. When the word count "rolls over" from zero to FFFFH, a Terminal Count (TC) will cause an Autoinitialize if the channel has been programmed to do so.

DREQ must be held active until DACK becomes active in order to be recognized. If DREQ is held active throughout the single transfer, HRQ will go inactive and release the bus to the system. It will again go active and, upon receipt of a new HLDA, another single transfer will be performed, in 8080A, 8085AH, 8088, or 8086 system this will ensure one full machine cycle execution between DMA transfers. Details of timing between the 8237A and other bus control protocols will depend upon the characteristics of the microprocessor involved.

Block Transfer Mode — In Block Transfer mode the device is activated by DREQ to continue making transfers during the service until a TC, caused by word count going to FFFFH, or an external End of Process ($\overline{EOP}$) is encountered. DREQ need only be held active until DACK becomes active. Again, an Autoinitialization will occur at the end of the service if the channel has been programmed for it.

Demand Transfer Mode — In Demand Transfer mode the device is programmed to continue making transfers until a TC or external $\overline{EOP}$ is encountered or until DREQ goes inactive. Thus transfers may continue until the I/O device has exhausted its data capacity. After the I/O device has had a chance to catch up, the DMA service is re-established by means of a DREQ. During the time between services when the microprocessor is allowed to operate, the intermediate values of address and word count are stored in the 8237A Current Address and Current Word Count registers. Only an $\overline{EOP}$ can cause an Autoinitialize at the end of the service. $\overline{EOP}$ is generated either by TC or by an external signal.

Cascade Mode—This mode is used to cascade more than one 8237A together for simple system expansion. The HRQ and HLDA signals from the additional 8237A are connected to the DREQ and DACK signals of a channel of the initial 8237A. This allows the DMA requests of the additional device to propagate through the priority network circuitry of the preceding device. The priority chain is preserved and the new device must wait for its turn to acknowledge requests. Since the cascade channel of the initial 8237A is used only for prioritizing the additional device, it does not output any address or control signals of its own. These could conflict with the outputs of the active channel in the added device. The 8237A will respond to DREQ and DACK but all other outputs except HRQ will be disabled. The ready input is ignored.

Figure 4 shows two additional devices cascaded into an initial device using two of the previous channels. This forms a two level DMA system. More 8237As could be added at the second level by using the remaining channels of the first level. Additional devices can also be added by cascading into the channels of the second level devices, forming a third level.

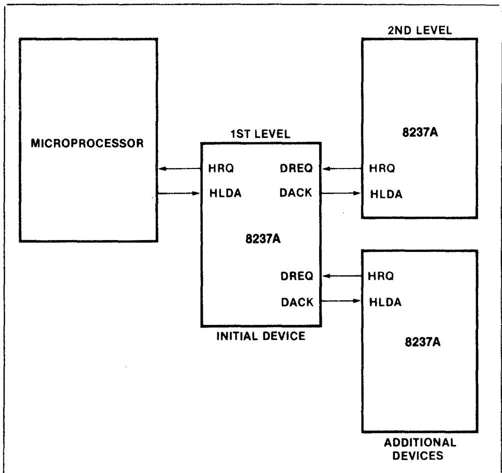


Figure 4. Cascaded 8237As

TRANSFER TYPES

Each of the three active transfer modes can perform three different types of transfers. These are Read, Write and Verify. Write transfers move data from and I/O device to the memory by activating $\overline{\text{MEMW}}$ and $\overline{\text{IOR}}$. Read transfers move data from memory to an I/O device by activating $\overline{\text{MEMR}}$ and $\overline{\text{IOW}}$. Verify transfers are pseudo transfers. The 8237A operates as in Read or Write transfers generating addresses, and responding to EOP, etc. However, the memory and I/O control lines all remain inactive. The ready input is ignored in verify mode.

Memory-to-Memory—To perform block moves of data from one memory address space to another with a minimum of program effort and time, the 8237A includes a memory-to-memory transfer feature. Programming a bit in the Command register selects channels 0 to 1 to operate as memory-to-memory transfer channels. The transfer is initiated by setting the software DREQ for channel 0. The 8237A requests a DMA service in the normal manner. After HLDA is true, the device, using four state transfers in Block Transfer mode, reads data from the memory. The channel 0 Current Address register is the source for the address used and is decremented or incremented in the normal manner. The data byte read from the memory is stored in the 8237A internal Temporary register. Channel 1 then performs a four-state transfer of the data from the Temporary register to memory using the address in its Current Address register and incrementing or decrementing it in the normal manner. The channel 1 current Word Count is decremented. When the word count of channel 1 goes to FFFFH, a TC is generated causing an $\overline{\text{EOP}}$ output terminating the service.

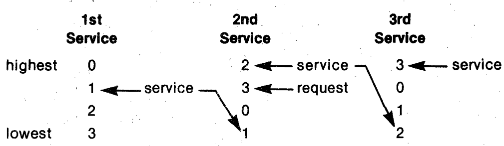
Channel 0 may be programmed to retain the same address for all transfers. This allows a single word to be written to a block of memory.

The 8237A will respond to external $\overline{\text{EOP}}$ signals during memory-to-memory transfers. Data comparators in block search schemes may use this input to terminate the service when a match is found. The timing of memory-to-memory transfers is found in Figure 12. Memory-to-memory operations can be detected as an active AEN with no DACK outputs.

Autoinitialize—By programming a bit in the Mode register, a channel may be set up as an Autoinitialize channel. During Autoinitialize initialization, the original values of the Current Address and Current Word Count registers are automatically restored from the Base Address and Base Word count registers of that channel following $\overline{\text{EOP}}$. The base registers are loaded simultaneously with the current registers by the microprocessor and remain unchanged throughout the DMA service. The mask bit is not altered when the channel is in Autoinitialize. Following Autoinitialize the channel is ready to perform another DMA service, without CPU intervention, as soon as a valid DREQ is detected. In order to Autoninitialize both channels in a memory-to-memory transfer, both word counts should be programmed identically. If interrupted externally, $\overline{\text{EOP}}$ pulses should be applied in both bus cycles.

Priority—The 8237A has two types of priority encoding available as software selectable options. The first is Fixed Priority which fixes the channels in priority order based upon the descending value of their number. The channel with the lowest priority is 3 followed by 2, 1 and the highest priority channel, 0. After the recognition of any one channel for service, the other channels are prevented from interferring with that service until it is completed.

The second scheme is Rotating Priority. The last channel to get service becomes the lowest priority channel with the others rotating accordingly.



With Rotating Priority in a single chip DMA system, any device requesting service is guaranteed to be recognized after no more than three higher priority services have occurred. This prevents any one channel from monopolizing the system.

Compressed Timing — In order to achieve even greater throughput where system characteristics permit, the 8237A can compress the transfer time to two clock cycles. From Figure 11 it can be seen that state S3 is used to extend the access time of the read pulse. By removing state S3, the read pulse width is made equal to the write pulse width and a transfer consists only of state S2 to change the address and state S4 to perform the read/write. S1 states will still occur when A8-A15 need updating (see Address Generation). Timing for compressed transfers is found in Figure 14.

Address Generation — In order to reduce pin count, the 8237A multiplexes the eight higher order address bits on the data lines. State S1 is used to output the higher order address bits to an external latch from which they may be placed on the address bus. The falling edge of Address Strobe (ADSTB) is used to load these bits from the data lines to the latch. Address Enable (AEN) is used to enable the bits onto the address bus through a three-state enable. The lower order address bits are output by the 8237A directly. Lines A0-A7 should be connected to the address bus. Figure 11 shows the time relationships between CLK, AEN, ADSTB, DB0-DB7 and A0-A7.

During Block and Demand Transfer mode services, which include multiple transfers, the addresses generated will be sequential. For many transfers the data held in the external address latch will remain the same. This data need only change when a carry or borrow from A7 to A8 takes place in the normal sequence of addresses. To save time and speed transfers, the 8237A executes S1 states only when updating of A8-A15 in the latch is necessary. This means for long services, S1 states and Address Strobes may occur only once every 256 transfers, a savings of 255 clock cycles for each 256 transfers.

REGISTER DESCRIPTION

Current Address Register — Each channel has a 16-bit Current Address register. This register holds the value of the address used during DMA transfers. The address is automatically incremented or decremented after each transfer and the intermediate values of the address are stored in the Current Address register during the transfer. This register is written or read by the microprocessor in successive 8-bit bytes. It may also be reinitialized by an Autoinitialize back to its original value. Autoinitialize takes place only after an $\overline{\text{EOP}}$.

Current Word Register — Each channel has a 16-bit Current Word Count register. This register determines the number of transfers to be performed. The actual number of transfers will be one more than the number programmed in the Current Word Count register (i.e., programming a count of 100 will result in 101 transfers). The word count is decremented after each transfer. The intermediate value of the word count is stored in the register during the transfer. When the value in the register goes from zero to FFFFH, a TC will be generated. This register is loaded or read in successive 8-bit bytes by the microprocessor in the Program Condition. Following the end of a DMA service it may also be reinitialized by an Autoinitialization back to its original value. Autoinitialize can occur only when an $\overline{\text{EOP}}$ occurs. If it is not Autoinitialized, this register will have a count of FFFFH after TC.

Base Address and Base Word Count Registers — Each channel has a pair of Base Address and Base Word Count registers. These 16-bit registers store the original value of their associated current registers. During Autoinitialize these values are used to restore the current registers to their original values. The base registers are written simultaneously with their corresponding current register in 8-bit bytes in the Program Condition by the microprocessor. These registers cannot be read by the microprocessor.

Command Register — This 8-bit register controls the operation of the 8237A. It is programmed by the microprocessor in the Program Condition and is cleared by Reset or a Master Clear instruction. The following table lists the function of the command bits. See Figure 6 for address coding.

Mode Register — Each channel has a 6-bit Mode register associated with it. When the register is being written to by the microprocessor in the Program Condition, bits 0 and 1 determine which channel Mode register is to be written.

Request Register — The 8237A can respond to requests for DMA service which are initiated by software as well as by a DREQ. Each channel has a request bit associated with it in the 4-bit Request register. These are nonmaskable and subject to prioritization by the Priority Encoder network. Each register bit is set or reset separately under software control or is cleared upon generation of a TC or external $\overline{\text{EOP}}$. The entire register is cleared by a Reset. To set or reset a bit, the software loads the proper form of the data word. See Figure 5 for register address coding. In order to make a software request, the channel must be in Block Mode.

Command Register

7 6 5 4 3 2 1 0 ← Bit Number

Bit	Value	Function
0	0	Memory-to-memory disable
	1	Memory-to-memory enable
1	0	Channel 0 address hold disable
	1	Channel 0 address hold enable
	X	If bit 0 = 0
2	0	Controller enable
	1	Controller disable
3	0	Normal timing
	1	Compressed timing
	X	If bit 0 = 1
4	0	Fixed priority
	1	Rotating priority
5	0	Late write selection
	1	Extended write selection
	X	If bit 3 = 1
6	0	DREQ sense active high
	1	DREQ sense active low
7	0	DACK sense active low
	1	DACK sense active high

Mode Register

7 6 5 4 3 2 1 0 ← Bit Number

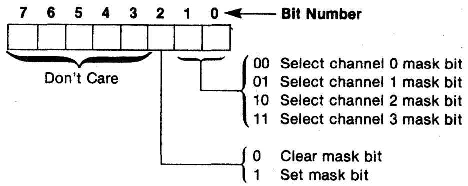
Bits	Value	Function
1, 0	00	Channel 0 select
	01	Channel 1 select
	10	Channel 2 select
	11	Channel 3 select
3, 2	00	Verify transfer
	01	Write transfer
	10	Read transfer
	11	Illegal
	XX	If bits 6 and 7 = 11
4	0	Autoinitialization disable
	1	Autoinitialization enable
5	0	Address increment select
	1	Address decrement select
7, 6	00	Demand mode select
	01	Single mode select
	10	Block mode select
	11	Cascade mode select

Request Register

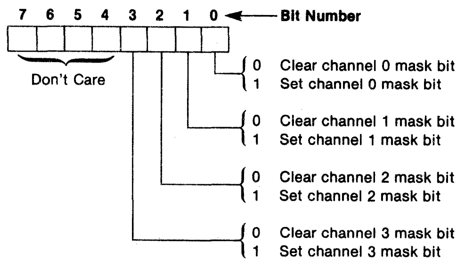
7 6 5 4 3 2 1 0 ← Bit Number

Bits	Value	Function
7–3		Don't Care
1, 0	00	Select channel 0
	01	Select channel 1
	10	Select channel 2
	11	Select channel 3
2	0	Reset request bit
	1	Set request bit

Mask Register — Each channel has associated with it a mask bit which can be set to disable the incoming DREQ. Each mask bit is set when its associated channel produces an $\overline{EOP}$ if the channel is not programmed for Autoinitialize. Each bit of the 4-bit Mask register may also be set or cleared separately under software control. The entire register is also set by a Reset. This disables all DMA requests until a clear Mask register instruction allows them to occur. The instruction to separately set or clear the mask bits is similar in form to that used with the Request register. See Figure 5 for instruction addressing.



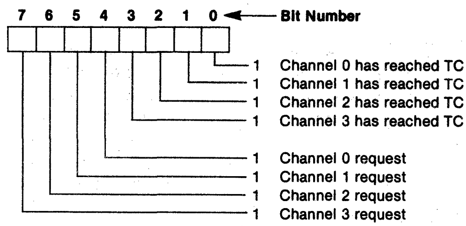
All four bits of the Mask register may also be written with a single command.



Register	Operation	Signals						
		$\overline{CS}$	$\overline{IOR}$	$\overline{IOW}$	A3	A2	A1	A0
Command	Write	0	1	0	1	0	0	0
Mode	Write	0	1	0	1	0	1	1
Request	Write	0	1	0	1	0	0	1
Mask	Set/Reset	0	1	0	1	0	1	0
Mask	Write	0	1	0	1	1	1	1
Temporary	Read	0	0	1	1	1	0	1
Status	Read	0	0	1	1	0	0	0

Figure 5. Definition of Register Codes

Status Register — The Status register is available to be read out of the 8237A by the microprocessor. It contains information about the status of the devices at this point. This information includes which channels have reached a terminal count and which channels have pending DMA requests. Bits 0–3 are set every time a TC is reached by that channel or an external $\overline{EOP}$ is applied. These bits are cleared upon Reset and on each Status Read. Bits 4–7 are set whenever their corresponding channel is requesting service.



Temporary Register — The Temporary register is used to hold data during memory-to-memory transfers. Following the completion of the transfers, the last word moved can be read by the microprocessor in the Program Condition. The Temporary register always contains the last byte transferred in the previous memory-to-memory operation, unless cleared by a Reset.

Software Commands—These are additional special software commands which can be executed in the Program Condition. They do not depend on any specific bit pattern on the data bus. The three software commands are:

Clear First/Last Flip-Flop: This command is executed prior to writing or reading new address or word count information to the 8237A. This initializes the flip-flop to a known state so that subsequent accesses to register contents by the microprocessor will address upper and lower bytes in the correct sequence.

Master Clear: This software instruction has the same effect as the hardware Reset. The Command, Status, Request, Temporary, and Internal First/Last Flip-Flop registers are cleared and the Mask register is set. The 8237A will enter the Idle cycle.

Clear Mask Register: This command clears the mask bits of all four channels, enabling them to accept DMA requests.

Figure 6 lists the address codes for the software commands:

Signals						Operation
A3	A2	A1	A0	$\overline{IOR}$	$\overline{IOW}$	
1	0	0	0	0	1	Read Status Register
1	0	0	0	1	0	Write Command Register
1	0	0	1	0	1	Illegal
1	0	0	1	1	0	Write Request Register
1	0	1	0	0	1	Illegal
1	0	1	0	1	0	Write Single Mask Register Bit
1	0	1	1	0	1	Illegal
1	0	1	1	1	0	Write Mode Register
1	1	0	0	0	1	Illegal
1	1	0	0	1	0	Clear Byte Pointer Flip/Flop
1	1	0	1	0	1	Read Temporary Register
1	1	0	1	1	0	Master Clear
1	1	1	0	0	1	Illegal
1	1	1	0	1	0	Clear Mask Register
1	1	1	1	0	1	Illegal
1	1	1	1	1	0	Write All Mask Register Bits

Figure 6. Software Command Codes

Channel	Register	Operation	Signals							Internal Flip-Flop	Data Bus DB0-DB7
			$\overline{CS}$	$\overline{IOR}$	$\overline{IOW}$	A3	A2	A1	A0		
0	Base and Current Address	Write	0	1	0	0	0	0	0	0	A0-A7
			0	1	0	0	0	0	0	1	A8-A15
	Current Address	Read	0	0	1	0	0	0	0	0	A0-A7
			0	0	1	0	0	0	0	1	A8-A15
	Base and Current Word Count	Write	0	1	0	0	0	0	1	0	W0-W7
			0	1	0	0	0	0	1	1	W8-W15
	Current Word Count	Read	0	0	1	0	0	0	1	0	W0-W7
			0	0	1	0	0	0	1	1	W8-W15
1	Base and Current Address	Write	0	1	0	0	0	1	0	0	A0-A7
			0	1	0	0	0	1	0	1	A8-A15
	Current Address	Read	0	0	1	0	0	1	0	0	A0-A7
			0	0	1	0	0	1	0	1	A8-A15
	Base and Current Word Count	Write	0	1	0	0	0	1	1	0	W0-W7
			0	1	0	0	0	1	1	1	W8-W15
	Current Word Count	Read	0	0	1	0	0	1	1	0	W0-W7
			0	0	1	0	0	1	1	1	W8-W15
2	Base and Current Address	Write	0	1	0	0	1	0	0	0	A0-A7
			0	1	0	0	1	0	0	1	A8-A15
	Current Address	Read	0	0	1	0	1	0	0	0	A0-A7
			0	0	1	0	1	0	0	1	A8-A15
	Base and Current Word Count	Write	0	1	0	0	1	0	1	0	W0-W7
			0	1	0	0	1	0	1	1	W8-W15
	Current Word Count	Read	0	0	1	0	1	0	1	0	W0-W7
			0	0	1	0	1	0	1	1	W8-W15
3	Base and Current Address	Write	0	1	0	0	1	1	0	0	A0-A7
			0	1	0	0	1	1	0	1	A8-A15
	Current Address	Read	0	0	1	0	1	1	0	0	A0-A7
			0	0	1	0	1	1	0	1	A8-A15
	Base and Current Word Count	Write	0	1	0	0	1	1	1	0	W0-W7
			0	1	0	0	1	1	1	1	W8-W15
	Current Word Count	Read	0	0	1	0	1	1	1	0	W0-W7
			0	0	1	0	1	1	1	1	W8-W15

Figure 7. Word Count and Address Register Command Codes

PROGRAMMING

The 8237A will accept programming from the host processor any time that HLDA is inactive; this is true even if HRQ is active. The responsibility of the host is to assure that programming and HLDA are mutually exclusive. Note that a problem can occur if a DMA request occurs, on an unmasked channel while the 8237A is being programmed. For instance, the CPU may be starting to reprogram the two byte Address register of channel 1 when channel 1 receives a DMA request. If the 8237A is enabled (bit 2 in the command register is 0) and channel 1 is unmasked, a DMA service will occur after only one byte of the Address register has been reprogrammed. This can be avoided by disabling the controller (setting bit 2 in the command register) or masking the channel before programming any other registers. Once the programming is complete, the controller can be enabled/unmasked.

After power-up it is suggested that all internal locations, especially the Mode registers, be loaded with some valid value. This should be done even if some channels are unused.

APPLICATION INFORMATION

Figure 8 shows a convenient method for configuring a DMA system with the 8237A controller and an 8080A/8085AH microprocessor system. The multimode DMA controller issues a HRQ to the processor whenever there is at least one valid DMA request from a peripheral device. When the processor replies with a HLDA signal, the 8237A takes control of the address bus, the data bus and the control bus. The address for the first transfer operation comes out in two bytes — the least significant 8 bits on the eight address outputs and the most significant 8 bits on the data bus. The contents of the data bus are then latched into the 8282 8-bit latch to complete the full 16 bits of the address bus. The 8282 is a high speed, 8-bit, three-state latch in a 20-pin package. After the initial transfer takes place, the latch is updated only after a carry or borrow is generated in the least significant address byte. Four DMA channels are provided when one 8237A is used.

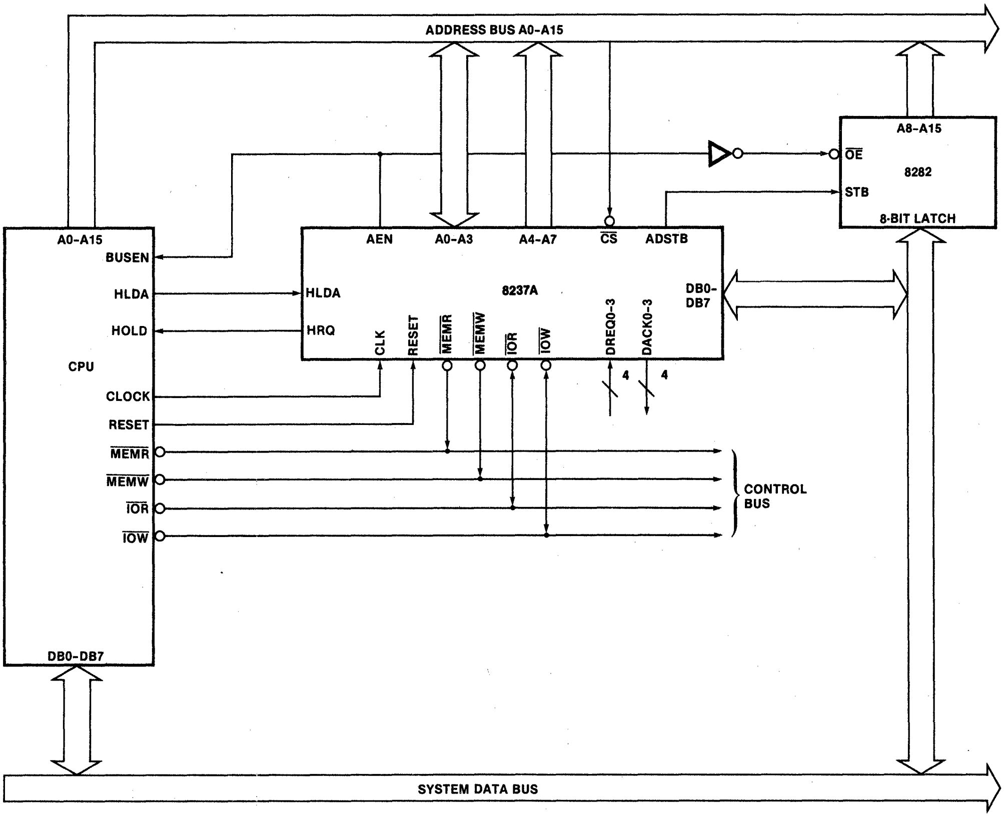


Figure 8. 8237A System Interface

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias . . . . 0°C to 70°C
Storage Temperature . . . . – 65°C to + 150°C
Voltage on any Pin with
  Respect to Ground . . . . – 0.5 to 7V
Power Dissipation . . . . 1.5 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS ($T_A = 0°C$ to $70°C$, $V_{CC} = 5.0V \pm 5\%$, GND = 0V)

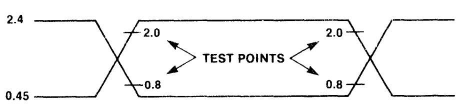
Symbol	Parameter	Min.	Typ.(1)	Max.	Unit	Test Conditions
V_{OH}	Output High Voltage	2.4			V	I_{OH} = -200 μA
		3.3			V	I_{OH} = -100 μA (HRQ Only)
V_{OL}	Output LOW Voltage			.40	V	I_{OL} = 2.0mA (data Bus, $\overline{EOP}$) I_{OL} = 3.2mA (other outputs) I_{OL} = 2.5mA (ADSTB) (Note 8)
V_{IH}	Input HIGH Voltage	2.0		V_{CC} + 0.5	V	(Note 8)
V_{IL}	Input LOW Voltage	-0.5		0.8	V	
I_{LI}	Input Load Current			±10	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current			±10	μA	$0.45V \leq V_{OUT} \leq V_{CC}$
I_{CC}	V_{CC} Supply Current		110	130	mA	$T_A = +25°C$
			130	150	mA	$T_A = 0°C$
C_O	Output Capacitance		4	8	pF	
C_1	Input Capacitance		8	15	pF	fc = 1.0 MHz, Inputs = 0V
C_{IO}	I/O Capacitance		10	18	pF	

NOTES:

1. Typical values are for T_A = 25°C, nominal supply voltage and nominal processing parameters.
2. Input timing parameters assume transition times fo 20 ns or less. Waveform measurement points for both input and output signals are 2.0V for HIGH and 0.8V for LOW, unless otherwise noted.
3. Output loading is 1 TTL gate plus 150pF capacitance, unless otherwise noted.
4. The net $\overline{IOW}$ or $\overline{MEMW}$ Pulse width for normal write will be TCY-100 ns and for extended write will be 2TCY-100 ns. The net $\overline{IOR}$ or $\overline{MEMR}$ pulse width for normal read will be 2TCY-50 ns and for compressed read will be TCY-50 ns.
5. TDQ is specified for two different output HIGH levels. TDQ1 is measured at 2.0V. TDQ2 is measured at 3.3V. The value for TDQ2 assumes an external 3.3KΩ pull-up resistor connected from HRQ to V_{CC}.
6. DREQ should be held active until DACK is returned.
7. DREQ and DACK signals may be active high or active low. Timing diagrams assume the active high mode.
8. The values of V_{OL} and V_{IH} have been changed from the 1985 specification to allow more design margin.
9. Successive read and/or write operations by the external processor to program or examine the controller must be timed to allow at least 600 ns for the 8237A, at least 500 ns for the 8237A-4 and at least 400 ns for the 8237A-5, as recovery time between active read or write pulses. The same recovery time is needed between an active read or write pulse followed by a DMA transfer.
10. $\overline{EOP}$ is an open collector output. This parameter assumes the presence of a 2.2K pullup to V_{CC}.
11. Pin 5 is an input that should always be at a logic high level. An internal pull-up resistor will establish a logic high when the pin is left floating. It is recommended however, that pin 5 be tied to V_{CC}.
12. Output Loading on the Data Bus is I_{TTL} Gate plus 100pF capacitance.

A.C. TESTING INPUT, OUTPUT WAVEFORM

INPUT/OUTPUT



A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0.8V FOR A LOGIC "0." (Note 2)

A.C. CHARACTERISTICS—DMA (MASTER) MODE (T_A = 0°C to 70°C, V_{CC} = +5V ±5%, GND = 0V)

Symbol	Parameter	8237A		8237A-4		8237A-5		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	
TAEL	AEN HIGH from CLK LOW (S1) Delay Time		300		225		200	ns
TAET	AEN LOW from CLK HIGH (SI) Delay Time		200		150		130	ns
TAFAB	ADR Active to Float Delay from CLK HIGH		150		120		90	ns
TAFC	$\overline{READ}$ or $\overline{WRITE}$ Float from CLK HIGH		150		120		120	ns
TAFDB	DB Active to Float Delay from CLK HIGH		250		190		170	ns
TAHR	ADR from $\overline{READ}$ HIGH Hold Time	TCY–100		TCY–100		TCY–100		ns
TAHS	DB from ADSTB LOW Hold Time	40		40		30		ns
TAHW	ADR from $\overline{WRITE}$ HIGH Hold Time	TCY–50		TCY–50		TCY–50		ns
TAK	DACK Valid from CLK LOW Delay Time (Note 7)		250		220		170	ns
	$\overline{EOP}$ HIGH from CLK HIGH Delay Time (Note 10)		250		190		170	ns
	$\overline{EOP}$ LOW from CLK HIGH Delay Time		250		190		170	ns
TASM	ADR Stable from CLK HIGH		250		190		170	ns
TASS	DB to ADSTB LOW Setup Time	100		100		100		ns
TCH	Clock High Time (Transitions ≤ 10 ns)	120		100		80		ns
TCL	Clock LOW Time (Transitions ≤ 10 ns)	150		110		68		ns
TCY	CLK Cycle Time	320		250		200		ns
TDCL	CLK HIGH to $\overline{READ}$ or $\overline{WRITE}$ LOW Delay (Note 4)		270		200		190	ns
TDCTR	$\overline{READ}$ HIGH from CLK HIGH (S4) Delay Time (Note 4)		270		210		190	ns
TDCTW	$\overline{WRITE}$ HIGH from CLK HIGH (S4) Delay Time (Note 4)		200		150		130	ns
TDQ1	HRQ Valid from CLK HIGH Delay Time (Note 5)		160		120		120	ns
TDQ2			250		190		120	ns
TEPS	$\overline{EOP}$ LOW from CLK LOW Setup Time	60		45		40		ns
TEPW	$\overline{EOP}$ Pulse Width	300		225		220		ns
TFAAB	ADR Float to Active Delay from CLK HIGH		250		190		170	ns
TFAC	$\overline{READ}$ or $\overline{WRITE}$ Active from CLK HIGH		200		150		150	ns
TFADB	DB Float to Active Delay from CLK HIGH		300		225		200	ns
THS	HLDA Valid to CLK HIGH Setup Time	100		75		75		ns
TIDH	Input Data from $\overline{MEMR}$ HIGH Hold Time	0		0		0		ns
TIDS	Input Data to $\overline{MEMR}$ HIGH Setup Time	250		190		170		ns
TODH	Output Data from $\overline{MEMW}$ HIGH Hold Time	20		20		10		ns
TODV	Output Data Valid to $\overline{MEMW}$ HIGH	200		125		125		ns
TQS	DREQ to CLK LOW (SI, S4) Setup Time (Note 7)	0		0		0		ns
TRH	CLK to READY LOW Hold Time	20		20		20		ns
TRS	READY to CLK LOW Setup Time	100		60		60		ns
TSTL	ADSTB HIGH from CLK HIGH Delay Time		200		150		130	ns
TSTT	ADSTB LOW from CLK HIGH Delay Time		140		110		90	ns

A.C. CHARACTERISTICS—PERIPHERAL (SLAVE) MODE (T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V)

Symbol	Parameter	8237A Min.	8237A Max.	8237A-4 Min.	8237A-4 Max.	8237A-5 Min.	8237A-5 Max.	Unit
TAR	ADR Valid or $\overline{\text{CS}}$ LOW to $\overline{\text{READ}}$ LOW	50		50		50		ns
TAW	ADR Valid to $\overline{\text{WRITE}}$ HIGH Setup Time	200		150		130		ns
TCW	CS LOW to $\overline{\text{WRITE}}$ HIGH Setup Time	200		150		130		ns
TDW	Data Valid to $\overline{\text{WRITE}}$ HIGH Setup Time	200		150		130		ns
TRA	ADR or CS Hold from $\overline{\text{READ}}$ HIGH	0		0		0		ns
TRDE	Data Access from $\overline{\text{READ}}$ LOW (Note 12)		200		200		140	ns
TRDF	DB Float Delay from $\overline{\text{READ}}$ HIGH	20	100	20	100	0	70	ns
TRSTD	Power Supply HIGH to RESET LOW Setup Time	500		500		500		ns
TRSTS	RESET to First $\overline{\text{IOWR}}$	2TCY		2TCY		2TCY		ns
TRSTW	RESET Pulse Width	300		300		300		ns
TRW	$\overline{\text{READ}}$ Width	300		250		200		ns
TWA	ADR from $\overline{\text{WRITE}}$ HIGH Hold Time	20		20		20		ns
TWC	CS HIGH from $\overline{\text{WRITE}}$ HIGH Hold Time	20		20		20		ns
TWD	Data from $\overline{\text{WRITE}}$ HIGH Hold Time	30		30		30		ns
TWWS	Write Width	200		200		160		ns

WAVEFORMS

SLAVE MODE WRITE TIMING

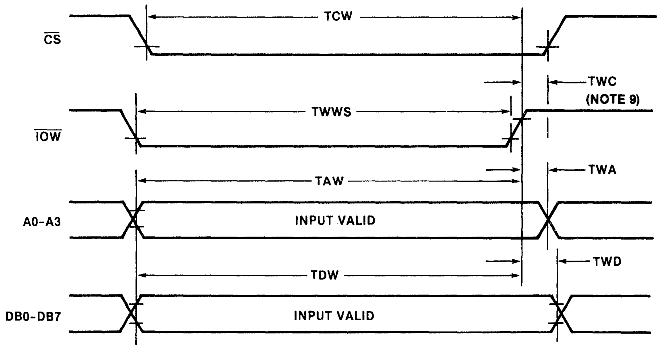


Figure 9. Slave Mode Write

SLAVE MODE READ TIMING

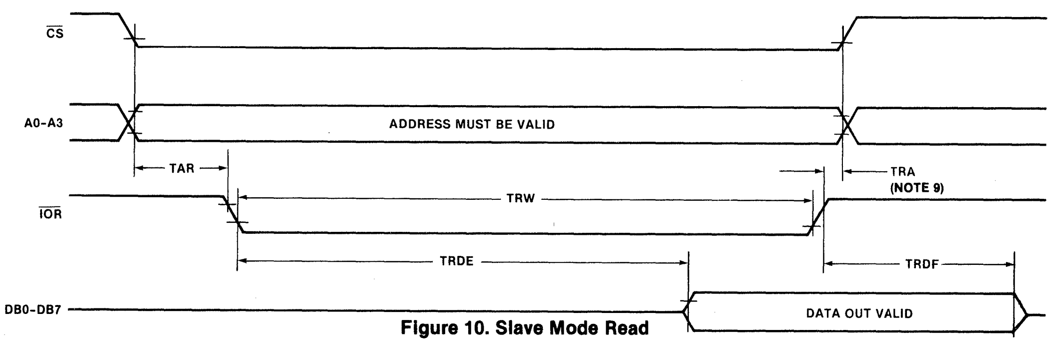


Figure 10. Slave Mode Read

WAVEFORMS (Continued)

DMA TRANSFER TIMING

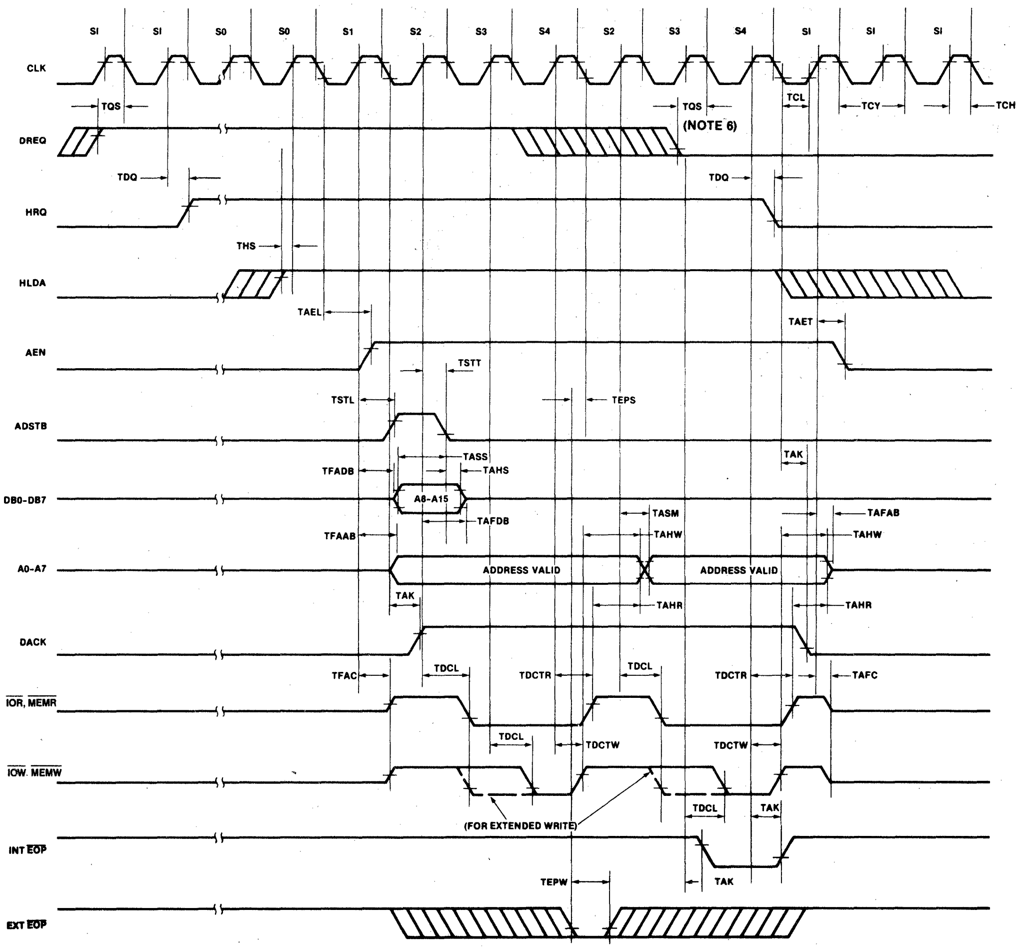


Figure 11. DMA Transfer

WAVEFORMS (Continued)

MEMORY-TO-MEMORY TRANSFER TIMING

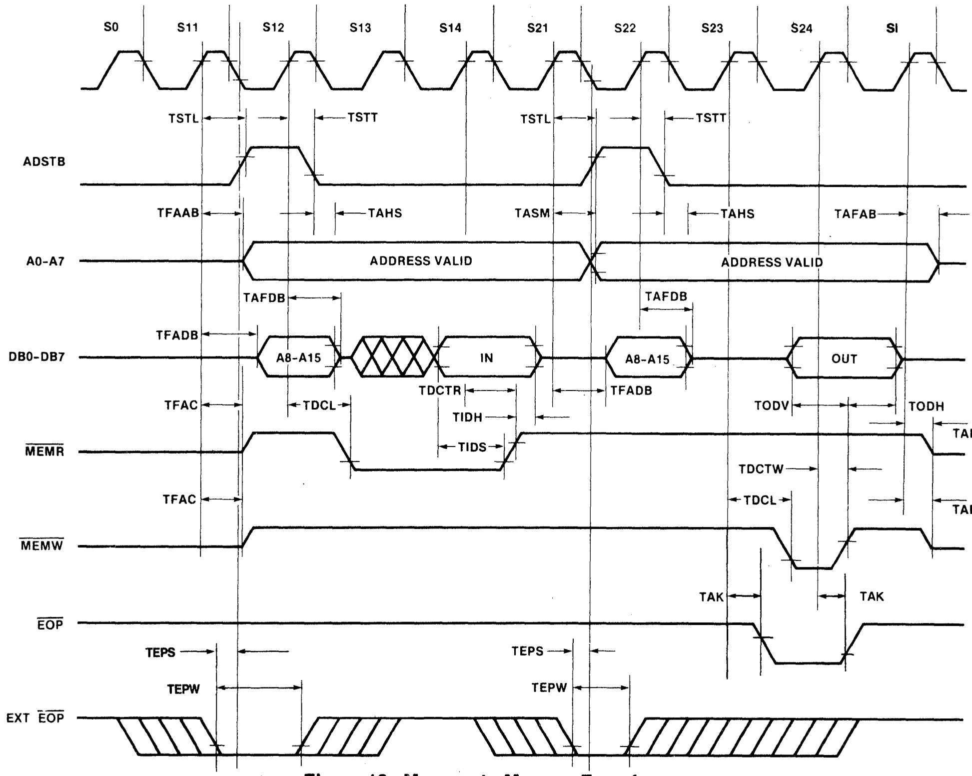


Figure 12. Memory-to-Memory Transfer

READY TIMING

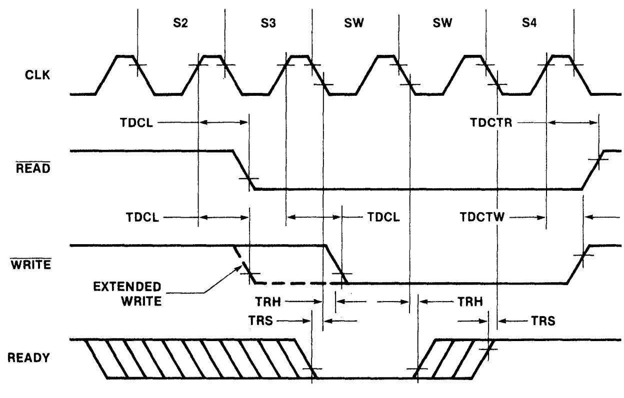


Figure 13. Ready

WAVEFORMS (Continued)

COMPRESSED TRANSFER TIMING

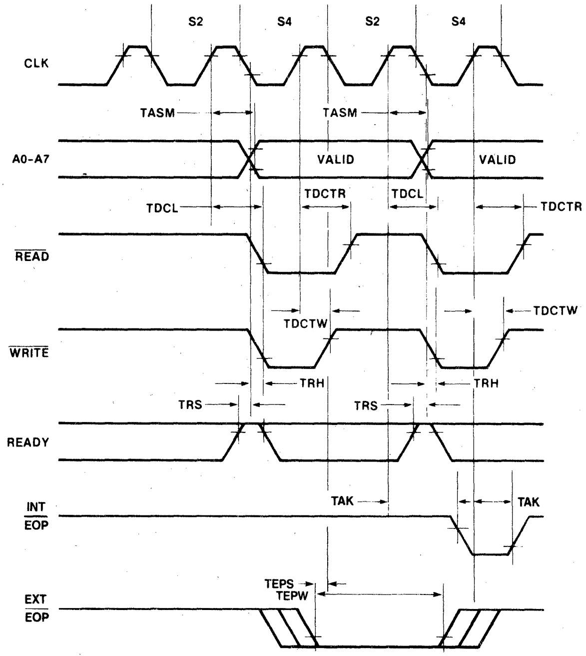


Figure 14. Compressed Transfer

RESET TIMING

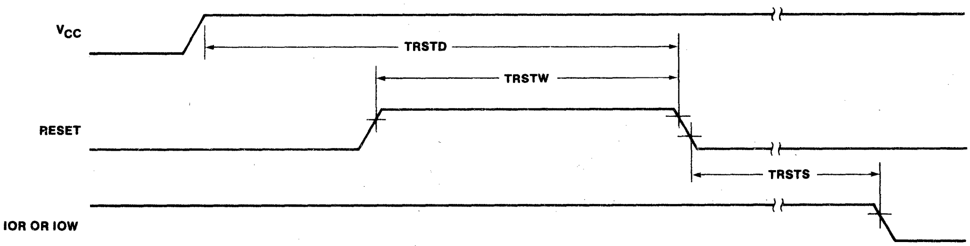


Figure 15. Reset

82C37A-5 CHMOS HIGH PERFORMANCE PROGRAMMABLE DMA CONTROLLER

- **Pin Compatible with NMOS 8237A-5**
- **Enable/Disable Control of Individual DMA Requests**
- **Four Independent DMA Channels**
- **Independent Autoinitialization of all Channels**
- **Memory-to-Memory Transfers**
- **Memory Block Initialization**
- **Address Increment or Decrement**
- **High performance: 5 MHz Speed Transfers up to 1.6 MBytes/Second**
- **Directly Expandable to any Number of Channels**
- **End of Process Input for Terminating Transfers**
- **Software DMA Requests**
- **Independent Polarity Control for DREQ and DACK Signals**
- **Will Be Available in 40-Lead Plastic DIP and 44-Lead PLCC Packages**
  (See Packaging Spec., Order #231369)

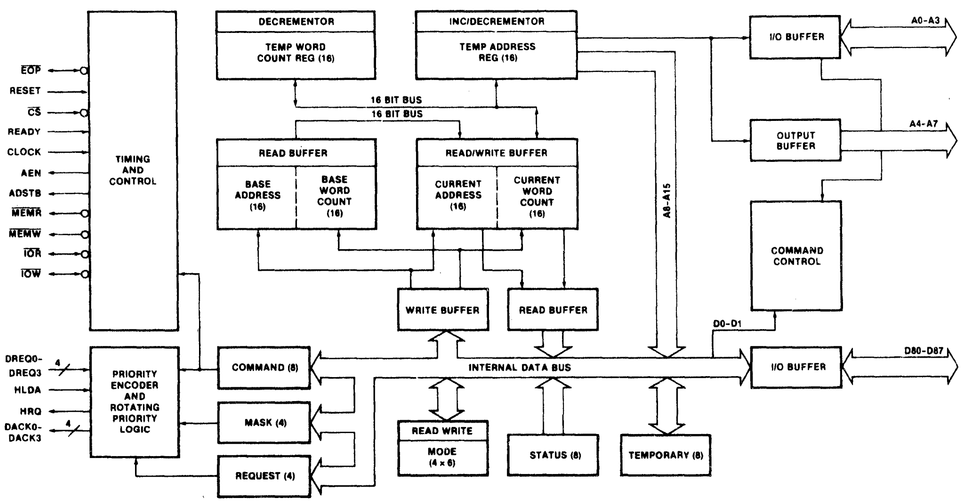
The Intel 82C37A-5 Multimode Direct Memory Access (DMA) Controller is a CHMOS peripheral interface circuit for microprocessor systems. It is designed to improve system performance by allowing external devices to directly transfer information from the system memory. Memory-to-memory transfer capability is also provided. The 82C37A-5 offers a wide variety of programmable control features to enhance data throughput and system optimization and to allow dynamic reconfiguration under program control.

The 82C37A-5 is designed to be used in conjunction with an external 8-bit address register. It contains four independent channels and may be expanded to any number of channels by cascading additional controller chips.

The three basic transfer modes allow programmability of the types of DMA service by the user. Each channel can be individually programmed to Autoinitialize to its original condition following an End of Process ($\overline{EOP}$).

Each channel has a full 64K address and word count capability.

The 82C37A-5 will not be available from Intel until 2nd half 1986.

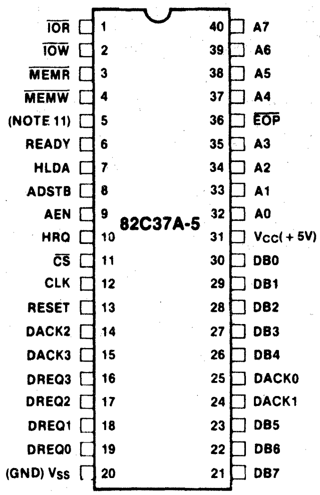


231202–1

Figure 1. Block Diagram

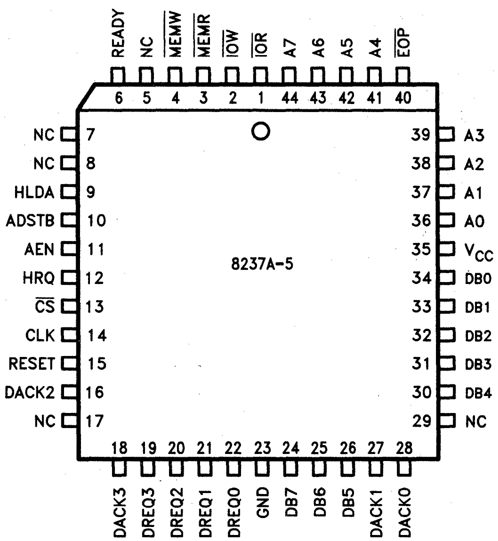
Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel. **November 1985**

Order Number: 231202-002



231202-2

Figure 2a. 8237A 40-Lead DIP Configuration



231202-10

Figure 2b. 8237A 44-Lead PLCC Pin Configuration

Table 1. Pin Description

Symbol	Type	Name and Function
V_{CC}		**POWER:** +5 volt supply.
V_{SS}		**GROUND:** Ground.
CLK	I	**CLOCK INPUT:** Clock Input controls the internal operations of the 82C37A-5 and its rate of data transfers. The input may be driven at up to 5 MHz for the 82C37A-5.
$\overline{CS}$	I	**CHIP SELECT:** Chip Select is an active low input used to select the 82C37A-5 as an I/O device during the Idle cycle. This allows CPU communication on the data bus.
RESET	I	**RESET:** Reset is an active high input which clears the Command, Status, Request and Temporary registers. It also clears the first/last flip-flop and sets the Mask register. Following a Reset the device is in the Idle cycle.
READY	I	**READY:** Ready is an input used to extend the memory read and write pulses from the 82C37A-5 to accommodate slow memories or I/O peripheral devices. Ready must not make transitions during its specified setup/hold time.
HLDA	I	**HOLD ACKNOWLEDGE:** The active high Hold Acknowledge from the CPU indicates that it has relinquished control of the system busses.
DREQ0–DREQ3	I	**DMA REQUEST:** The DMA Request lines are individual asynchronous channel request inputs used by peripheral circuits to obtain DMA service. In fixed Priority, DREQ0 has the highest priority and DREQ3 has the lowest priority. A request is generated by activating the DREQ line of a channel. DACK will acknowledge the recognition of DREQ signal. Polarity of DREQ is programmable. Reset initializes these lines to active high. DREQ must be maintained until the corresponding DACK goes active.
DB0–DB7	I/O	**DATA BUS:** The Data Bus lines are bidirectional three-state signals connected to the system data bus. The outputs are enabled in the Program condition during the I/O Read to output the contents of an Address register, a Status register, the Temporary register or a Word Count register to the CPU. The outputs are disabled and the inputs are read during an I/O Write cycle when the CPU is programming the 82C37A-5 control registers. During DMA cycles the most significant 8 bits of the address are output onto the data bus to be strobed into an external latch by ADSTB. In memory-to-memory operations, data from the memory comes into the 82C37A-5 on the data bus during the read-from-memory transfer. In the write-to-memory transfer, the data bus outputs place the data into the new memory location.
$\overline{IOR}$	I/O	**I/O READ:** I/O Read is a bidirectional active low three-state line. In the Idle cycle, it is an input control signal used by the CPU to read the control registers. In the Active cycle, it is an output control signal used by the 82C37A-5 to access data from a peripheral during a DMA Write transfer.
$\overline{IOW}$	I/O	**I/O WRITE:** I/O Write is a bidirectional active low three-state line. In the Idle cycle, it is an input control signal used by the CPU to load information into the 82C37A-5. In the Active cycle, it is an output control signal used by the 82C37A-5 to load data to the peripheral during a DMA Read transfer.

Table 1. Pin Description (Continued)

Symbol	Type	Name and Function
$\overline{EOP}$	I/O	**END OF PROCESS:** End of Process is an active low bidirectional signal. Information concerning the completion of DMA services is available at the bidirectional $\overline{EOP}$ pin. The 82C37A-5 allows an external signal to terminate an active DMA service. This is accomplished by pulling the $\overline{EOP}$ input low with an external $\overline{EOP}$ signal. The 82C37A-5 also generates a pulse when the terminal count (TC) for any channel is reached. This generates an $\overline{EOP}$ signal which is output through the $\overline{EOP}$ Line. The reception of $\overline{EOP}$, either internal or external, will cause the 82C37A-5 to terminate the service, reset the request, and, if Autoinitialize is enabled, to write the base registers to the current registers of that channel. The mask bit and TC bit in the status word will be set for the currently active channel by $\overline{EOP}$ unless the channel is programmed for Autoinitialize. In that case, the mask bit remains unchanged. During memory-to-memory transfers, $\overline{EOP}$ will be output when the TC for channel 1 occurs. $\overline{EOP}$ should be tied high with a pull-up resistor if it is not used to prevent erroneous end of process inputs.
A0–A3	I/O	**ADDRESS:** The four least significant address lines are bidirectional three-state signals. In the Idle cycle they are inputs and are used by the CPU to address the register to be loaded or read. In the Active cycle they are outputs and provide the lower 4 bits of the output address.
A4–A7	O	**ADDRESS:** The four most significant address lines are three-state outputs and provide 4 bits of address. These lines are enabled only during the DMA service.
HRQ	O	**HOLD REQUEST:** This is the Hold Request to the CPU and is used to request control of the system bus. If the corresponding mask bit is clear, the presence of any valid DREQ causes 82C37A-5 to issue the HRQ. After HRQ goes active at least one clock cycle (TCY) must occur before HLDA goes active.
DACK0–DACK3	O	**DMA ACKNOWLEDGE:** DMA Acknowledge is used to notify the individual peripherals when one has been granted a DMA cycle. The sense of these lines is programmable. Reset initializes them to active low.
AEN	O	**ADDRESS ENABLE:** Address Enable enables the 8-bit latch containing the upper 8 address bits onto the system address bus. AEN can also be used to disable other system bus drivers during DMA transfers. AEN is active HIGH.
ADSTB	O	**ADDRESS STROBE:** The active high, Address Strobe is used to strobe the upper address byte into an external latch.
$\overline{MEMR}$	O	**MEMORY READ:** The Memory Read signal is an active low three-state output used to access data from the selected memory location during a DMA Read or a memory-to-memory transfer.
$\overline{MEMW}$	O	**MEMORY WRITE:** The Memory Write is an active low three-state output used to write data to the selected memory location during a DMA Write or a memory-to-memory transfer.

FUNCTIONAL DESCRIPTION

The 82C37A-5 block diagram includes the major logic blocks and all of the internal registers. The data interconnection paths are also shown. Not shown are the various control signals between the blocks. The 82C37A-5 contains 344 bits of internal memory in the form of registers. Figure 3 lists these registers by name and shows the size of each. A detailed description of the registers and their functions can be found under Register Description.

Name	Size	Number
Base Address Registers	16 bits	4
Base Word Count Registers	16 bits	4
Current Address Registers	16 bits	4
Current Word Count Registers	16 bits	4
Temporary Address Register	16 bits	1
Temporary Word Count Register	16 bits	1
Status Register	8 bits	1
Command Register	8 bits	1
Temporary Register	8 bits	1
Mode Registers	6 bits	4
Mask Register	4 bits	1
Request Register	4 bits	1

Figure 3. 82C37A-5 Internal Registers

The 82C37A-5 contains three basic blocks of control logic. The Timing Control block generates internal timing and external control signals for the 82C37A-5. The Program Command Control block decodes the various commands given to the 82C37A-5 by the microprocessor prior to servicing a DMA Request. It also decodes the Mode Control word used to select the type of DMA during the servicing. The Priority Encoder block resolves priority contention between DMA channels requesting service simultaneously.

The Timing Control block derives internal timing from the clock input. In 82C37A-5 systems this input will usually be the $\phi 2$ TTL clock from an 8224 or CLK from an 8085AH or 82C84A. For 8085AH-2 systems above 3.9 MHz, the 8085 CLK(OUT) does not satisfy 82C37A-5 clock LOW and HIGH time requirements. In this case, an external clock should be used to drive the 82C37A-5.

DMA Operation

The 82C37A-5 is designed to operate in two major cycles. These are called Idle and Active cycles. Each device cycle is made up of a number of states. The 82C37A-5 can assume seven separate states, each composed of one full clock period. State 1 (S1) is the inactive state. It is entered when the 82C37A-5 has no valid DMA requests pending. While in S1, the DMA controller is inactive but may be in the Program Condition, being programmed by the processor. State S0 (S0) is the first state of a DMA service. The 82C37A-5 has requested a hold but the processor has not yet returned an acknowledge. The 82C37A-5 may still be programmed until it receives HLDA from the CPU. An acknowledge from the CPU will signal that DMA transfers may begin. S1, S2, S3 and S4 are the working states of the DMA service. If more time is needed to complete a transfer than is available with normal timing, wait states (SW) can be inserted between S2 or S3 and S4 by the use of the Ready line on the 82C37A-5. Note that the data is transferred directly from the I/O device to memory (or vice versa) with $\overline{IOR}$ and $\overline{MEMW}$ (or $\overline{MEMR}$ and $\overline{IOW}$) being active at the same time. The data is not read into or driven out of the 82C37A-5 in I/O-to-memory or memory-to-I/O DMA transfers.

Memory-to-memory transfers require a read-from and a write-to-memory to complete each transfer. The states, which resemble the normal working states, use two digit numbers for identification. Eight states are required for a single transfer. The first four states (S11, S12, S13, S14) are used for the read-from-memory half and the last four states (S21, S22, S23, S24) for the write-to-memory half of the transfer.

IDLE CYCLE

When no channel is requesting service, the 82C37A-5 will enter the Idle cycle and perform "S1" states. In this cycle the 82C37A-5 will sample the DREQ lines every clock cycle to determine if any channel is requesting a DMA service. The device will also sample $\overline{CS}$, looking for an attempt by the microprocessor to write or read the internal registers of the 82C37A-5. When $\overline{CS}$ is low and HLDA is low, the 82C37A-5 enters the Program Condition. The CPU can now establish, change or inspect the internal definition of the part by reading from or writing to the internal registers. Address lines A0–A3 are inputs to the device and select which registers will be read or written. The $\overline{IOR}$ and $\overline{IOW}$ lines are used to select and time reads or writes. Due to the number and size of the internal registers, an internal flip-flop is used to generate an additional bit of address. This bit is used to determine the upper or lower byte of the 16-bit Address and Word Count registers. The flip-flop is reset by Master Clear or Reset. A separate software command can also reset this flip-flop.

Special software commands can be executed by the 82C37A-5 in the Program Condition. These commands are decoded as sets of addresses with the $\overline{CS}$ and $\overline{IOW}$. The commands do not make use of the data bus. Instructions include Clear First/Last Flip-Flop and Master Clear.

ACTIVE CYCLE

When the 82C37A-5 is in the Idle cycle and a non-masked channel requests a DMA service, the device will output an HRQ to the microprocessor and enter the Active cycle. It is in this cycle that the DMA service will take place, in one of four modes:

Single Transfer Mode — In Single Transfer mode the device is programmed to make one transfer only. The word count will be decremented and the address decremented or incremented following each transfer. When the word count "rolls over" from zero to FFFFH, a Terminal Count (TC) will cause an Autoinitialize if the channel has been programmed to do so.

DREQ must be held active until DACK becomes active in order to be recognized. If DREQ is held active throughout the single transfer, HRQ will go inactive and release the bus to the system. It will again go active and, upon receipt of a new HLDA, another single transfer will be performed, in 8080A, 8085AH, 80C88, or 80C86 system this will ensure one full machine cycle execution between DMA transfers. Details of timing between the 82C37A-5 and other bus control protocols will depend upon the characteristics of the microprocessor involved.

Block Transfer Mode — In Block Transfer mode the device is activated by DREQ to continue making transfers during the service until a TC, caused by word count going to FFFFH, or an external End of Process (EOP) is encountered. DREQ need only be held active until DACK becomes active. Again, an Autoinitialization will occur at the end of the service if the channel has been programmed for it.

Demand Transfer Mode — In Demand Transfer mode the device is programmed to continue making transfers until a TC or external $\overline{EOP}$ is encountered or until DREQ goes inactive. Thus transfers may continue until the I/O device has exhausted its data capacity. After the I/O device has had a chance to catch up, the DMA service is re-established by means of a DREQ. During the time between services when the microprocessor is allowed to operate, the intermediate values of address and word count are stored in the 82C37A-5 Current Address and Current Word Count registers. Only an $\overline{EOP}$ can cause an Autoinitialize at the end of the service. $\overline{EOP}$ is generated either by TC or by an external signal.

Cascade Mode — This mode is used to cascade more than one 82C37A-5 together for simple system expansion. The HRQ and HLDA signals from the additional 82C37A-5 are connected to the DREQ and DACK signals of a channel of the initial 82C37A-5. This allows the DMA requests of the additional device to propagate through the priority network circuitry of the preceding device. The priority chain is preserved and the new device must wait for its turn to acknowledge requests. Since the cascade channel of the initial 82C37A-5 is used only for prioritizing the additional device, it does not output any address or control signals of its own. These could conflict with the outputs of the active channel in the added device. The 82C37A-5 will respond to DREQ and DACK but all other outputs except HRQ will be disabled. The ready input is ignored.

Figure 4 shows two additional devices cascaded into an initial device using two of the previous channels. This forms a two level DMA system. More 82C37A-5s could be added at the second level by using the remaining channels of the first level. Additional devices can also be added by cascading into the channels of the second level devices, forming a third level.

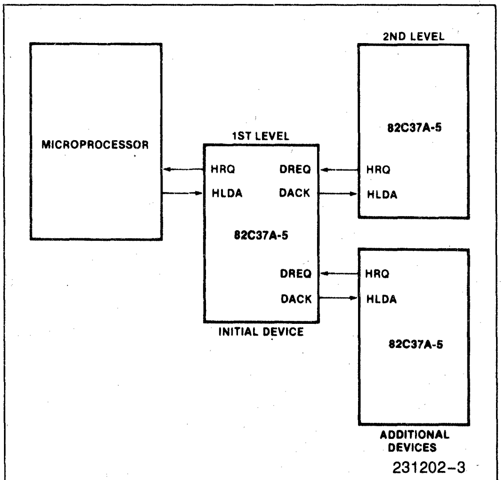


Figure 4. Cascaded 82C37A-5s

TRANSFER TYPES

Each of the three active transfer modes can perform three different types of transfers. These are Read, Write and Verify. Write transfers move data from and I/O device to the memory by activating $\overline{MEMW}$ and $\overline{IOR}$. Read transfers move data from memory to an I/O device by activating $\overline{MEMR}$ and $\overline{IOW}$. Verify transfers are pseudo transfers. The 82C37A-5 operates as in Read or Write transfers generating addresses, and responding to EOP, etc. However, the memory and I/O control lines all remain inactive. The ready input is ignored in verify mode.

Memory-to-Memory — To perform block moves of data from one memory address space to another with a minimum of program effort and time, the 82C37A-5 includes a memory-to-memory transfer feature. Programming a bit in the Command register selects channels 0 to 1 to operate as memory-to-memory transfer channels. The transfer is initiated by setting the software DREQ for channel 0. The

82C37A-5 requests a DMA service in the normal manner. After HLDA is true, the device, using four state transfers in Block Transfer mode, reads data from the memory. The channel 0 Current Address register is the source for the address used and is decremented or incremented in the normal manner. The data byte read from the memory is stored in the 82C37A-5 internal Temporary register. Channel 1 then performs a four-state transfer of the data from the Temporary register to memory using the address in its Current Address register and incrementing or decrementing it in the normal manner. The channel 1 current Word Count is decremented. When the word count of channel 1 goes to FFFFH, a TC is generated causing an $\overline{\text{EOP}}$ output terminating the service.

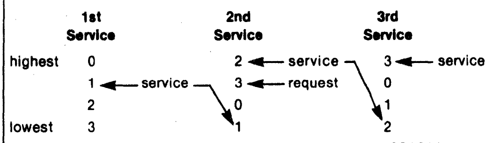
Channel 0 may be programmed to retain the same address for all transfers. This allows a single word to be written to a block of memory.

The 82C37A-5 will respond to external $\overline{\text{EOP}}$ signals during memory-to-memory transfers. Data comparators in block search schemes may use this input to terminate the service when a match is found. The timing of memory-to-memory transfers is found in Figure 12. Memory-to-memory operations can be detected as an active AEN with no DACK outputs.

Autoinitialize — By programming a bit in the Mode register, a channel may be set up as an Autoinitialize channel. During Autoinitialize initialization, the original values of the Current Address and Current Word Count registers are automatically restored from the Base Address and Base Word count registers of that channel following $\overline{\text{EOP}}$. The base registers are loaded simultaneously with the current registers by the microprocessor and remain unchanged throughout the DMA service. The mask bit is not altered when the channel is in Autoinitialize. Following Autoinitialize the channel is ready to perform another DMA service, without CPU intervention, as soon as a valid DREQ is detected. In order to Autoinitialize both channels in a memory-to-memory transfer, both word counts should be programmed identically. If interrupted externally, $\overline{\text{EOP}}$ pulses should be applied in both bus cycles.

Priority — The 82C37A-5 has two types of priority encoding available as software selectable options. The first is Fixed Priority which fixes the channels in priority order based upon the descending value of their number. The channel with the lowest priority is 3 followed by 2, 1 and the highest priority channel, 0. After the recognition of any one channel for service, the other channels are prevented from interfering with that service until it is completed.

The second scheme is Rotating Priority. The last channel to get service becomes the lowest priority channel with the others rotating accordingly.



231202-4

With Rotating Priority in a single chip DMA system, any device requesting service is guaranteed to be recognized after no more than three higher priority services have occurred. This prevents any one channel from monopolizing the system.

Compressed Timing — In order to achieve even greater throughput where system characteristics permit, the 82C37A-5 can compress the transfer time to two clock cycles. From Figure 11 it can be seen that state S3 is used to extend the access time of the read pulse. By removing state S3, the read pulse width is made equal to the write pulse width and a transfer consists only of state S2 to change the address and state S4 to perform the read/write. S1 states will still occur when A8–A15 need updating (see Address Generation). Timing for compressed transfers is found in Figure 14.

Address Generation — In order to reduce pin count, the 82C37A-5 multiplexes the eight higher order address bits on the data lines. State S1 is used to output the higher order address bits to an external latch from which they may be placed on the address bus. The falling edge of Address Strobe (ADSTB) is used to load these bits from the data lines to the latch. Address Enable (AEN) is used to enable the bits onto the address bus through a three-state enable. The lower order address bits are output by the 82C37A-5 directly. Lines A0–A7 should be connected to the address bus. Figure 11 shows the time relationships between CLK, AEN, ADSTB, DB0–DB7 and A0–A7.

During Block and Demand Transfer mode services, which include multiple transfers, the addresses generated will be sequential. For many transfers the data held in the external address latch will remain the same. This data need only change when a carry or borrow from A7 to A8 takes place in the normal sequence of addresses. To save time and speed transfers, the 82C37A-5 executes S1 states only when updating of A8–A15 in the latch is necessary. This means for long services, S1 states and Address Strobes may occur only once every 256 transfers, a savings of 255 clock cycles for each 256 transfers.

REGISTER DESCRIPTION

Current Address Register — Each channel has a 16-bit Current Address register. This register holds the value of the address used during DMA transfers. The address is automatically incremented or decremented after each transfer and the intermediate values of the address are stored in the Current Address register during the transfer. This register is written or read by the microprocessor in successive 8-bit bytes. It may also be reinitialized by an Autoinitialize back to its original value. Autoinitialize takes place only after an $\overline{EOP}$.

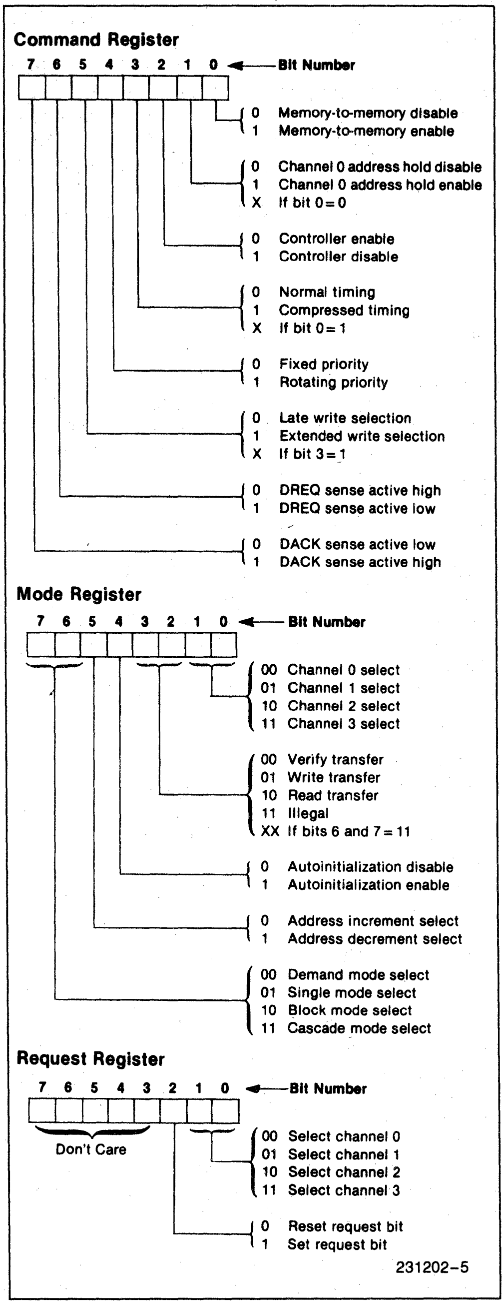
Current Word Register — Each channel has a 16-bit Current Word Count register. This register determines the number of transfers to be performed. The actual number of transfers will be one more than the number programmed in the Current Word Count register (i.e., programming a count of 100 will result in 101 transfers). The word count is decremented after each transfer. The intermediate value of the word count is stored in the register during the transfer. When the value in the register goes from zero to FFFFH, a TC will be generated. This register is loaded or read in successive 8-bit bytes by the microprocessor in the Program Condition. Following the end of a DMA service it may also be reinitialized by an Autoinitialization back to its original value. Autoinitialize can occur only when an $\overline{EOP}$ occurs. If it is not Autoinitialized, this register will have a count of FFFFH after TC.

Base Address and Base Word Count Registers — Each channel has a pair of Base Address and Base Word Count registers. These 16-bit registers store the original value of their associated current registers. During Autoinitialize these values are used to restore the current registers to their original values. The base registers are written simultaneously with their corresponding current register in 8-bit bytes in the Program Condition by the microprocessor. These registers cannot be read by the microprocessor.

Command Register — This 8-bit register controls the operation of the 82C37A-5. It is programmed by the microprocessor in the Program Condition and is cleared by Reset or a Master Clear instruction. The following table lists the function of the command bits. See Figure 6 for address coding.

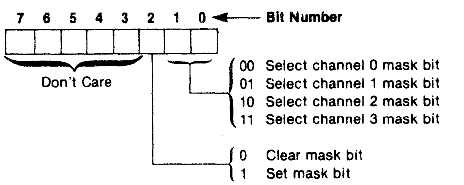
Mode Register — Each channel has a 6-bit Mode register associated with it. When the register is being written to by the microprocessor in the Program Condition, bits 0 and 1 determine which channel Mode register is to be written.

Request Register — The 82C37A-5 can respond to requests for DMA service which are initiated by software as well as by a DREQ. Each channel has a request bit associated with it in the 4-bit Request register. These are non-maskable and subject to prioritization by the Priority Encoder network. Each register bit is set or reset separately under software control or is cleared upon generation of a TC or external $\overline{EOP}$. The entire register is cleared by a Reset. To set or reset a bit, the software loads the proper form of the data word. See Figure 5 for register ad-



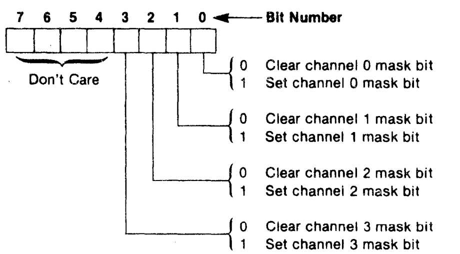
dress coding. In order to make a software request, the channel must be in Block Mode.

Mask Register — Each channel has associated with it a mask bit which can be set to disable the incoming DREQ. Each mask bit is set when its associated channel produces an $\overline{EOP}$ if the channel is not programmed for Autoinitialize. Each bit of the 4-bit Mask register may also be set or cleared separately under software control. The entire register is also set by a Reset. This disables all DMA requests until a clear Mask register instruction allows them to occur. The instruction to separately set or clear the mask bits is similar in form to that used with the Request register. See Figure 5 for instruction addressing.



231202-6

All four bits of the Mask register may also be written with a single command.

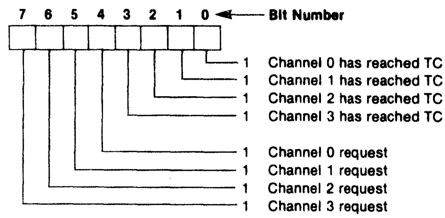


231202-7

Register	Operation	Signals						
		$\overline{CS}$	$\overline{IOR}$	$\overline{IOW}$	A3	A2	A1	A0
Command	Write	0	1	0	1	0	0	0
Mode	Write	0	1	0	1	0	1	1
Request	Write	0	1	0	1	0	0	1
Mask	Set/Reset	0	1	0	1	0	1	0
Mask	Write	0	1	0	1	1	1	1
Temporary	Read	0	0	1	1	1	0	1
Status	Read	0	0	1	1	0	0	0

Figure 5. Definition of Register Codes

Status Register — The Status register is available to be read out of the 82C37A-5 by the microprocessor. It contains information about the status of the devices at this point. This information includes which channels have reached a terminal count and which channels have pending DMA requests. Bits 0–3 are set every time a TC is reached by that channel or an external $\overline{EOP}$ is applied. These bits are cleared upon Reset and on each Status Read. Bits 4–7 are set whenever their corresponding channel is requesting service.



231202-8

Temporary Register — The Temporary register is used to hold data during memory-to-memory transfers. Following the completion of the transfers, the last word moved can be read by the microprocessor in the Program Condition. The Temporary register always contains the last byte transferred in the previous memory-to-memory operation, unless cleared by a Reset.

Software Commands — These are additional special software commands which can be executed in the Program Condition. They do not depend on any specific bit pattern on the data bus. The three software commands are:

Clear First/Last Flip-Flop: This command is executed prior to writing or reading new address or word count information to the 82C37A-5. This initializes the flip-flop to a known state so that subsequent accesses to register contents by the microprocessor will address upper and lower bytes in the correct sequence.

Master Clear: This software instruction has the same effect as the hardware Reset. The Command, Status, Request, Temporary, and Internal First/Last Flip-Flop registers are cleared and the Mask register is set. The 82C37A-5 will enter the Idle cycle.

Clear Mask Register: This command clears the mask bits of all four channels, enabling them to accept DMA requests.

Figure 6 lists the address codes for the software commands:

Signals						Operation
A3	A2	A1	A0	$\overline{IOR}$	$\overline{IOW}$	
1	0	0	0	0	1	Read Status Register
1	0	0	0	1	0	Write Command Register
1	0	0	1	0	1	Illegal
1	0	0	1	1	0	Write Request Register
1	0	1	0	0	1	Illegal
1	0	1	0	1	0	Write Single Mask Register Bit
1	0	1	1	0	1	Illegal
1	0	1	1	1	0	Write Mode Register
1	1	0	0	0	1	Illegal
1	1	0	0	1	0	Clear Byte Pointer Flip-Flop
1	1	0	1	0	1	Read Temporary Register
1	1	0	1	1	0	Master Clear
1	1	1	0	0	1	Illegal
1	1	1	0	1	0	Clear Mask Register
1	1	1	1	0	1	Illegal
1	1	1	1	1	0	Write All Mask Register Bits

Figure 6. Software Command Codes

PROGRAMMING

The 82C37A-5 will accept programming from the host processor any time that HLDA is inactive; this is true even if HRQ is active. The responsibility of the host is to assure that programming and HLDA are mutually exclusive. Note that a problem can occur if a DMA request occurs, on an unmasked channel while the 82C37A-5 is being programmed. For instance, the CPU may be starting to reprogram the two byte Address register of channel 1 when channel 1 receives a DMA request. If the 82C37A-5 is enabled (bit 2 in the command register is 0) and channel 1 is unmasked, a DMA service will occur after only one byte of the Address register has been reprogrammed. This can be avoided by disabling the controller (setting bit 2 in the command register) or masking the channel before programming any other registers. Once the programming is complete, the controller can be enabled/unmasked.

Channel	Register	Operation	Signals							Internal Flip-Flop	Data Bus DB0–DB7
			$\overline{CS}$	$\overline{IOR}$	$\overline{IOW}$	A3	A2	A1	A0		
0	Base and Current Address	Write	0	1	0	0	0	0	0	0	A0–A7
			0	1	0	0	0	0	0	1	A8–A15
	Current Address	Read	0	0	1	0	0	0	0	0	A0–A7
			0	0	1	0	0	0	0	1	A8–A15
	Base and Current Word Count	Write	0	1	0	0	0	0	1	0	W0–W7
			0	1	0	0	0	0	1	1	W8–W15
	Current Word Count	Read	0	0	1	0	0	0	1	0	W0–W7
			0	0	1	0	0	0	1	1	W8–W15
1	Base and Current Address	Write	0	1	0	0	0	1	0	0	A0–A7
			0	1	0	0	0	1	0	1	A8–A15
	Current Address	Read	0	0	1	0	0	1	0	0	A0–A7
			0	0	1	0	0	1	0	1	A8–A15
	Base and Current Word Count	Write	0	1	0	0	0	1	1	0	W0–W7
			0	1	0	0	0	1	1	1	W8–W15
	Current Word Count	Read	0	0	1	0	0	1	1	0	W0–W7
			0	0	1	0	0	1	1	1	W8–W15
2	Base and Current Address	Write	0	1	0	0	1	0	0	0	A0–A7
			0	1	0	0	1	0	0	1	A8–A15
	Current Address	Read	0	0	1	0	1	0	0	0	A0–A7
			0	0	1	0	1	0	0	1	A8–A15
	Base and Current Word Count	Write	0	1	0	0	1	0	1	0	W0–W7
			0	1	0	0	1	0	1	1	W8–W15
	Current Word Count	Read	0	0	1	0	1	0	1	0	W0–W7
			0	0	1	0	1	0	1	1	W8–W15
3	Base and Current Address	Write	0	1	0	0	1	1	0	0	A0–A7
			0	1	0	0	1	1	0	1	A8–A15
	Current Address	Read	0	0	1	0	1	1	0	0	A0–A7
			0	0	1	0	1	1	0	1	A8–A15
	Base and Current Word Count	Write	0	1	0	0	1	1	1	0	W0–W7
			0	1	0	0	1	1	1	1	W8–W15
	Current Word Count	Read	0	0	1	0	1	1	1	0	W0–W7
			0	0	1	0	1	1	1	1	W8–W15

Figure 7. Word Count and Address Register Command Codes

After power-up it is suggested that all internal locations, especially the Mode registers, be loaded with some valid value. This should be done even if some channels are unused.

APPLICATION INFORMATION

Figure 8 shows a convenient method for configuring a DMA system with the 82C37A-5 controller and an 8080A/8085AH microprocessor system. The multimode DMA controller issues a HRQ to the processor whenever there is at least one valid DMA request from a peripheral device. When the processor replies with a HLDA signal, the 82C37A-5 takes control of the address bus, the data bus and the control bus. The address for the first transfer operation comes out in two bytes — the least significant 8 bits on the eight address outputs and the most significant 8 bits on the data bus. The contents of the data bus are then latched into the 8-bit latch to complete the full 16 bits of the address bus. After the initial transfer takes place, the latch is updated only after a carry or borrow is generated in the least significant address byte. Four DMA channels are provided when one 82C37A-5 is used.

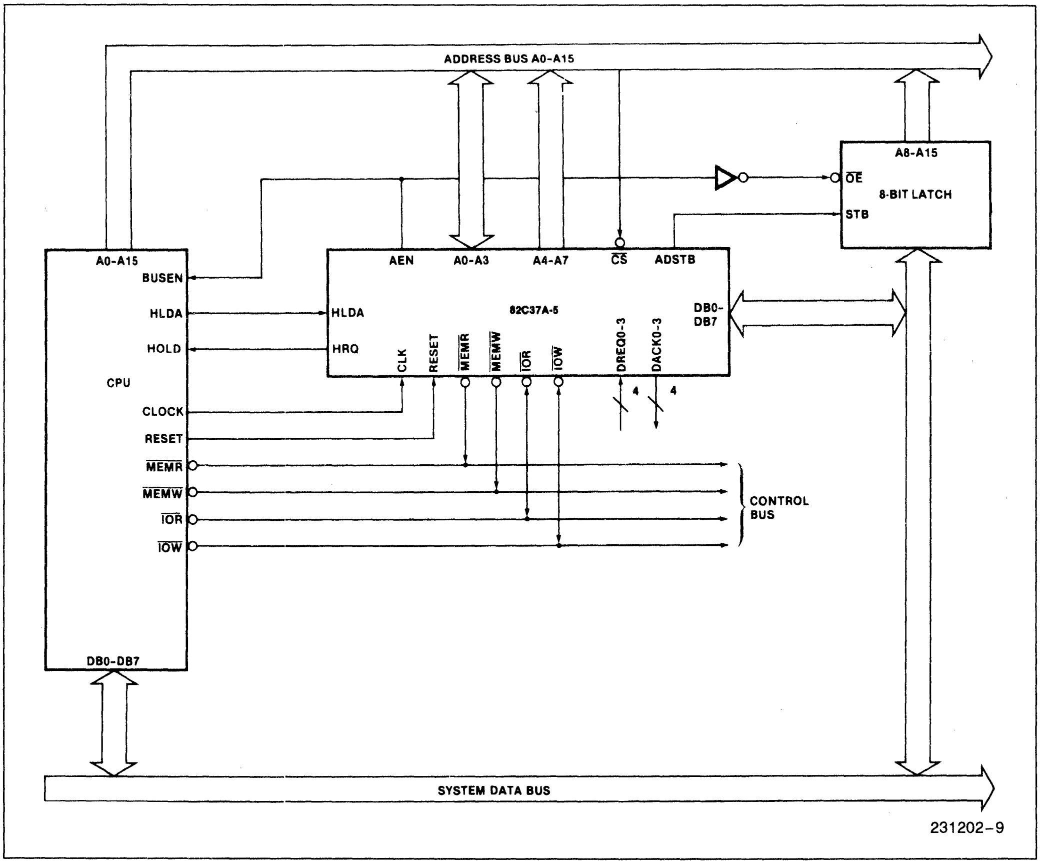


Figure 8. 82C37A-5 System Interface

intel®

8257/8257-5
PROGRAMMABLE DMA CONTROLLER

- **MCS-85® Compatible 8257-5**
- **4-Channel DMA Controller**
- **Priority DMA Request Logic**
- **Channel Inhibit Logic**
- **Terminal Count and Modulo 128 Outputs**
- **Single TTL Clock**
- **Single +5V Supply**
- **Auto Load Mode**
- **Available in EXPRESS**
  - Standard Temperature Range
- **Available in 40-Lead Cerdip and Plastic Package.**
  (See Packaging Spec, Order #231369)

The Intel® 8257 is a 4-channel direct memory access (DMA) controller. It is specifically designed to simplify the transfer of data at high speeds for the Intel® microcomputer systems. Its primary function is to generate, upon a peripheral request, a sequential memory address which will allow the peripheral to read or write data directly to or from memory. Acquisition of the system bus in accomplished via the CPU's hold function. The 8257 has priority logic that resolves the peripherals requests and issues a composite hold request to the CPU. It maintains the DMA cycle count for each channel and outputs a control signal to notify the peripheral that the programmed number of DMA cycles is complete. Other output control signals simplify sectored data transfers. The 8257 represents a significant savings in component count for DMA-based microcomputer systems and greatly simplifies the transfer of data at high speed between peripherals and memories.

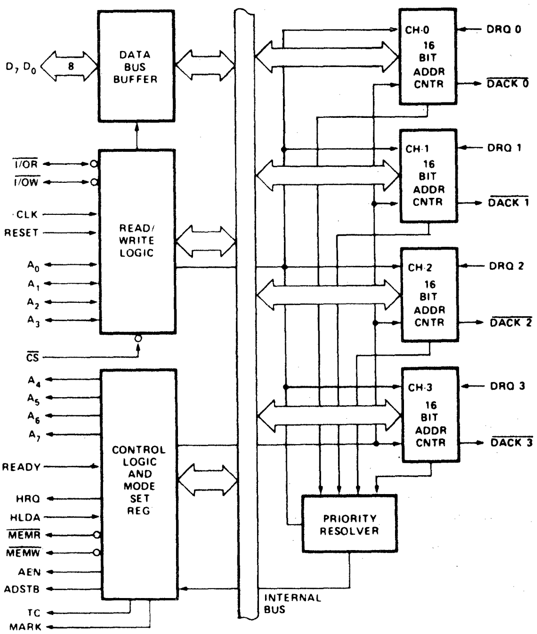


Figure 1. Block Diagram

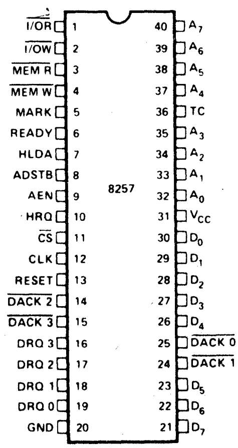


Figure 2. Pin Configuration

FUNCTIONAL DESCRIPTION

General

The 8257 is a programmable, Direct Memory Acess (DMA) device which, when coupled with a single 8-bit latch provides a complete four-channel DMA controller for use in Intel® microcomputer systems. After being initialized by software, the 8257 can transfer a block of data, containing up to 16,384 bytes, between memory and a peripheral device directly, without further intervention required of the CPU. Upon receiving a DMA transfer request from an enabled peripheral, the 8257:

1. Acquires control of the system bus.
2. Acknowledges that requesting peripheral which is connected to the highest priority channel.
3. Outputs the least significant eight bits of the memory address onto system address lines A_0-A_7, outputs the most significant eight bits of the memory address to the 8-bit latch via the data bus (the outputs of the latch should drive address lines A_8-A_{15}), and
4. Generates the appropriate memory and I/O read/write control signals that cause the peripheral to receive or deposit a data byte directly from or to the addressed location in memory.

The 8257 will retain control of the system bus and repeat the transfer sequence, as long as a peripheral maintains its DMA request. Thus, the 8257 can transfer a block of data to/from a high speed peripheral (e.g., a sector of data on a floppy disk) in a single "burst". When the specified number of data bytes have been transferred, the 8257 activates its Terminal Count (TC) output, informing the CPU that the operation is complete.

The 8257 offers three different modes of operation: (1) DMA read, which causes data to be transferred from memory to a peripheral; (2) DMA write, which causes data to be transferred from a peripheral to memory; and (3) DMA verify, which does not actually involve the transfer of data. When an 8257 channel is in the DMA verify mode, it will respond the same as described for transfer operations, except that no memory or I/O read/write control signals will be generated, thus preventing the transfer of data. The 8257, however, will gain control of the system bus and will acknowledge the peripheral's DMA request for each DMA cycle. The peripheral can use these acknowledge signals to enable an internal access of each byte of a data block in order to execute some verification procedure, such as the accumulation of a CRC (Cyclic Redundancy Code) checkword. For example, a block of DMA verify cycles might follow a block of DMA read cycles (memory to peripheral) to allow the peripheral to verify its newly acquired data.

Block Diagram Description

1. DMA Channels

The 8257 provides four separate DMA channels (labeled CH-0 to CH-3). Each channel includes two sixteen-bit registers: (1) a DMA address register, and (2) a terminal count register. Both registers must be initialized before a channel is enabled. The DMA address register is loaded with the address of the first memory location to be accessed. The value loaded into the low-order 14-bits of the terminal count register specifies the number of DMA cycles minus one before the Terminal Count (TC) output is activated. For instance, a terminal count of 0 would cause the TC output to be active in the first DMA cycle for that channel. In general, if N = the number of desired DMA cycles, load the value N-1 into the low-order 14-bits of the terminal count register. The most significant two bits of the terminal count register specify the type of DMA operation for that channel.

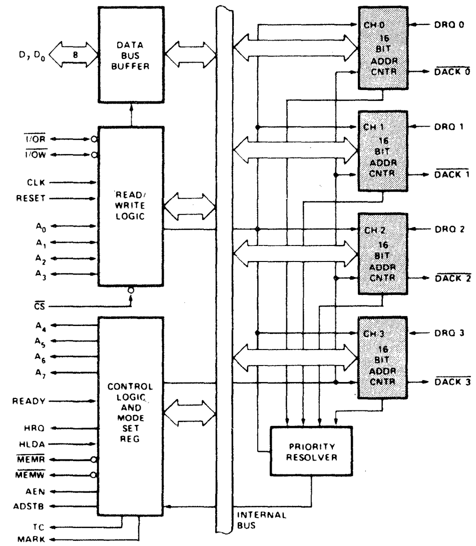


Figure 3. 8257 Block Diagram Showing DMA Channels

These two bits are not modified during a DMA cycle, but can be changed between DMA blocks.

Each channel accepts a DMA Request (DRQn) input and provides a DMA Acknowledge (DACKn) output.

(DRQ 0-DRQ 3)

DMA Request: These are individual asynchronous channel request inputs used by the peripherals to obtain a DMA cycle. If not in the rotating priority mode then DRQ 0 has the highest priority and DRQ 3 has the lowest. A request can be generated by raising the request line and holding it high until DMA acknowledge. For multiple DMA cycles (Burst Mode) the request line is held high until the DMA acknowledge of the last cycle arrives.

(DACK 0 - DACK 3)

DMA Acknowledge: An active low level on the acknowledge output informs the peripheral connected to that channel that it has been selected for a DMA cycle. The DACK output acts as a "chip select" for the peripheral device requesting service. This line goes active (low) and inactive (high) once for each byte transferred even if a burst of data is being transferred.

2. Data Bus Buffer

This three-state, bi-directional, eight bit buffer interfaces the 8257 to the system data bus.

(D_0-D_7)

Data Bus Lines: These are bi-directional three-state lines. When the 8257 is being programmed by the CPU, eight-bits of data for a DMA address register, a terminal count register or the Mode Set register are received on the data bus. When the CPU reads a DMA address register, a terminal count register or the Status register, the data is sent to the CPU over the data bus. During DMA cycles (when the 8257 is the bus master), the 8257 will output the most significant eight-bits of the memory address (from one of the DMA address registers) to the 8212 latch via the data bus. These address bits will be transferred at the beginning of the DMA cycle; the bus will then be released to handle the memory data transfer during the balance of the DMA cycle.

BIT 15	BIT 14	TYPE OF DMA OPERATION
0	0	Verify DMA Cycle
0	1	Write DMA Cycle
1	0	Read DMA Cycle
1	1	(Illegal)

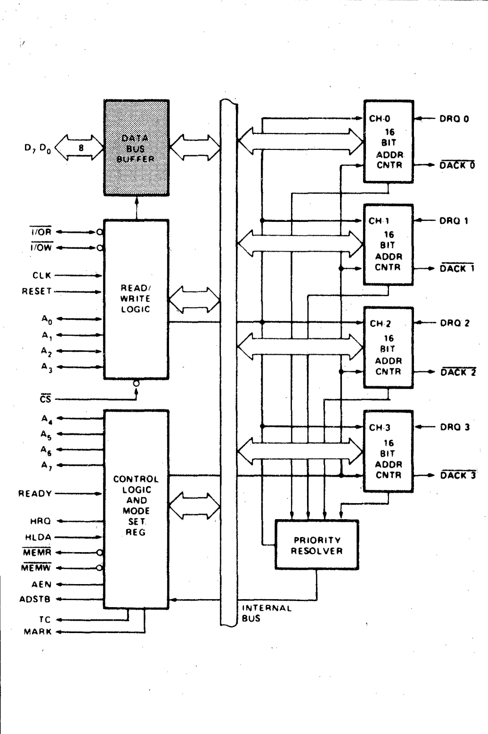


Figure 4. 8257 Block Diagram Showing Data Bus Buffer

3. Read/Write Logic

When the CPU is programming or reading one of the 8257's registers (i.e., when the 8257 is a "slave" device on the system bus), the Read/Write Logic accepts the I/O Read ($\overline{I/OR}$) or I/O Write ($\overline{I/OW}$) signal, decodes the least significant four address bits, (A_0-A_3), and either writes the contents of the data bus into the addressed register (if $\overline{I/OW}$ is true) or places the contents of the addressed register onto the data bus (if $\overline{I/OR}$ is true).

During DMA cycles (i.e., when the 8257 is the bus "master"), the Read/Write Logic generates the I/O read and memory write (DMA write cycle) or I/O Write and memory read (DMA read cycle) signals which control the data link with the peripheral that has been granted the DMA cycle.

Note that during DMA transfers Non-DMA I/O devices should be de-selected (disabled) using "AEN" signal to inhibit I/O device decoding of the memory address as an erroneous device address.

($\overline{I/OR}$)

I/O Read: An active-low, bi-directional three-state line. In the "slave" mode, it is an input which allows the 8-bit status register or the upper/lower byte of a 16-bit DMA address register or terminal count register to be read. In the "master" mode, $\overline{I/OR}$ is a control output which is used to access data from a peripheral during the DMA write cycle.

($\overline{I/OW}$)

I/O Write: An active-low, bi-directional three-state line. In the "slave" mode, it is an input which allows the contents of the data bus to be loaded into the 8-bit mode set register or the upper/lower byte of a 16-bit DMA address register or terminal count register. In the "master" mode, $\overline{I/OW}$ is a control output which allows data to be output to a peripheral during a DMA read cycle.

(CLK)

Clock Input: Generally from an Intel® 8224 Clock Generator device. (ϕ2 TTL) or Intel ® 8085AH CLK output.

(RESET)

Reset: An asynchronous input (generally from an 8224 or 8085 device) which disables all DMA channels by clearing the mode register and 3-states all control lines.

(A_0-A_3)

Address Lines: These least significant four address lines are bi-directional. In the "slave" mode they are inputs which select one of the registers to be read or programmed. In the "master" mode, they are outputs which constitute the least significant four bits of the 16-bit memory address generated by the 8257.

($\overline{CS}$)

Chip Select: An active-low input which enables the I/O Read or I/O Write input when the 8257 is being read or programmed in the "slave" mode. In the "master" mode, $\overline{CS}$ is automatically disabled to prevent the chip from selecting itself while performing the DMA function.

4. Control Logic

This block controls the sequence of operations during all DMA cycles by generating the appropriate control signals and the 16-bit address that specifies the memory location to be accessed.

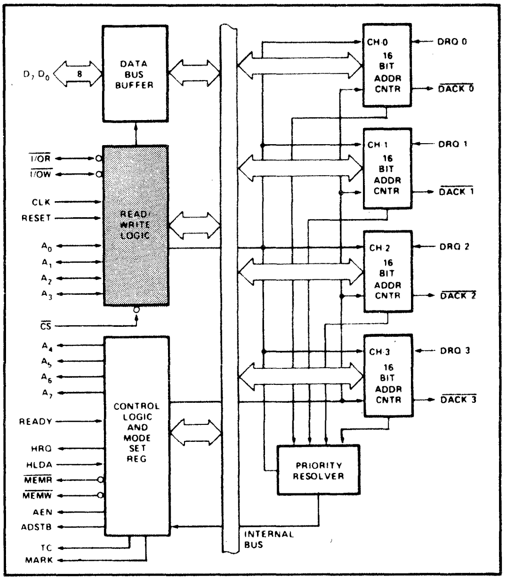


Figure 5. 8257 Block Diagram Showing Read/Write Logic Function

(A_4-A_7)

Address Lines: These four address lines are three-state outputs which constitute bits 4 through 7 of the 16-bit memory address generated by the 8257 during all DMA cycles.

(READY)

Ready: This asynchronous input is used to elongate the memory read and write cycles in the 8257 with wait states if the selected memory requires longer cycles. READY must conform to specified setup and hold times.

(HRQ)

Hold Request: This output requests control of the system bus. In systems with only one 8257, HRQ will normally be applied to the HOLD input on the CPU. HRQ must conform to specified setup and hold times.

(HLDA)

Hold Acknowledge: This input from the CPU indicates that the 8257 has acquired control of the system bus. HLDA must remain stable during the specified set-up time.

($\overline{MEMR}$)

Memory Read: This active-low three-state output is used to read data from the addressed memory location during DMA Read cycles.

($\overline{MEMW}$)

Memory Write: This active-low three-state output is used to write data into the addressed memory location during DMA Write cycles.

(ADSTB)

Address Strobe: This output strobes the most significant byte of the memory address into the latch device from the data bus.

(AEN)

Address Enable: This output is used to disable (float) the System Data Bus and the System Control Bus. It may also be used to disable (float) the System Address Bus by use of an enable on the Address Bus drivers in systems to inhibit non-DMA devices from responding during DMA cycles. It may be further used to isolate the 8257 data bus from the System Data Bus to facilitate the transfer of the 8 most significant DMA address bits over the 8257 data I/O pins without subjecting the System Data Bus to any timing constraints for the transfer. When the 8257 is used in an I/O device structure (as opposed to memory mapped), this AEN output should be used to disable the selection of an I/O device when the DMA address is on the address bus. The I/O device selection should be determined by the DMA acknowledge outputs for the 4 channels.

(TC)

Terminal Count: This output notifies the currently selected peripheral that the present DMA cycle should be the last cycle for this data block. If the TC STOP bit in the Mode Set register is set, the selected channel will be automatically disabled at the end of that DMA cycle. TC is activated when the 14-bit value in the selected channel's terminal count register equals zero. Recall that the low-order 14-bits of the terminal count register should be loaded with the values (n-1), where n = the desired number of the DMA cycles.

(MARK)

Modulo 128 Mark: This output notifies the selected peripheral that the current DMA cycle is the 128th cycle since the previous MARK output. MARK always occurs at 128 (and all multiples of 128) cycles from the end of the data block. Only if the total number of DMA cycles (n) is evenly divisable by 128 (and the terminal count register was loaded with n-1), will MARK occur at 128 (and each succeeding multiple of 128) cycles from the beginning of the data block.

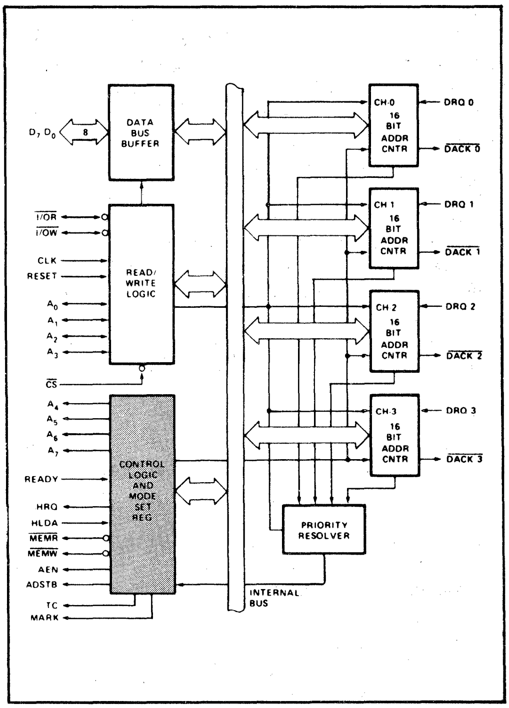
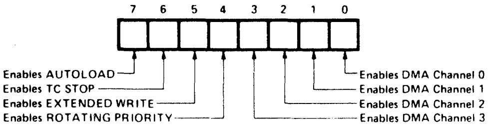


Figure 6. 8257 Block Diagram Showing Control Logic and Mode Set Register

5. Mode Set Register

When set, the various bits in the Mode Set register enable each of the four DMA channels, and allow four different options for the 8257:

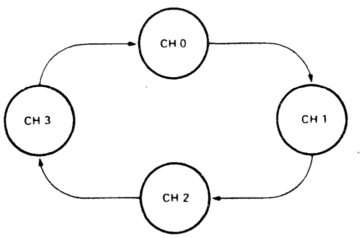


The Mode Set register is normally programmed by the CPU after the DMA address register(s) and terminal count register(s) are initialized. The Mode Set Register is cleared by the RESET input, thus disabling all options, inhibiting all channels, and preventing bus conflicts on power-up. A channel should not be left enabled unless its DMA address and terminal count registers contain valid values; otherwise, an inadvertent DMA request (DRQn) from a peripheral could initiate a DMA cycle that would destroy memory data.

The various options which can be enabled by bits in the Mode Set register are explained below:

Rotating Priority Bit 4

In the Rotating Priority Mode, the priority of the channels has a circular sequence. After each DMA cycle, the priority of each channel changes. The channel which had just been serviced will have the lowest priority.



If the ROTATING PRIORITY bit is not set (set to a zero), each DMA channel has a fixed priority. In the fixed priority mode, Channel 0 has the highest priority and Channel 3 has the lowest priority. If the ROTATING PRIORITY bit is set to a one, the priority of each channel changes after each DMA cycle (not each DMA request). Each channel moves up to the next highest priority assignment, while the channel which has just been serviced moves to the lowest priority assignment:

	CHANNEL→ JUST SERVICED	CH-0	CH-1	CH-2	CH-3
Priority → Assignments	Highest	CH-1	CH-2	CH-3	CH-0
	↕	CH-2	CH-3	CH-0	CH-1
		CH-3	CH-0	CH-1	CH-2
	Lowest	CH-0	CH-1	CH-2	CH-3

Note that rotating priority will prevent any one channel from monopolizing the DMA mode; consecutive DMA cycles will service different channels if more than one channel is enabled and requesting service. There is no overhead penalty associated with this mode of operation. All DMA operations began with Channel 0 initially assigned to the highest priority for the first DMA cycle.

Extended Write Bit 5

If the EXTENDED WRITE bit is set, the duration of both the $\overline{MEMW}$ and $\overline{I/OW}$ signals is extended by activating them earlier in the DMA cycle. Data transfers within microcomputer systems proceed asynchronously to allow use of various types of memory and I/O devices with different access times. If a device cannot be accessed within a specific amount of time it returns a "not ready" indication to the 8257 that causes the 8257 to insert one or more wait states in its internal sequencing. Some devices are fast enough to be accessed without the use of wait states, but if they generate their READY response with the leading edge of the $\overline{I/OW}$ or $\overline{MEMW}$ signal (which generally occurs late in the transfer sequence), they would normally cause the 8257 to enter a wait state because it does not receive READY in time. For systems with these types of devices, the Extended Write option provides alternative timing for the I/O and memory write signals which allows the devices to return an early READY and prevents the unnecessary occurrence of wait states in the 8257, thus increasing system throughput.

TC Stop Bit 6

If the TC STOP bit is set, a channel is disabled (i.e., its enable bit is reset) after the Terminal Count (TC) output goes true, thus automatically preventing further DMA operation on that channel. The enable bit for that channel must be re-programmed to continue or begin another DMA operation. If the TC STOP bit is not set, the occurrence of the TC output has no effect on the channel enable bits. In this case, it is generally the responsibility of the peripheral to cease DMA requests in order to terminate a DMA operation.

Auto Load Bit 7

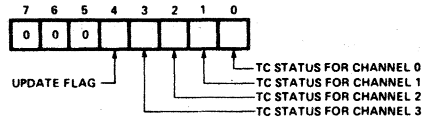
The Auto Load mode permits Channel 2 to be used for repeat block or block chaining operations, without immediate software intervention between blocks. Channel 2 registers are initialized as usual for the first data block; Channel 3 registers, however, are used to store the block re-initialization parameters (DMA starting address, terminal count and DMA transfer mode). After the first block of DMA cycles is executed by Channel 2 (i.e., after the TC output goes true), the parameters stored in the Channel 3 registers are transferred to Channel 2 during an "update" cycle. Note that the TC STOP feature, described above, has no effect on Channel 2 when the Auto Load bit is set.

If the Auto Load bit is set, the initial parameters for Channel 2 are automatically duplicated in the Channel 3 registers when Channel 2 is programmed. This permits repeat block operations to be set up with the programming of a single channel. Repeat block operations can be used in applications such as CRT refreshing. Channels 2 and 3 can still be loaded with separate values if Channel 2 is loaded before loading Channel 3. Note that in the Auto Load mode, Channel 3 is still available to the user if the Channel 3 enable bit is set, but use of this channel will change the values to be auto loaded into Channel 2 at update time. All that is necessary to use the Auto Load feature for chaining operations is to reload Channel 3 registers at the conclusion of each update cycle with the new parameters for the next data block transfer.

Each time that the 8257 enters an update cycle, the update flag in the status register is set and parameters in Channel 3 are transferred to Channel 2, non-destructively for Channel 3. The actual re-initialization of Channel 2 occurs at the beginning of the next channel 2 DMA cycle after the TC cycle. This will be the first DMA cycle of the new data block for Channel 2. The update flag is cleared at the conclusion of this DMA cycle. For chaining operations, the update flag in the status register can be monitored by the CPU to determine when the re-initialization process has been completed so that the next block parameters can be safely loaded into Channel 3.

6. Status Register

The eight-bit status register indicates which channels have reached a terminal count condition and includes the update flag described previously.



The TC status bits are set when the Terminal Count (TC) output is activated for that channel. These bits remain set until the status register is read or the 8257 is reset. The UPDATE FLAG, however, is not affected by a status register read operation. The UPDATE FLAG can be cleared by resetting the 8257, by changing to the non-auto load mode (i.e., by resetting the AUTO LOAD bit in the Mode Set register) or it can be left to clear itself at the completion of the update cycle. The purpose of the UPDATE FLAG is to prevent the CPU from inadvertently skipping a data block by overwriting a starting address or terminal count in the Channel 3 registers before those parameters are properly auto-loaded into Channel 2.

The user is cautioned against reading the TC status register and using this information to reenable channels that have not completed operation. Unless the DMA channels are inhibited a channel could reach terminal count (TC) between the status read and the mode write. DMA can be inhibited by a hardware gate on the HRQ line or by disabling channels with a mode word before reading the TC status.

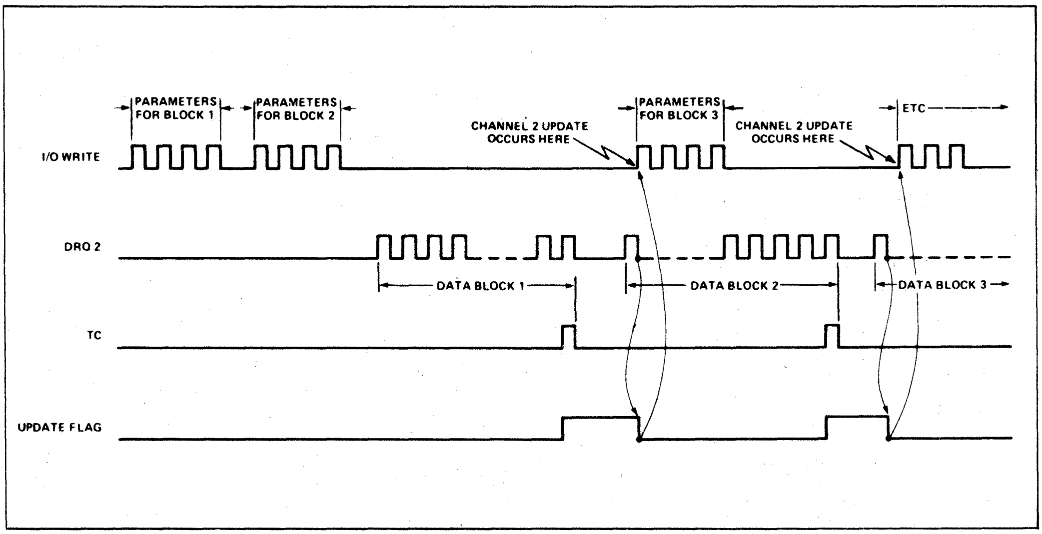


Figure 7. Autoload Timing

OPERATIONAL SUMMARY

Programming and Reading the 8257 Registers

There are four pairs of "channel registers": each pair consisting of a 16-bit DMA address register and a 16-bit terminal count register (one pair for each channel). The 8257 also includes two "general registers": one 8-bit Mode Set register and one 8-bit Status register. The registers are loaded or read when the CPU executes a write or read instruction that addresses the 8257 device and the appropriate register within the 8257. The 8228 generates the appropriate read or write control signal (generally I/OR or I/OW while the CPU places a 16-bit address on the system address bus, and either outputs the data to be written onto the system data bus or accepts the data being read from the data bus. All or some of the most significant 12 address bits A_4-A_{15} (depending on the systems memory, I/O configuration) are usually decoded to produce the chip select ($\overline{CS}$) input to the 8257. An I/O Write input (or Memory Write in memory mapped I/O configurations, described below) specifies that the addressed register is to be programmed, while an I/O Read input (or Memory Read) specifies that the addressed register is to be read. Address bit 3 specifies whether a "channel register" (A_3 = 0) or the Mode Set (program only)/Status (read only) register (A_3 = 1) is to be accessed.

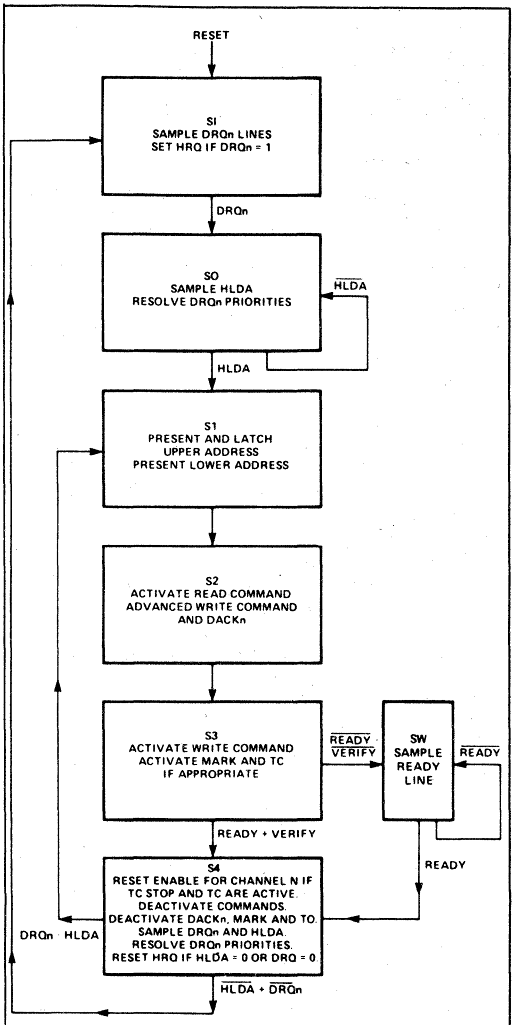
The least significant three address bits, A_0-A_2, indicate the specific register to be accessed. When accessing the Mode Set or Status register, A_0-A_2 are all zero. When accessing a channel register bit A_0 differentiates between the DMA address register (A_0 = 0) and the terminal count register (A_0 = 1), while bits A_1 and A_2 specify one of the four channels. Because the "channel registers" are 16-bits, two program instruction cycles are required to load or read an entire register. The 8257 contains a first/last (F/L) flip flop which toggles at the completion of each channel program or read operation. The F/L flip flop determines whether the upper or lower byte of the register is to be accessed. The F/L flip flop is reset by the RESET input and whenever the Mode Set register is loaded. To maintain proper synchronization when accessing the "channel registers" all channel command instruction operations should occur in pairs, with the lower byte of a register always being accessed first. Do not allow $\overline{CS}$ to clock while either $\overline{I/OR}$ or $\overline{I/OW}$ is active, as this will cause an erroneous F/L flip flop state. In systems utilizing an interrupt structure, interrupts should be disabled prior to any paired programming operations to prevent an interrupt from splitting them. The result of such a split would leave the F/L F/F in the wrong state. This problem is particularly obvious when other DMA channels are programmed by an interrupt structure.

CONTROL INPUT	$\overline{CS}$	$\overline{I/OW}$	$\overline{I/OR}$	A_3
Program Half of a Channel Register	0	0	1	0
Read Half of a Channel Register	0	1	0	0
Program Mode Set Register	0	0	1	1
Read Status Register	0	1	0	1

8257 Register Selection

REGISTER	BYTE	ADDRESS INPUTS				F/L	*BI-DIRECTIONAL DATA BUS							
		A_3	A_2	A_1	A_0		D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CH-0 DMA Address	LSB	0	0	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	0	0	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-0 Terminal Count	LSB	0	0	0	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	0	0	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
CH-1 DMA Address	LSB	0	0	1	0	0	Same as Channel 0							
	MSB	0	0	1	0	1								
CH-1 Terminal Count	LSB	0	0	1	1	0								
	MSB	0	0	1	1	1								
CH-2 DMA Address	LSB	0	1	0	0	0	Same as Channel 0							
	MSB	0	1	0	0	1								
CH-2 Terminal Count	LSB	0	1	0	1	0								
	MSB	0	1	0	1	1								
CH-3 DMA Address	LSB	0	1	1	0	0	Same as Channel 0							
	MSB	0	1	1	0	1								
CH-3 Terminal Count	LSB	0	1	1	1	0								
	MSB	0	1	1	1	1								
MODE SET (Program only)	—	1	0	0	0	0	AL	TCS	EW	RP	EN3	EN2	EN1	EN0
STATUS (Read only)	—	1	0	0	0	0	0	0	0	UP	TC3	TC2	TC1	TC0

*A_0-A_{15}: DMA Starting Address, C_0-C_{13}: Terminal Count value (N-1), Rd and Wr: DMA Verify (00), Write (01) or Read (10) cycle selection, AL: Auto Load, TCS: TC STOP, EW: EXTENDED WRITE, RP: ROTATING PRIORITY, EN3-EN0: CHANNEL ENABLE MASK, UP: UPDATE FLAG, TC3-TC0: TERMINAL COUNT STATUS BITS.



1 DRQn REFERS TO ANY DRQ LINE ON AN ENABLED DMA CHANNEL.

Figure 8. DMA Operation State Diagram

DMA OPERATION

Single Byte Transfers

A single byte transfer is initiated by the I/O device raising the DRQ line of one channel of the 8257. If the channel is enabled, the 8257 will output a HRQ to the CPU. The 8257 now waits until a HLDA is received insuring that the system bus is free for its use. Once HLDA is received the $\overline{\text{DACK}}$ line for the requesting channel is activated (LOW). The $\overline{\text{DACK}}$ line acts as a chip select for the requesting I/O device. The 8257 then generates the read and write commands and byte transfer occurs between the selected I/O device and memory. After the transfer is complete, the $\overline{\text{DACK}}$ line is set HIGH and the HRQ line is set LOW to indicate to the CPU that the bus is now free for use. DRQ must remain HIGH until $\overline{\text{DACK}}$ is issued to be recognized and must go LOW before S4 of the transfer sequence to prevent another transfer from occuring. (See timing diagram.)

Consecutive Transfers

If more than one channel requests service simultaneously, the transfer will occur in the same way a burst does. No overhead is incurred by switching from one channel to another. In each S4 the DRQ lines are sampled and the highest priority request is recognized during the next transfer. A burst mode transfer in a lower priority channel will be overridden by a higher priority request. Once the high priority transfer has completed control will return to the lower priority channel if its DRQ is still active. No extra cycles are needed to execute this sequence and the HRQ line remains active until all DRQ lines go LOW.

Control Override

The continuous DMA transfer mode described above can be interrupted by an external device by lowering the HLDA line. After each DMA transfer the 8257 samples the HLDA line to insure that it is still active. If it is not active, the 8257 completes the current transfer, releases the HRQ line (LOW) and returns to the idle state. If DRQ lines are still active the 8257 will raise the HRQ line in the third cycle and proceed normally. (See timing diagram.)

Not Ready

The 8257 has a Ready input similar to the 8080A and the 8085AH. The Ready line is a sampled in State 3. If Ready is LOW the 8257 enters a wait state. Ready is sampled during every wait state. When Ready returns HIGH the 8257 proceeds to State 4 to complete the transfer. Ready is used to interface memory of I/O devices that cannot meet the bus set up times required by the 8257.

Speed

The 8257 uses four clock cycles to transfer a byte of data. No cycles are lost in the master to master transfer maximizing bus efficiency. A 2MHz clock input will allow the 8257 to transfer at a rate of 500K bytes/second.

Memory Mapped I/O Configurations

The 8257 can be connected to the system bus as a memory device instead of as an I/O device for memory mapped I/O configurations by connecting the system memory control lines to the 8257's I/O control lines and the system I/O control lines to the 8257's memory control lines.

This configuration permits use of the 8080's considerably larger repertoire of memory instructions when reading or loading the 8257's registers. Note that with this connection, the programming of the Read (bit 15) and Write (bit 14) bits in the terminal count register will have a different meaning:

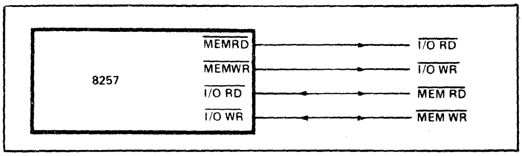


Figure 9. System Interface for Memory Mapped I/O

BIT 15 READ	BIT 14 WRITE	
0	0	DMA Verify Cycle
0	1	DMA Read Cycle
1	0	DMA Write Cycle
1	1	Illegal

Figure 10. TC Register for Memory Mapped I/O Only

SYSTEM APPLICATION EXAMPLES

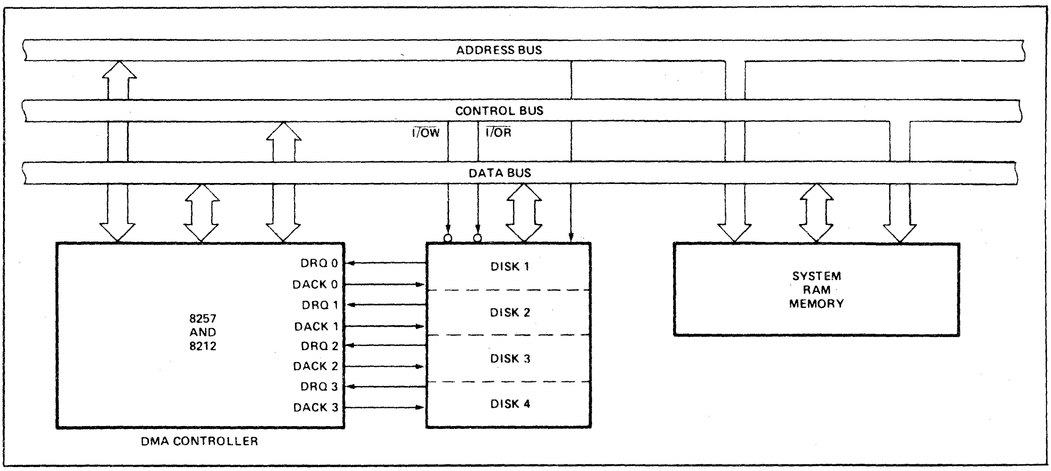


Figure 11. Floppy Disk Controller (4 Drives)

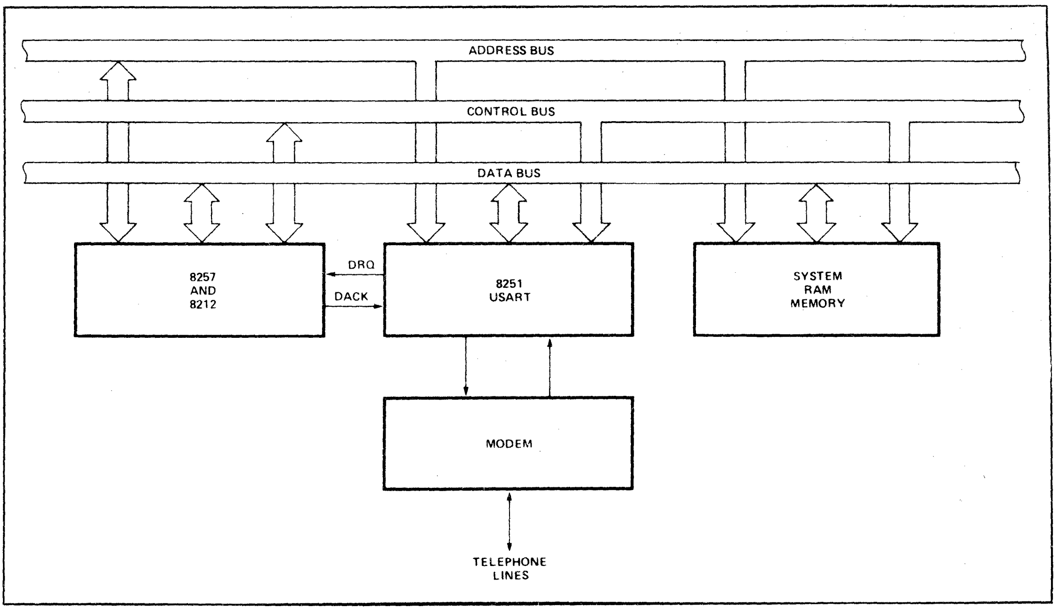
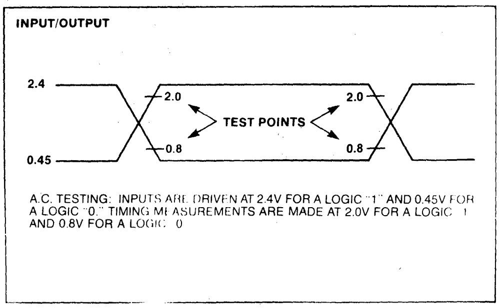
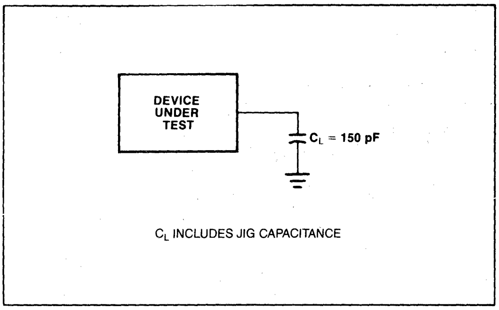


Figure 12. High-Speed Communication Controller

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING LOAD CIRCUIT



Tracking Parameters

Signals labeled as Tracking Parameters (footnotes 1 and 5-7 under A.C. Specifications) are signals that follow similar paths through the silicon die. The propagation speed of these signals varies in the manufacturing process but the relationship between all these parameters is constant. The variation is less than or equal to 50 ns.

Suppose the following timing equation is being evaluated,

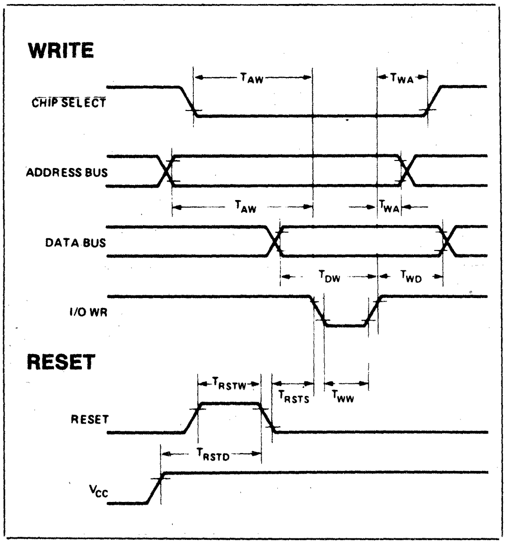
$$T_{A(MIN)} + T_{B(MAX)} \leq 150 \text{ ns}$$

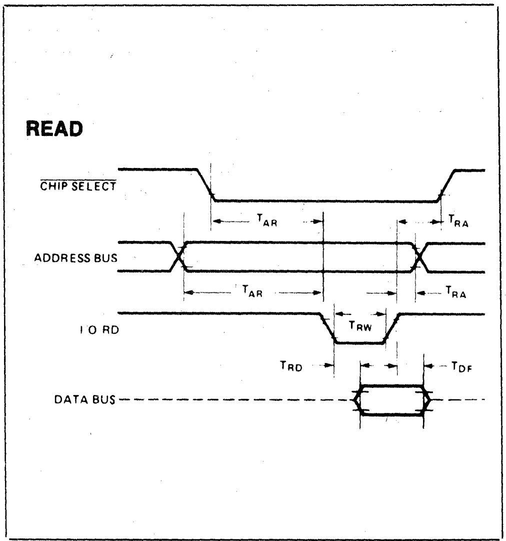
and only minimum specifications exist for T_A and T_B. If $T_{A(MIN)}$ is used, and if T_A and T_B are tracking parameters, $T_{B(MAX)}$ can be taken as $T_{B(MIN)}$ + 50 ns.

$$T_{A(MIN)} + (T_{B(MIN)}^{*} + 50 \text{ ns}) \leq 150 \text{ ns}$$

*if T_A and T_B are tracking parameters

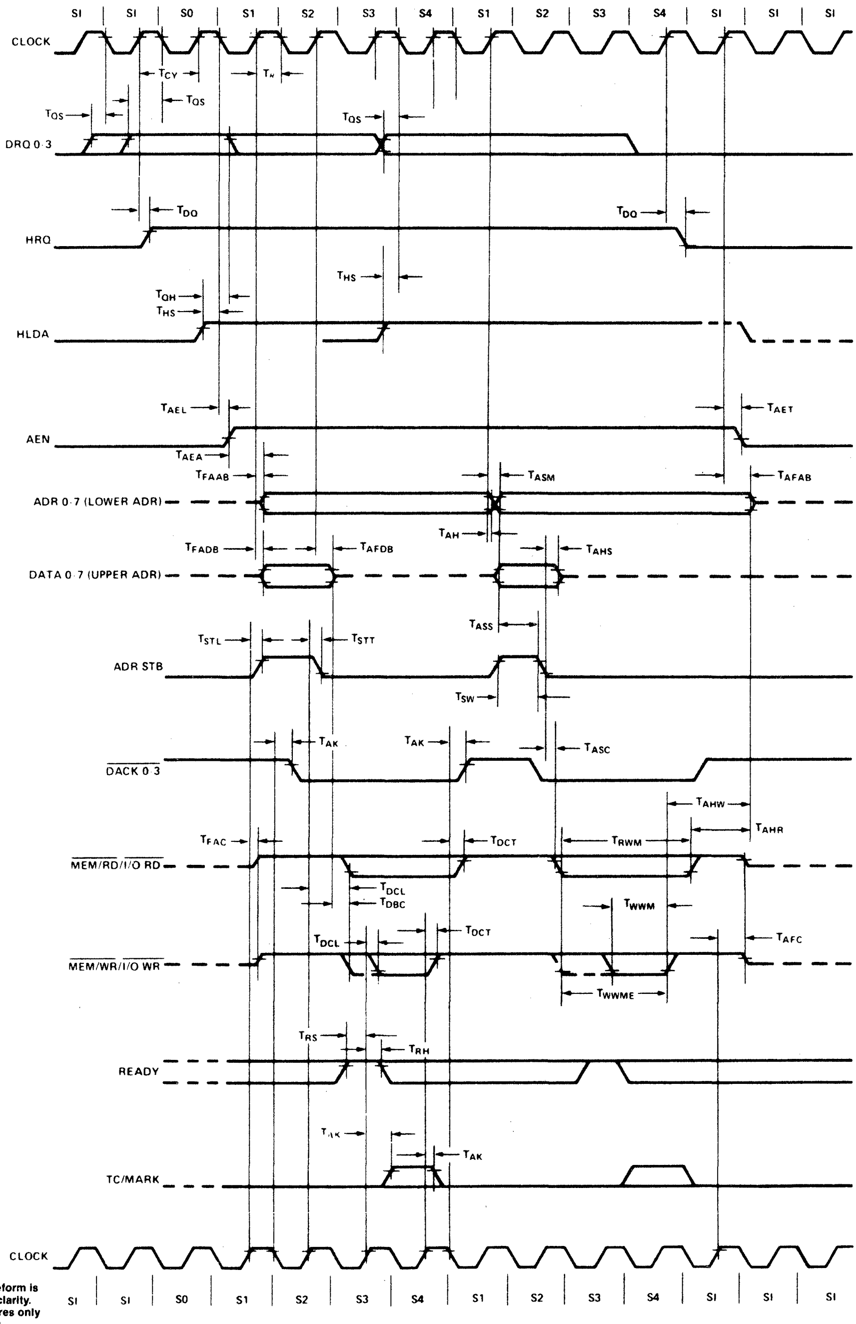
WAVEFORMS—PERIPHERAL MODE





WAVEFORMS—DMA

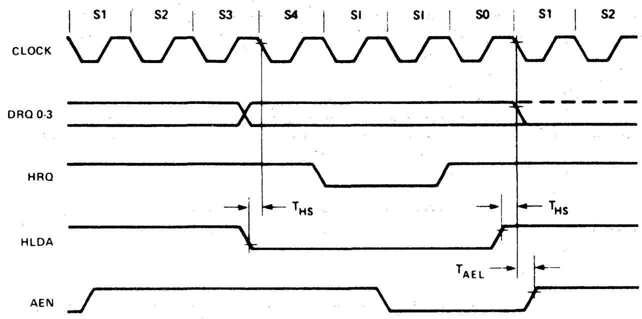
CONSECUTIVE CYCLES AND BURST MODE SEQUENCE



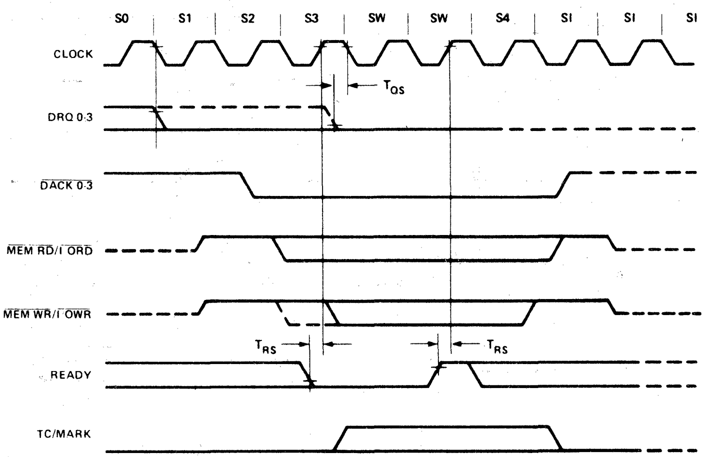
NOTE: The clock waveform is duplicated for clarity. The 8257 requires only one clock input.

WAVEFORMS (Continued)

CONTROL OVERRIDE SEQUENCE



NOT READY SEQUENCE



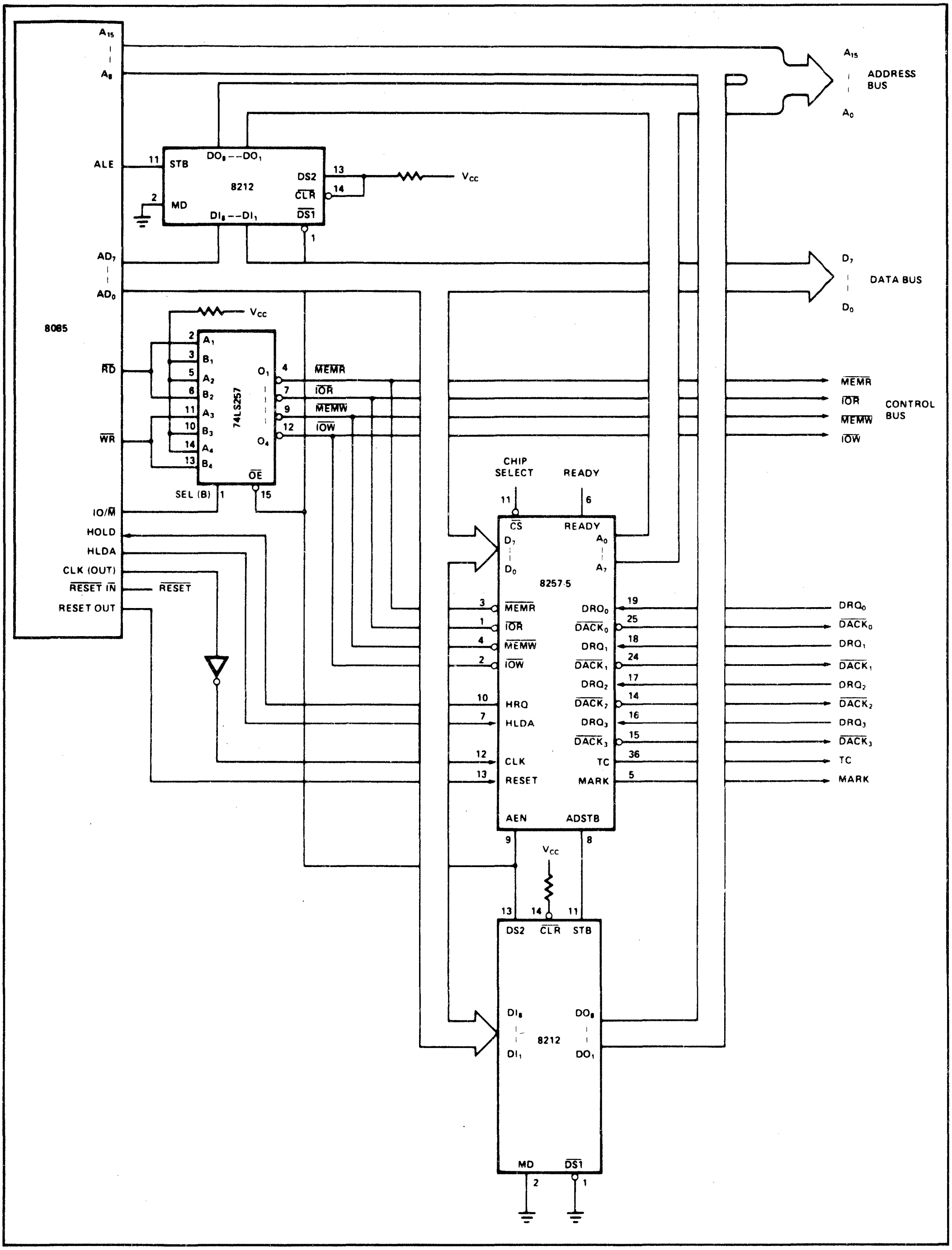


Figure 13. Detailed System Interface Schematic

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .... 0°C to 70°C
Storage Temperature .... −65°C to +150°C
Voltage on Any Pin
With Respect to Ground .... −0.5V to +7V
Power Dissipation .... 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

(8257: T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V)
(8257-5: T_A = 0°C to 70°C, V_{CC} = 5.0V ±10%, GND = 0V)

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	-0.5	0.8	Volts	
V_{IH}	Input High Voltage	2.0	V_{CC}+.5	Volts	
V_{OL}	Output Low Voltage		0.45	Volts	I_{OL} = 1.6 mA
V_{OH}	Output High Voltage	2.4	V_{CC}	Volts	I_{OH}=-150μA for AB, DB and AEN I_{OH}=-80μA for others
V_{HH}	HRQ Output High Voltage	3.3	V_{CC}	Volts	I_{OH} = -80μA
I_{CC}	V_{CC} Current Drain		120	mA	
I_{IL}	Input Leakage		±10	μA	0V ≤ V_{IN} ≤ V_{CC}
I_{OFL}	Output Leakage During Float		±10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}

CAPACITANCE

(T_A = 25°C; V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	fc = 1MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to GND

A.C. CHARACTERISTICS—PERIPHERAL (SLAVE) MODE

(8257: T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V)
(8257-5: T_A = 0°C to 70°C, V_{CC} = 5.0V ±10%, GND = 0V)

8080 Bus Parameters

READ CYCLE

Symbol	Parameter	8257 Min.	8257 Max.	8257-5 Min.	8257-5 Max.	Unit	Test Conditions
T_{AR}	Adr or $\overline{CS}$↓ Setup to $\overline{RD}$↓	0		0		ns	
T_{RA}	Adr or $\overline{CS}$↑ Hold from $\overline{RD}$↑	0		0		ns	
T_{RD}	Data Access from $\overline{RD}$↓	0	300	0	220	ns	
T_{DF}	DB→Float Delay from $\overline{RD}$↑	20	150	20	120	ns	
T_{RR}	$\overline{RD}$ Width	250		250		ns	

WRITE CYCLE

Symbol	Parameter	8257 Min.	8257 Max.	8257-5 Min.	8257-5 Max.	Unit	Test Conditions
T_{AW}	Adr Setup to $\overline{WR}$↓	20		20		ns	
T_{WA}	Adr Hold from $\overline{WR}$↑	0		0		ns	
T_{DW}	Data Setup to $\overline{WR}$↑	200		200		ns	
T_{WD}	Data Hold from $\overline{WR}$↑	10		10		ns	
T_{WW}	$\overline{WR}$ Width	200		200		ns	

OTHER TIMING

Symbol	Parameter	8257 Min.	8257 Max.	8257-5 Min.	8257-5 Max.	Unit	Test Conditions
T_{RSTW}	Reset Pulse Width	300		300		ns	
T_{RSTD}	Power Supply↑ (V_{CC}) Setup to Reset↓	500		500		μs	
T_r	Signal Rise Time		20		20	ns	
T_f	Signal Fall Time		20		20	ns	
T_{RSTS}	Reset to First $\overline{I/OWR}$	2		2		t_{CY}	

A.C. CHARACTERISTICS—DMA (MASTER) MODE

(8257: T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V)
(8257-5: T_A = 0°C to 70°C, V_{CC} = 5.0V ±10%, GND = 0V)

TIMING REQUIREMENTS

Symbol	Parameter	8257 Min.	8257 Max.	8257-5 Min.	8257-5 Max.	Unit
T_{CY}	Cycle Time (Period)	0.320	4	0.320	4	μs
T_θ	Clock Active (High)	120	.8T_{CY}	80	.8T_{CY}	ns
T_{QS}	DRQ↑ Setup to CLK↓ (SI, S4)	120		120		ns
T_{QH}	DRQ↓ Hold from HLDA↑ [1]	0		0		ns
T_{HS}	HLDA↑ or ↓Setup to CLK↓(SI, S4) [7]	100	280	100	280	ns
T_{RS}	READY Setup Time to CLK↑(S3, Sw)	30		30		ns
T_{RH}	READY Hold Time from CLK↑(S3, Sw)	30		30		ns

A.C. CHARACTERISTICS—DMA (MASTER) MODE

(8257: T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V)
(8257-5: T_A = 0°C to 70°C, V_{CC} = 5.0V ±10%, GND = 0V)

TIMING RESPONSES

Symbol	Parameter	8257		8257-5		Unit
		Min.	Max.	Min.	Max.	
T_{DQ}	HRQ↑ or ↓Delay from CLK↑ (SI, S4) (measured at 2.0V)		160		160	ns
T_{DQ1}	HRQ↑ or ↓Delay from CLK↑ (SI, S4) (measured at 3.3V)[3]		250		250	ns
T_{AEL}	AEN↑ Delay from CLK↓ (S1)		300		300	ns
T_{AET}	AEN↓ Delay from CLK↑ (SI)		200		200	ns
T_{AEA}	Adr (AB) (Active) Delay from AEN↑ (S1)[1]	20		20		ns
T_{FAAB}	Adr (AB) (Active) Delay from CLK↑ (S1)[2]		250		250	ns
T_{AFAB}	Adr (AB) (Float) Delay from CLK↑ (SI)[2]		150		150	ns
T_{ASM}	Adr (AB) (Stable) Delay from CLK↑ (S1)[2]		250		250	ns
T_{AH}	Adr (AB) (Stable) Hold from CLK↑ (S1)[2]	$T_{ASM}-50$		$T_{ASM}-50$		ns
T_{AHR}	Adr (AB) (Valid) Hold from $\overline{RD}$↑ (S1, SI)[1]	60		60		ns
T_{AHW}	Adr (AB) (Valid) Hold from $\overline{Wr}$↑ (S1, SI)[1]	300		300		ns
T_{FADB}	Adr (DB) (Active) Delay from CLK↑ (S1)[2]		300		300	ns
T_{AFDB}	Adr (DB) (Float) Delay from CLK↑ (S2)[2]	$T_{STT}+20$	250	$T_{STT}+20$	170	ns
T_{ASS}	Adr (DB) Setup to Adr Stb ↓ (S1-S2)[1]	100		100		ns
T_{AHS}	Adr (DB) (Valid) Hold from Adr Stb↓ (S2)[1]	20		20		ns
T_{STL}	Adr Stb↑ Delay from CLK↑ (S1)		200		200	ns
T_{STT}	Adr Stb↓ Delay from CLK↑ (S2)		140		140	ns
T_{SW}	Adr Stb Width (S1-S2)[1]	$T_{CY}-100$		$T_{CY}-100$		ns
T_{ASC}	$\overline{Rd}$↓ or $\overline{Wr}$(Ext)↓ Delay from Adr Stb↓ (S2)[1]	25		25		ns
T_{DBC}	$\overline{RD}$↓ or $\overline{WR}$(Ext)↓ Delay from Adr (DB) (Float) (S2)[1]	-10		-10		ns
T_{AK}	DACK↑ or ↓Delay from CLK↓ (S2, S1) and TC/Mark↑ Delay from CLK↑ (S3) and TC/Mark↓ Delay from CLK↑ (S4)[4]		250		250	ns
T_{DCL}	$\overline{RD}$↓ or $\overline{Wr}$(Ext)↓ Delay from CLK↑ (S2) and $\overline{Wr}$↓ Delay from CLK↑ (S3)[2,5]		200		200	ns
T_{DCT}	$\overline{Rd}$ ↑ Delay from CLK ↓ (S1, SI) and $\overline{Wr}$ ↑ Delay from CLK ↑ (S4) [2, 6]		200		200	ns
T_{FAC}	$\overline{Rd}$ or $\overline{Wr}$ (Active) from CLK↑ (S1)[2]		300		300	ns
T_{AFC}	$\overline{Rd}$ or $\overline{Wr}$ (Float) from CLK↑ (S1)[2]		150		150	ns
T_{RWM}	$\overline{Rd}$ Width (S2-S1 or SI)[1]	$2T_{CY}+T_\theta-50$		$2T_{CY}+T_\theta-50$		ns
T_{WWM}	$\overline{Wr}$ Width (S3-S4)[1]	$T_{CY}-50$		$T_{CY}-50$		ns
T_{WWME}	$\overline{WR}$(Ext) Width (S2-S4)[1]	$2T_{CY}-50$		$2T_{CY}-50$		ns

NOTES:
1. Tracking Parameter.
2. Load = + 50 pF.
3. Load = V_{OH} = 3.3V.
4. $\Delta T_{AK} < 50$ ns.
5. $\Delta T_{DCL} < 50$ ns.
6. $\Delta T_{DCT} < 50$ ns.
7. HLDA must remain stable during t_{HS}.

intel®

8259A/8259A-2/8259A-8 PROGRAMMABLE INTERRUPT CONTROLLER

- **iAPX 86, iAPX 88 Compatible**
- **MCS-80®, MCS-85® Compatible**
- **Eight-Level Priority Controller**
- **Expandable to 64 Levels**
- **Programmable Interrupt Modes**
- **Individual Request Mask Capability**
- **Single +5V Supply (No Clocks)**
- **28-Pin Dual-In-Line Package**
- **Available in EXPRESS**
  - **Standard Temperature Range**
  - **Extended Temperature Range**

The Intel® 8259A Programmable Interrupt Controller handles up to eight vectored priority interrupts for the CPU. It is cascadable for up to 64 vectored priority interrupts without additional circuitry. It is packaged in a 28-pin DIP, uses NMOS technology and requires a single +5V supply. Circuitry is static, requiring no clock input.

The 8259A is designed to minimize the software and real time overhead in handling multi-level priority interrupts. It has several modes, permitting optimization for a variety of system requirements.

The 8259A is fully upward compatible with the Intel® 8259. Software originally written for the 8259 will operate the 8259A in all 8259 equivalent modes (MCS-80/85, Non-Buffered, Edge Triggered).

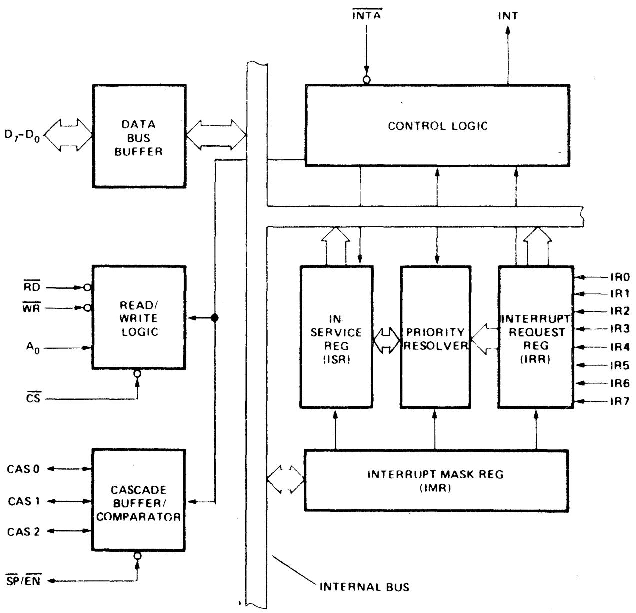


Figure 1. Block Diagram

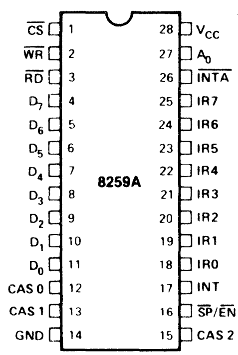


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}	28	I	**Supply:** +5V Supply.
GND	14	I	**Ground.**
$\overline{CS}$	1	I	**Chip Select:** A low on this pin enables $\overline{RD}$ and $\overline{WR}$ communication between the CPU and the 8259A. INTA functions are independent of CS.
$\overline{WR}$	2	I	**Write:** A low on this pin when CS is low enables the 8259A to accept command words from the CPU.
$\overline{RD}$	3	I	**Read:** A low on this pin when CS is low enables the 8259A to release status onto the data bus for the CPU.
D_7–D_0	4–11	I/O	**Bidirectional Data Bus:** Control, status and interrupt-vector information is transferred via this bus.
CAS_0–CAS_2	12, 13, 15	I/O	**Cascade Lines:** The CAS lines form a private 8259A bus to control a multiple 8259A structure. These pins are outputs for a master 8259A and inputs for a slave 8259A.
$\overline{SP}/\overline{EN}$	16	I/O	**Slave Program/Enable Buffer:** This is a dual function pin. When in the Buffered Mode it can be used as an output to control buffer transceivers (EN). When not in the buffered mode it is used as an input to designate a master (SP = 1) or slave (SP = 0).
INT	17	O	**Interrupt:** This pin goes high whenever a valid interrupt request is asserted. It is used to interrupt the CPU, thus it is connected to the CPU's interrupt pin.
IR_0–IR_7	18–25	I	**Interrupt Requests:** Asynchronous inputs. An interrupt request is executed by raising an IR input (low to high), and holding it high until it is acknowledged (Edge Triggered Mode), or just by a high level on an IR input (Level Triggered Mode).
$\overline{INTA}$	26	I	**Interrupt Acknowledge:** This pin is used to enable 8259A interrupt-vector data onto the data bus by a sequence of interrupt acknowledge pulses issued by the CPU.
A_0	27	I	**AO Address Line:** This pin acts in conjunction with the $\overline{CS}$, $\overline{WR}$, and $\overline{RD}$ pins. It is used by the 8259A to decipher various Command Words the CPU writes and status the CPU wishes to read. It is typically connected to the CPU A0 address line (A1 for iAPX 86, 88).

FUNCTIONAL DESCRIPTION

Interrupts in Microcomputer Systems

Microcomputer system design requires that I/O devices such as keyboards, displays, sensors and other components receive servicing in an efficient manner so that large amounts of the total system tasks can be assumed by the microcomputer with little or no effect on throughput.

The most common method of servicing such devices is the *Polled* approach. This is where the processor must test each device in sequence and in effect "ask" each one if it needs servicing. It is easy to see that a large portion of the main program is looping through this continuous polling cycle and that such a method would have a serious, detrimental effect on system throughput, thus limiting the tasks that could be assumed by the microcomputer and reducing the cost effectiveness of using such devices.

A more desirable method would be one that would allow the microprocessor to be executing its main program and only stop to service peripheral devices when it is told to do so by the device itself. In effect, the method would provide an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed and fetch a new routine that will service the requesting device. Once this servicing is complete, however, the processor would resume exactly where it left off.

This method is called *Interrupt*. It is easy to see that system throughput would drastically increase, and thus more tasks could be assumed by the microcomputer to further enhance its cost effectiveness.

The Programmable Interrupt Controller (PIC) functions as an overall manager in an Interrupt-Driven system environment. It accepts requests from the peripheral equipment, determines which of the incoming requests is of the highest importance (priority), ascertains whether the incoming request has a higher priority value than the level currently being serviced, and issues an interrupt to the CPU based on this determination.

Each peripheral device or structure usually has a special program or "routine" that is associated with its specific functional or operational requirements; this is referred to as a "service routine". The PIC, after issuing an Interrupt to the CPU, must somehow input information into the CPU that can "point" the Program Counter to the service routine associated with the requesting device. This "pointer" is an address in a vectoring table and will often be referred to, in this document, as vectoring data.

The 8259A

The 8259A is a device specifically designed for use in real time, interrupt driven microcomputer systems. It manages eight levels or requests and has built-in features for expandability to other 8259A's (up to 64 levels). It is programmed by the system's software as an I/O peripheral. A selection of priority modes is available to the programmer so that the manner in which the requests are processed by the 8259A can be configured to match his system requirements. The priority modes can be changed or reconfigured dynamically at any time during the main program. This means that the complete interrupt structure can be defined as required, based on the total system environment.

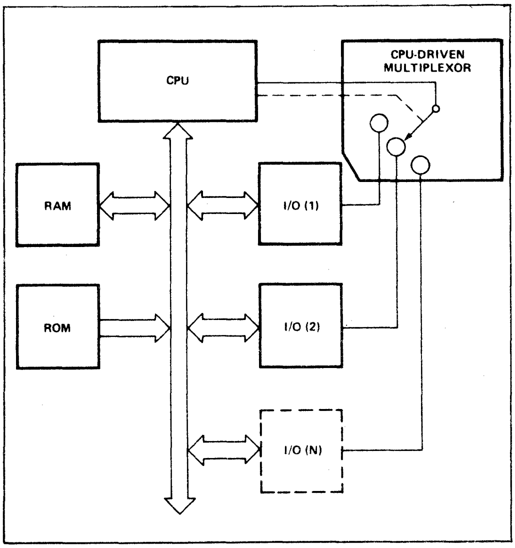


Figure 3a. Polled Method

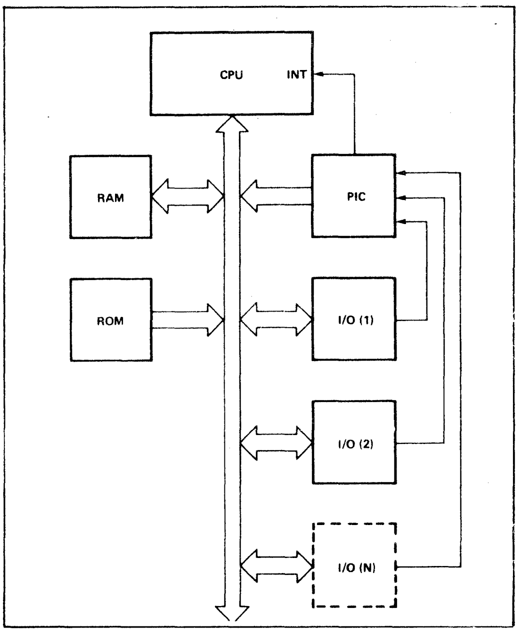


Figure 3b. Interrupt Method

INTERRUPT REQUEST REGISTER (IRR) AND IN-SERVICE REGISTER (ISR)

The interrupts at the IR input lines are handled by two registers in cascade, the Interrupt Request Register (IRR) and the In-Service Register (ISR). The IRR is used to store all the interrupt levels which are requesting service; and the ISR is used to store all the interrupt levels which are being serviced.

PRIORITY RESOLVER

This logic block determines the priorities of the bits set in the IRR. The highest priority is selected and strobed into the corresponding bit of the ISR during $\overline{\text{INTA}}$ pulse.

INTERRUPT MASK REGISTER (IMR)

The IMR stores the bits which mask the interrupt lines to be masked. The IMR operates on the IRR. Masking of a higher priority input will not affect the interrupt request lines of lower priority.

INT (INTERRUPT)

This output goes directly to the CPU interrupt input. The V_{OH} level on this line is designed to be fully compatible with the 8080A, 8085A and 8086 input levels.

$\overline{\text{INTA}}$ (INTERRUPT ACKNOWLEDGE)

$\overline{\text{INTA}}$ pulses will cause the 8259A to release vectoring information onto the data bus. The format of this data depends on the system mode (μPM) of the 8259A.

DATA BUS BUFFER

This 3-state, bidirectional 8-bit buffer is used to interface the 8259A to the system Data Bus. Control words and status information are transferred through the Data Bus Buffer.

READ/WRITE CONTROL LOGIC

The function of this block is to accept OUTput commands from the CPU. It contains the Initialization Command Word (ICW) registers and Operation Command Word (OCW) registers which store the various control formats for device operation. This function block also allows the status of the 8259A to be transferred onto the Data Bus.

$\overline{\text{CS}}$ (CHIP SELECT)

A LOW on this input enables the 8259A. No reading or writing of the chip will occur unless the device is selected.

$\overline{\text{WR}}$ (WRITE)

A LOW on this input enables the CPU to write control words (ICWs and OCWs) to the 8259A.

$\overline{\text{RD}}$ (READ)

A LOW on this input enables the 8259A to send the status of the Interrupt Request Register (IRR), In Service Register (ISR), the Interrupt Mask Register (IMR), or the Interrupt level onto the Data Bus.

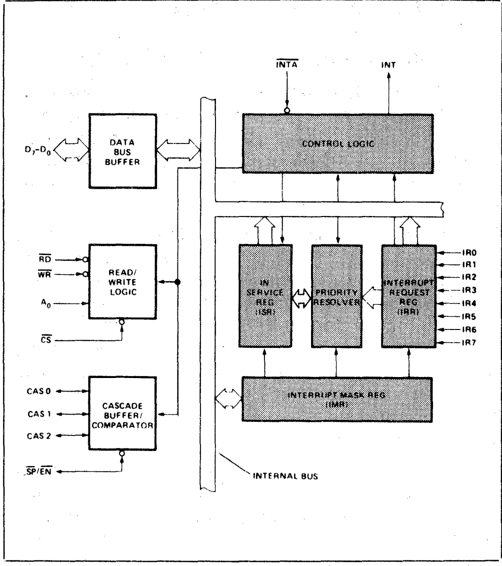


Figure 4a. 8259A Block Diagram

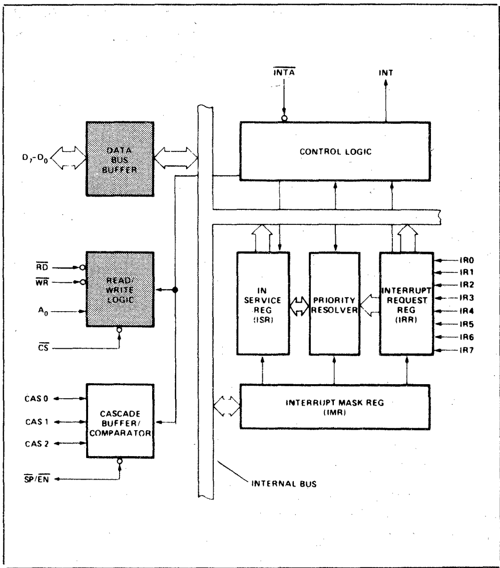


Figure 4b. 8259A Block Diagram

A_0

This input signal is used in conjunction with $\overline{\text{WR}}$ and $\overline{\text{RD}}$ signals to write commands into the various command registers, as well as reading the various status registers of the chip. This line can be tied directly to one of the address lines.

THE CASCADE BUFFER/COMPARATOR

This function block stores and compares the IDs of all 8259A's used in the system. The associated three I/O pins (CAS0-2) are outputs when the 8259A is used as a master and are inputs when the 8259A is used as a slave. As a master, the 8259A sends the ID of the interrupting slave device onto the CAS0-2 lines. The slave thus selected will send its preprogrammed subroutine address onto the Data Bus during the next one or two consecutive $\overline{\text{INTA}}$ pulses. (See section "Cascading the 8259A".)

INTERRUPT SEQUENCE

The powerful features of the 8259A in a microcomputer system are its programmability and the interrupt routine addressing capability. The latter allows direct or indirect jumping to the specific interrupt routine requested without any polling of the interrupting devices. The normal sequence of events during an interrupt depends on the type of CPU being used.

The events occur as follows in an MCS-80/85 system:

1. One or more of the INTERRUPT REQUEST lines (IR7-0) are raised high, setting the corresponding IRR bit(s).
2. The 8259A evaluates these requests, and sends an INT to the CPU, if appropriate.
3. The CPU acknowledges the INT and responds with an $\overline{\text{INTA}}$ pulse.
4. Upon receiving an $\overline{\text{INTA}}$ from the CPU group, the highest priority ISR bit is set, and the corresponding IRR bit is reset. The 8259A will also release a CALL instruction code (11001101) onto the 8-bit Data Bus through its D7-0 pins.
5. This CALL instruction will initiate two more $\overline{\text{INTA}}$ pulses to be sent to the 8259A from the CPU group.
6. These two $\overline{\text{INTA}}$ pulses allow the 8259A to release its preprogrammed subroutine address onto the Data Bus. The lower 8-bit address is released at the first $\overline{\text{INTA}}$ pulse and and the higher 8-bit address is released at the second $\overline{\text{INTA}}$ pulse.
7. This completes the 3-byte CALL instruction released by the 8259A. In the AEOI mode the ISR bit is reset at the end of the third $\overline{\text{INTA}}$ pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt sequence.

The events occurring in an iAPX 86 system are the same until step 4.

4. Upon receiving an $\overline{\text{INTA}}$ from the CPU group, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259A does not drive the Data Bus during this cycle.
5. The iAPX 86/10 will initiate a second $\overline{\text{INTA}}$ pulse. During this pulse, the 8259A releases an 8-bit pointer onto the Data Bus where it is read by the CPU.
6. This completes the interrupt cycle. In the AEOI mode the ISR bit is reset at the end of the second $\overline{\text{INTA}}$ pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt subroutine.

If no interrupt request is present at step 4 of either sequence (i.e., the request was too short in duration) the 8259A will issue an interrupt level 7. Both the vectoring bytes and the CAS lines will look like an interrupt level 7 was requested.

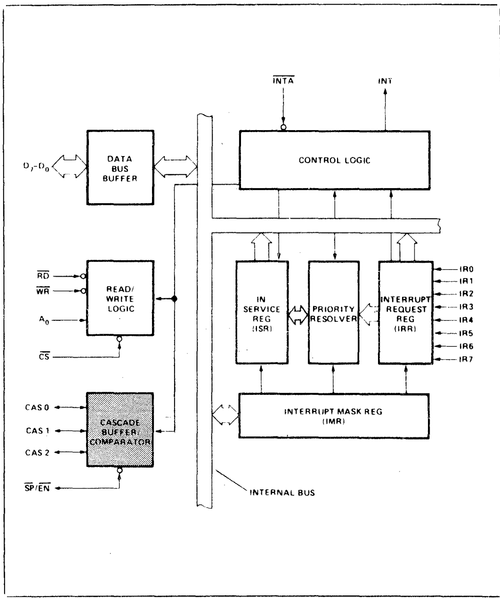


Figure 4c. 8259A Block Diagram

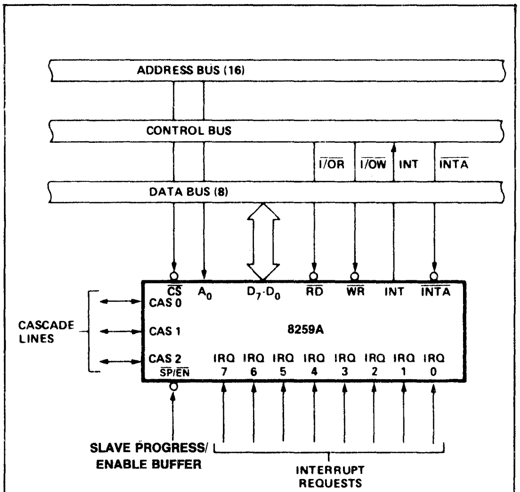


Figure 5. 8259A Interface to Standard System Bus

INTERRUPT SEQUENCE OUTPUTS

MCS-80®, MCS-85®

This sequence is timed by three $\overline{INTA}$ pulses. During the first $\overline{INTA}$ pulse the CALL opcode is enabled onto the data bus.

Content of First Interrupt Vector Byte

	D7	D6	D5	D4	D3	D2	D1	D0
CALL CODE	1	1	0	0	1	1	0	1

During the second $\overline{INTA}$ pulse the lower address of the appropriate service routine is enabled onto the data bus. When Interval = 4 bits A_5–A_7 are programmed, while A_0–A_4 are automatically inserted by the 8259A. When Interval = 8 only A_6 and A_7 are programmed, while A_0–A_5 are automatically inserted.

Content of Second Interrupt Vector Byte

IR	Interval = 4							
	D7	D6	D5	D4	D3	D2	D1	D0
7	A7	A6	A5	1	1	1	0	0
6	A7	A6	A5	1	1	0	0	0
5	A7	A6	A5	1	0	1	0	0
4	A7	A6	A5	1	0	0	0	0
3	A7	A6	A5	0	1	1	0	0
2	A7	A6	A5	0	1	0	0	0
1	A7	A6	A5	0	0	1	0	0
0	A7	A6	A5	0	0	0	0	0

IR	Interval = 8							
	D7	D6	D5	D4	D3	D2	D1	D0
7	A7	A6	1	1	1	0	0	0
6	A7	A6	1	1	0	0	0	0
5	A7	A6	1	0	1	0	0	0
4	A7	A6	1	0	0	0	0	0
3	A7	A6	0	1	1	0	0	0
2	A7	A6	0	1	0	0	0	0
1	A7	A6	0	0	1	0	0	0
0	A7	A6	0	0	0	0	0	0

During the third INTA pulse the higher address of the appropriate service routine, which was programmed as byte 2 of the initialization sequence ($A_8 - A_{15}$), is enabled onto the bus.

Content of Third Interrupt Vector Byte

D7	D6	D5	D4	D3	D2	D1	D0
A15	A14	A13	A12	A11	A10	A9	A8

iAPX 86, iAPX 88

iAPX 86 mode is similar to MCS-80 mode except that only two Interrupt Acknowledge cycles are issued by the processor and no CALL opcode is sent to the processor. The first interrupt acknowledge cycle is similar to that of MCS-80, 85 systems in that the 8259A uses it to internally freeze the state of the interrupts for priority resolution and as a master it issues the interrupt code on the cascade lines at the end of the INTA pulse. On this first cycle it does not issue any data to the processor and leaves its data bus buffers disabled. On the second interrupt acknowledge cycle in iAPX 86 mode the master (or slave if so programmed) will send a byte of data to the processor with the acknowledged interrupt code composed as follows (note the state of the ADI mode control is ignored and A_5–A_{11} are unused in iAPX 86 mode):

Content of Interrupt Vector Byte for iAPX 86 System Mode

	D7	D6	D5	D4	D3	D2	D1	D0
IR7	T7	T6	T5	T4	T3	1	1	1
IR6	T7	T6	T5	T4	T3	1	1	0
IR5	T7	T6	T5	T4	T3	1	0	1
IR4	T7	T6	T5	T4	T3	1	0	0
IR3	T7	T6	T5	T4	T3	0	1	1
IR2	T7	T6	T5	T4	T3	0	1	0
IR1	T7	T6	T5	T4	T3	0	0	1
IR0	T7	T6	T5	T4	T3	0	0	0

PROGRAMMING THE 8259A

The 8259A accepts two types of command words generated by the CPU:

1. *Initialization Command Words (ICWs):* Before normal operation can begin, each 8259A in the system must be brought to a starting point — by a sequence of 2 to 4 bytes timed by $\overline{WR}$ pulses.
2. *Operation Command Words (OCWs):* These are the command words which command the 8259A to operate in various interrupt modes. These modes are:
   a. Fully nested mode
   b. Rotating priority mode
   c. Special mask mode
   d. Polled mode

The OCWs can be written into the 8259A anytime after initialization.

INITIALIZATION COMMAND WORDS (ICWS)

GENERAL

Whenever a command is issued with A0 = 0 and D4 = 1, this is interpreted as Initialization Command Word 1 (ICW1). ICW1 starts the initialization sequence during which the following automatically occur.

a. The edge sense circuit is reset, which means that following initialization, an interrupt request (IR) input must make a low-to-high transition to generate an interrupt.
b. The Interrupt Mask Register is cleared.
c. IR7 input is assigned priority 7.
d. The slave mode address is set to 7.
e. Special Mask Mode is cleared and Status Read is set to IRR.
f. If IC4=0, then all functions selected in ICW4 are set to zero. (Non-Buffered mode*, no Auto-EOI, MCS-80, 85 system).

***Note:** Master/Slave in ICW4 is only used in the buffered mode.

INITIALIZATION COMMAND WORDS 1 AND 2 (ICW1, ICW2)

A_5–A_{15}: *Page starting address of service routines.* In an MCS 80/85 system, the 8 request levels will generate CALLs to 8 locations equally spaced in memory. These can be programmed to be spaced at intervals of 4 or 8 memory locations, thus the 8 routines will occupy a page of 32 or 64 bytes, respectively.

The address format is 2 bytes long (A_0–A_{15}). When the routine interval is 4, A_0–A_4 are automatically inserted by the 8259A, while A_5–A_{15} are programmed externally. When the routine interval is 8, A_0–A_5 are automatically inserted by the 8259A, while A_6–A_{15} are programmed externally.

The 8-byte interval will maintain compatibility with current software, while the 4-byte interval is best for a compact jump table.

In an iAPX 86 system A_{15}–A_{11} are inserted in the five most significant bits of the vectoring byte and the 8259A sets the three least significant bits according to the interrupt level. A_{10}–A_5 are ignored and ADI (Address interval) has no effect.

LTIM: If LTIM = 1, then the 8259A will operate in the level interrupt mode. Edge detect logic on the interrupt inputs will be disabled.

ADI: CALL address interval. ADI = 1 then interval = 4; ADI = 0 then interval = 8.

SNGL: Single. Means that this is the only 8259A in the system. If SNGL = 1 no ICW3 will be issued.

IC4: If this bit is set — ICW4 has to be read. If ICW4 is not needed, set IC4 = 0.

INITIALIZATION COMMAND WORD 3 (ICW3)

This word is read only when there is more than one 8259A in the system and cascading is used, in which case SNGL = 0. It will load the 8-bit slave register. The functions of this register are:

a. In the master mode (either when SP = 1, or in buffered mode when M/S = 1 in ICW4) a "1" is set for each slave in the system. The master then will release byte 1 of the call sequence (for MCS-80/85 system) and will enable the corresponding slave to release bytes 2 and 3 (for iAPX 86 only byte 2) through the cascade lines.

b. In the slave mode (either when $\overline{SP}$ = 0, or if BUF = 1 and M/S = 0 in ICW4) bits 2-0 identify the slave. The slave compares its cascade input with these bits and, if they are equal, bytes 2 and 3 of the call sequence (or just byte 2 for iAPX 86 are released by it on the Data Bus.

INITIALIZATION COMMAND WORD 4 (ICW4)

SFNM: If SFNM = 1 the special fully nested mode is programmed.

BUF: If BUF = 1 the buffered mode is programmed. In buffered mode $\overline{SP}/\overline{EN}$ becomes an enable output and the master/slave determination is by M/S.

M/S: If buffered mode is selected: M/S = 1 means the 8259A is programmed to be a master, M/S = 0 means the 8259A is programmed to be a slave. If BUF = 0, M/S has no function.

AEOI: If AEOI = 1 the automatic end of interrupt mode is programmed.

μPM: Microprocessor mode: μPM = 0 sets the 8259A for MCS-80, 85 system operation, μPM = 1 sets the 8259A for iAPX 86 system operation.

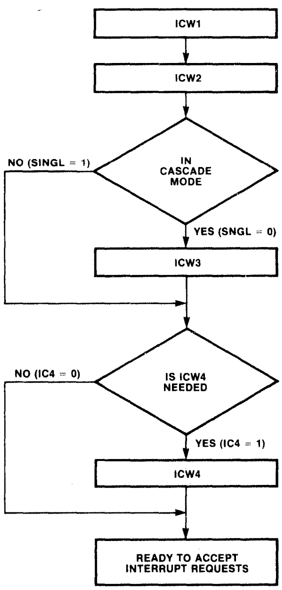


Figure 6. Initialization Sequence

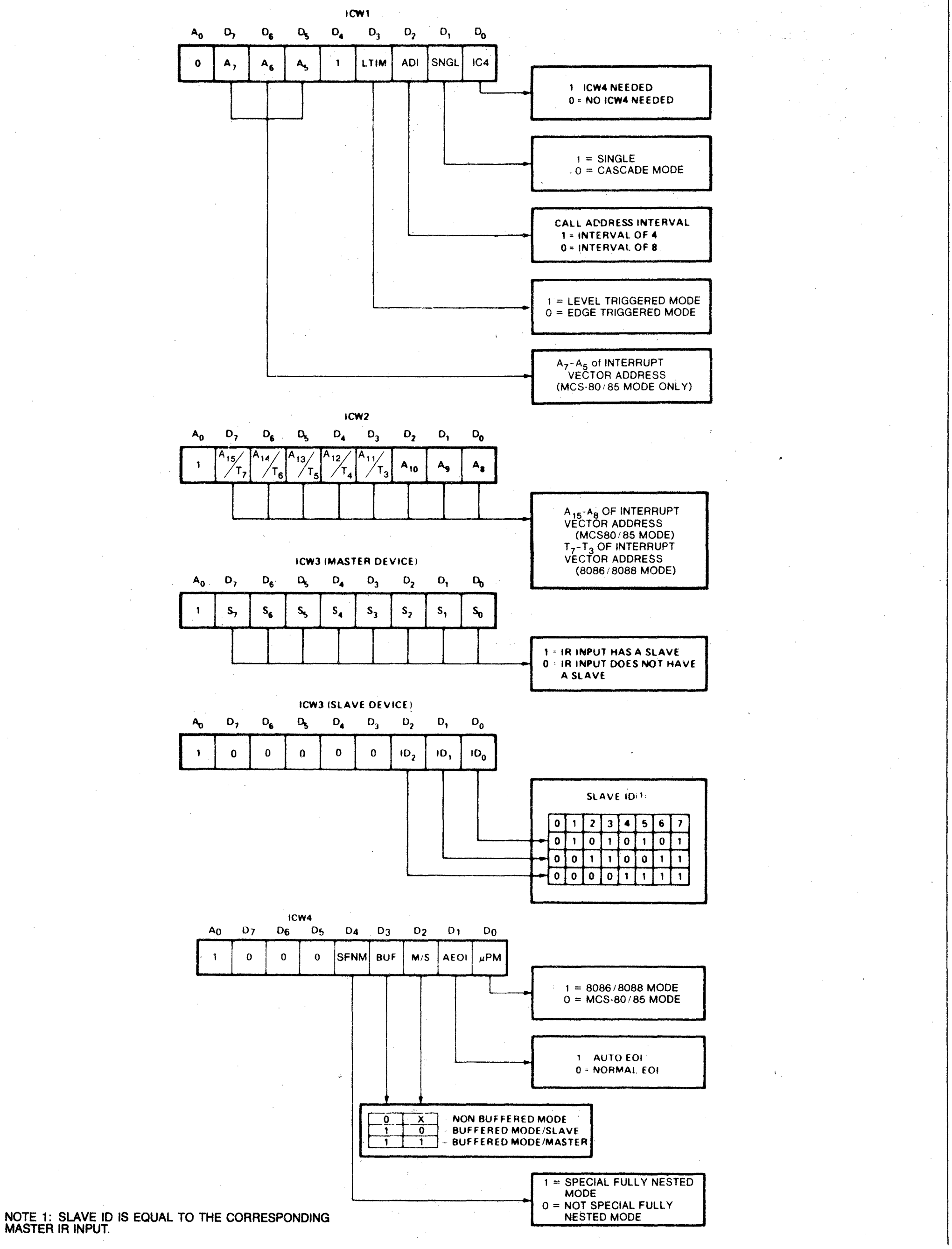


Figure 7. Initialization Command Word Format

OPERATION COMMAND WORDS (OCWs)

After the Initialization Command Words (ICWs) are programmed into the 8259A, the chip is ready to accept interrupt requests at its input lines. However, during the 8259A operation, a selection of algorithms can command the 8259A to operate in various modes through the Operation Command Words (OCWs).

OPERATION CONTROL WORDS (OCWs)

OCW1

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	M7	M6	M5	M4	M3	M2	M1	M0

OCW2

A0	D7	D6	D5	D4	D3	D2	D1	D0
0	R	SL	EOI	0	0	L2	L1	L0

OCW3

A0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	ESMM	SMM	0	1	P	RR	RIS

OPERATION CONTROL WORD 1 (OCW1)

OCW1 sets and clears the mask bits in the interrupt Mask Register (IMR). $M_7 - M_0$ represent the eight mask bits. M = 1 indicates the channel is masked (inhibited), M = 0 indicates the channel is enabled.

OPERATION CONTROL WORD 2 (OCW2)

R, SL, EOI — These three bits control the Rotate and End of Interrupt modes and combinations of the two. A chart of these combinations can be found on the Operation Command Word Format.

L_2, L_1, L_0—These bits determine the interrupt level acted upon when the SL bit is active.

OPERATION CONTROL WORD 3 (OCW3)

ESMM — Enable Special Mask Mode. When this bit is set to 1 it enables the SMM bit to set or reset the Special Mask Mode. When ESMM = 0 the SMM bit becomes a "don't care".

SMM — Special Mask Mode. If ESMM = 1 and SMM = 1 the 8259A will enter Special Mask Mode. If ESMM = 1 and SMM = 0 the 8259A will revert to normal mask mode. When ESMM = 0, SMM has no effect.

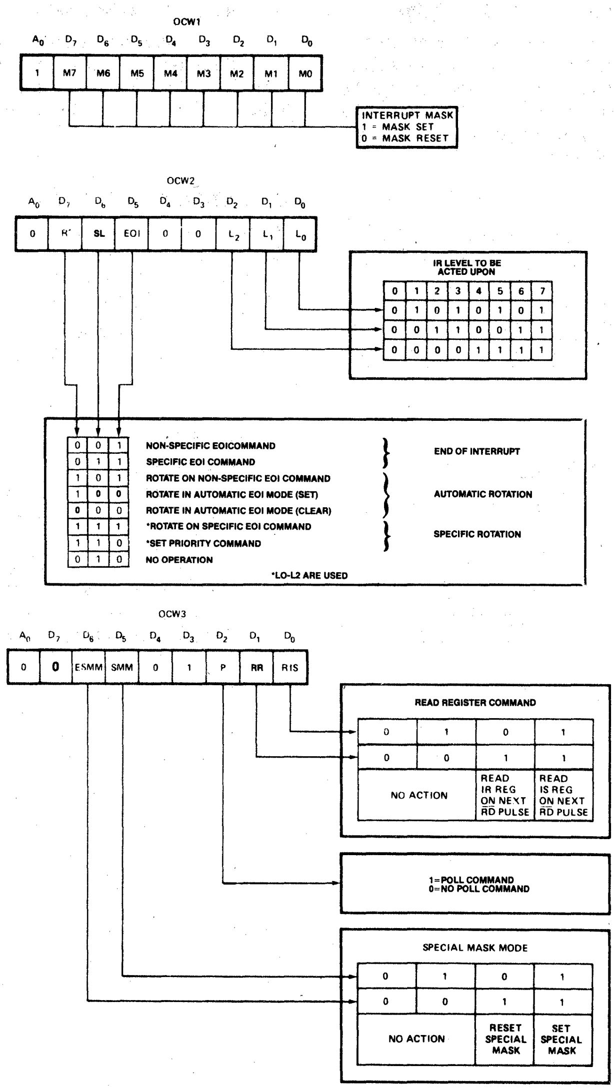


Figure 8. Operation Command Word Format

FULLY NESTED MODE

This mode is entered after initialization unless another mode is programmed. The interrupt requests are ordered in priority form 0 through 7 (0 highest). When an interrupt is acknowledged the highest priority request is determined and its vector placed on the bus. Additionally, a bit of the Interrupt Service register (ISO-7) is set. This bit remains set until the microprocessor issues an End of Interrupt (EOI) command immediately before returning from the service routine, or if AEOI (Automatic End of Interrupt) bit is set, until the trailing edge of the last INTA. While the IS bit is set, all further interrupts of the same or lower priority are inhibited, while higher levels will generate an interrupt (which will be acknowledged only if the microprocessor internal Interrupt enable flip-flop has been re-enabled through software).

After the initialization sequence, IR0 has the highest priority and IR7 the lowest. Priorities can be changed, as will be explained, in the rotating priority mode.

END OF INTERRUPT (EOI)

The In Service (IS) bit can be reset either automatically following the trailing edge of the last in sequence $\overline{INTA}$ pulse (when AEOI bit in ICW1 is set) or by a command word that must be issued to the 8259A before returning from a service routine (EOI command). An EOI command must be issued twice if in the Cascade mode, once for the master and once for the corresponding slave.

There are two forms of EOI command: Specific and Non-Specific. When the 8259A is operated in modes which preserve the fully nested structure, it can determine which IS bit to reset on EOI. When a Non-Specific EOI command is issued the 8259A will automatically reset the highest IS bit of those that are set, since in the fully nested mode the highest IS level was necessarily the last level acknowledged and serviced. A non-specific EOI can be issued with OCW2 (EOI = 1, SL = 0, R = 0).

When a mode is used which may disturb the fully nested structure, the 8259A may no longer be able to determine the last level acknowledged. In this case a Specific End of Interrupt must be issued which includes as part of the command the IS level to be reset. A specific EOI can be issued with OCW2 (EOI = 1, SL = 1, R = 0, and LO-L2 is the binary level of the IS bit to be reset).

It should be noted that an IS bit that is masked by an IMR bit will not be cleared by a non-specific EOI if the 8259A is in the Special Mask Mode.

AUTOMATIC END OF INTERRUPT (AEOI) MODE

If AEOI = 1 in ICW4, then the 8259A will operate in AEOI mode continuously until reprogrammed by ICW4. In this mode the 8259A will automatically perform a non-specific EOI operation at the trailing edge of the last interrupt acknowledge pulse (third pulse in MCS-80/85, second in iAPX 86). Note that from a system standpoint, this mode should be used only when a nested multilevel interrupt structure is not required within a single 8259A.

The AEOI mode can only be used in a master 8259A and not a slave.

AUTOMATIC ROTATION
(Equal Priority Devices)

In some applications there are a number of interrupting devices of equal priority. In this mode a device, after being serviced, receives the lowest priority, so a device requesting an interrupt will have to wait, in the worst case until each of 7 other devices are serviced at most *once*. For example, if the priority and "in service" status is:

Before Rotate (IR4 the highest priority requiring service)

	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
"IS" Status	0	1	0	1	0	0	0	0
	Lowest Priority						Highest Priority	
Priority Status	7	6	5	4	3	2	1	0

After Rotate (IR4 was serviced, all other priorities rotated correspondingly)

	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
"IS" Status	0	1	0	0	0	0	0	0
	Highest Priority						Lowest Priority	
Priority Status	2	1	0	7	6	5	4	3

There are two ways to accomplish Automatic Rotation using OCW2, the Rotation on Non-Specific EOI Command (R = 1, SL = 0, EOI = 1) and the Rotate in Automatic EOI Mode which is set by (R = 1, SL = 0, EOI = 0) and cleared by (R = 0, SL = 0, EOI = 0).

SPECIFIC ROTATION
(Specific Priority)

The programmer can change priorities by programming the bottom priority and thus fixing all other priorities; i.e., if IR5 is programmed as the bottom priority device, then IR6 will have the highest one.

The Set Priority command is issued in OCW2 where: R = 1, SL = 1; LO–L2 is the binary priority level code of the bottom priority device.

Observe that in this mode internal status is updated by software control during OCW2. However, it is independent of the End of Interrupt (EOI) command (also executed by OCW2). Priority changes can be executed during an EOI command by using the Rotate on Specific EOI command in OCW2 (R = 1, SL = 1, EOI = 1 and LO–L2 = IR level to receive bottom priority).

INTERRUPT MASKS

Each Interrupt Request input can be masked individually by the Interrupt Mask Register (IMR) programmed through OCW1. Each bit in the IMR masks one interrupt channel if it is set (1). Bit 0 masks IR0, Bit 1 masks IR1 and so forth. Masking an IR channel does not affect the other channels operation.

SPECIAL MASK MODE

Some applications may require an interrupt service routine to dynamically alter the system priority structure during its execution under software control. For example, the routine may wish to inhibit lower priority requests for a portion of its execution but enable some of them for another portion.

The difficulty here is that if an Interrupt Request is acknowledged and an End of Interrupt command did not reset its IS bit (i.e., while executing a service routine), the 8259A would have inhibited all lower priority requests with no easy way for the routine to enable them

That is where the Special Mask Mode comes in. In the special Mask Mode, when a mask bit is set in OCW1, it inhibits further interrupts at that level *and enables* interrupts from *all other* levels (lower as well as higher) that are not masked.

Thus, any interrupts may be selectively enabled by loading the mask register.

The special Mask Mode is set by OCW3 where: SSMM = 1, SMM = 1, and cleared where SSMM = 1, SMM = 0.

POLL COMMAND

In this mode the INT output is not used or the microprocessor internal Interrupt Enable flip-flop is reset, disabling its interrupt input. Service to devices is achieved by software using a Poll command.

The Poll command is issued by setting P = "1" in OCW3. The 8259A treats the next $\overline{RD}$ pulse to the 8259A (i.e., $\overline{RD}$ = 0, $\overline{CS}$ = 0) as an interrupt acknowledge, sets the appropriate IS bit if there is a request, and reads the priority level. Interrupt is frozen from $\overline{WR}$ to $\overline{RD}$.

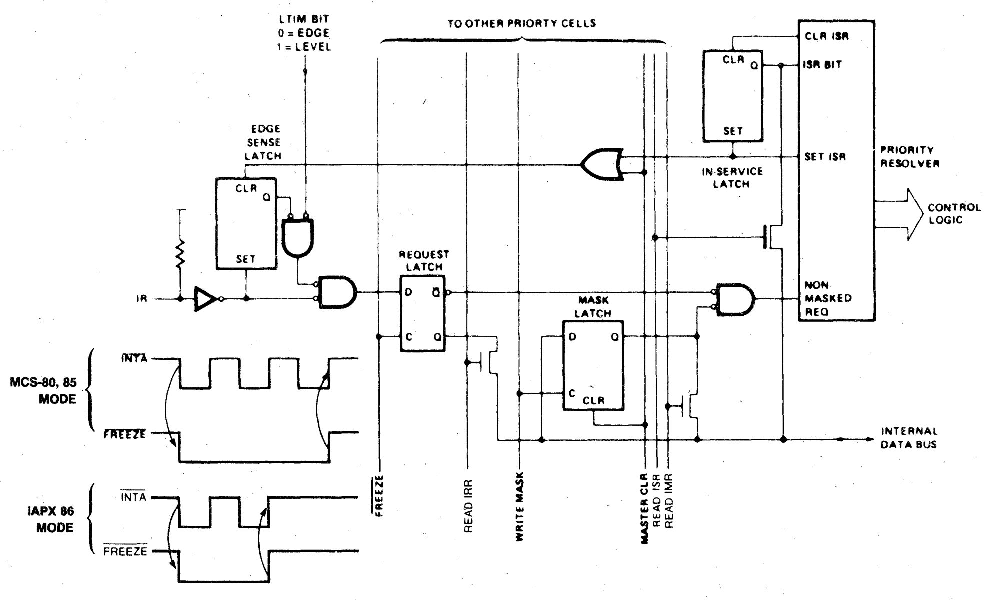
The word enabled onto the data bus during $\overline{RD}$ is:

D7	D6	D5	D4	D3	D2	D1	D0
I	—	—	—	—	W2	W1	W0

W0-W2: Binary code of the highest priority level requesting service.

I: Equal to a "1" if there is an interrupt.

This mode is useful if there is a routine command common to several levels so that the $\overline{INTA}$ sequence is not needed (saves ROM space). Another application is to use the poll mode to expand the number of priority levels to more than 64.



NOTES
1. MASTER CLEAR ACTIVE ONLY DURING ICW1
2. FREEZE/ IS ACTIVE DURING $\overline{INTA}$/ AND POLL SEQUENCES ONLY
3. TRUTH TABLE FOR D-LATCH

C	D	Q	OPERATION
1	Di	Di	FOLLOW
0	X	Qn-1	HOLD

Figure 9. Priority Cell—Simplified Logic Diagram

READING THE 8259A STATUS

The input status of several internal registers can be read to update the user information on the system. The following registers can be read via OCW3 (IRR and ISR or OCW1 [IMR]).

Interrupt Request Register (IRR): 8-bit register which contains the levels requesting an interrupt to be acknowledged. The highest request level is reset from the IRR when an interrupt is acknowledged. (Not affected by IMR.)

In-Service Register (ISR): 8-bit register which contains the priority levels that are being serviced. The ISR is updated when an End of Interrupt Command is issued.

Interrupt Mask Register: 8-bit register which contains the interrupt request lines which are masked.

The IRR can be read when, prior to the RD pulse, a Read Register Command is issued with OCW3 (RR = 1, RIS = 0.)

The ISR can be read when, prior to the RD pulse, a Read Register Command is issued with OCW3 (RR = 1, RIS = 1).

There is no need to write an OCW3 before every status read operation, as long as the status read corresponds with the previous one; i.e., the 8259A "remembers" whether the IRR or ISR has been previously selected by the OCW3. This is not true when poll is used.

After initialization the 8259A is set to IRR.

For reading the IMR, no OCW3 is needed. The output data bus will contain the IMR whenever $\overline{RD}$ is active and AO = 1 (OCW1).

Polling overrides status read when P = 1, RR = 1 in OCW3.

EDGE AND LEVEL TRIGGERED MODES

This mode is programmed using bit 3 in ICW1.

If LTIM = '0', an interrupt request will be recognized by a low to high transition on an IR input. The IR input can remain high without generating another interrupt.

If LTIM = '1', an interrupt request will be recognized by a 'high' level on IR Input, and there is no need for an edge detection. The interrupt request must be removed before the EOI command is issued or the CPU interrupt is enabled to prevent a second interrupt from occurring.

The priority cell diagram shows a conceptual circuit of the level sensitive and edge sensitive input circuitry of the 8259A. Be sure to note that the request latch is a transparent D type latch.

In both the edge and level triggered modes the IR inputs must remain high until after the falling edge of the first INTA. If the IR input goes low before this time a DEFAULT IR7 will occur when the CPU acknowledges the interrupt. This can be a useful safeguard for detecting interrupts caused by spurious noise glitches on the IR inputs. To implement this feature the IR7 routine is used for "clean up" simply executing a return instruction, thus ignoring the interrupt. If IR7 is needed for other purposes a default IR7 can still be detected by reading the ISR. A normal IR7 interrupt will set the corresponding ISR bit, a default IR7 won't. If a default IR7 routine occurs during a normal IR7 routine, however, the ISR will remain set. In this case it is necessary to keep track of whether or not the IR7 routine was previously entered. If another IR7 occurs it is a default.

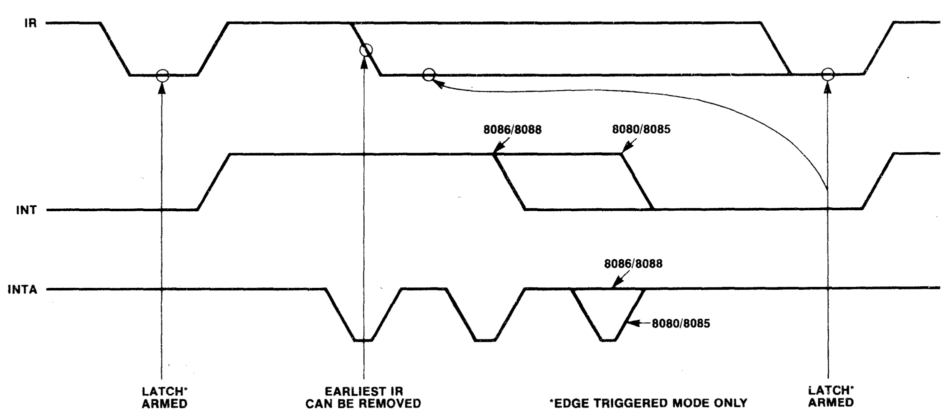


Figure 10. IR Triggering Timing Requirements

THE SPECIAL FULLY NESTED MODE

This mode will be used in the case of a big system where cascading is used, and the priority has to be conserved within each slave. In this case the fully nested mode will be programmed to the master (using ICW4). This mode is similar to the normal nested mode with the following exceptions:

a. When an interrupt request from a certain slave is in service this slave is not locked out from the master's priority logic and further interrupt requests from higher priority IR's within the slave will be recognized by the master and will initiate interrupts to the processor. (In the normal nested mode a slave is masked out when its request is in service and no higher requests from the same slave can be serviced.)

b. When exiting the Interrupt Service routine the software has to check whether the interrupt serviced was the only one from that slave. This is done by sending a non-specific End of Interrupt (EOI) command to the slave and then reading its In-Service register and checking for zero. If it is empty, a non-specific EOI can be sent to the master too. If not, no EOI should be sent.

BUFFERED MODE

When the 8259A is used in a large system where bus driving buffers are required on the data bus and the cascading mode is used, there exists the problem of enabling buffers.

The buffered mode will structure the 8259A to send an enable signal on $\overline{SP}/\overline{EN}$ to enable the buffers. In this mode, whenever the 8259A's data bus outputs are enabled, the $\overline{SP}/\overline{EN}$ output becomes active.

This modification forces the use of software programming to determine whether the 8259A is a master or a slave. Bit 3 in ICW4 programs the buffered mode, and bit 2 in ICW4 determines whether it is a master or a slave.

CASCADE MODE

The 8259A can be easily interconnected in a system of one master with up to eight slaves to handle up to 64 priority levels.

The master controls the slaves through the 3 line cascade bus. The cascade bus acts like chip selects to the slaves during the $\overline{INTA}$ sequence.

In a cascade configuration, the slave interrupt outputs are connected to the master interrupt request inputs. When a slave request line is activated and afterwards acknowledged, the master will enable the corresponding slave to release the device routine address during bytes 2 and 3 of INTA. (Byte 2 only for 8086/8088).

The cascade bus lines are normally low and will contain the slave address code from the trailing edge of the first INTA pulse to the trailing edge of the third pulse. Each 8259A in the system must follow a separate initialization sequence and can be programmed to work in a different mode. An EOI command must be issued twice: once for the master and once for the corresponding slave. An address decoder is required to activate the Chip Select (CS) input of each 8259A.

The cascade lines of the Master 8259A are activated only for slave inputs, non slave inputs leave the cascade line inactive (low).

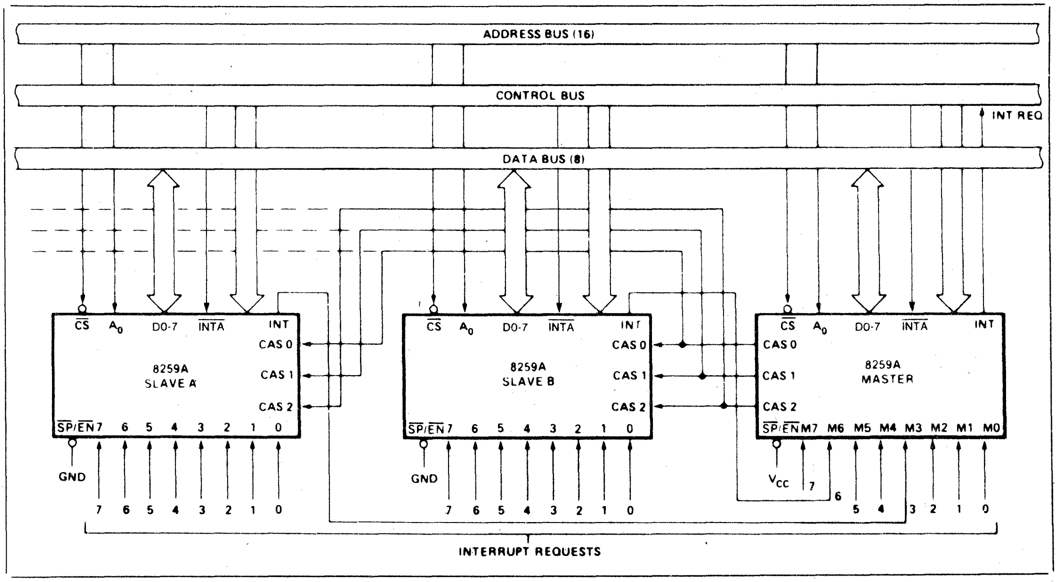


Figure 11. Cascading the 8259A

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .... 0°C to 70°C
Storage Temperature .... −65°C to +150°C
Voltage on Any Pin
  with Respect to Ground .... −0.5V to +7V
Power Dissipation .... 1 Watt

***NOTICE:** *Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.*

D.C. CHARACTERISTICS [T_A = 0°C to 70°C, V_{CC} = 5V ±5% (8259A-8), V_{CC} = 5V ±10% (8259A, 8259A-2)]

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0*	V_{CC} +0.5V	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400μA
$V_{OH(INT)}$	Interrupt Output High Voltage	3.5		V	I_{OH} = −100μA
		2.4		V	I_{OH} = −400μA
I_{LI}	Input Load Current	−10	+10	μA	0V ≤ V_{IN} ≤ V_{CC}
I_{LOL}	Output Leakage Current	−10	+10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}
I_{CC}	V_{CC} Supply Current		85	mA	
I_{LIR}	IR Input Load Current		−300	μA	V_{IN} = 0
			10	μA	V_{IN} = V_{CC}

***Note:** For Extended Temperature EXPRESS V_{IH} = 2.3V.

CAPACITANCE (T_A = 25°C; V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	fc = 1 MHZ
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to V_{SS}

A.C. CHARACTERISTICS [T_A = 0°C to 70°C, V_{CC} = 5V ±5% (8259A-8), V_{CC} = 5V ± 10% (8259A, 8259A-2)]

TIMING REQUIREMENTS

Symbol	Parameter	8259A-8		8259A		8259A-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TAHRL	AO/$\overline{CS}$ Setup to $\overline{RD}$/$\overline{INTA}$↓	50		0		0		ns	
TRHAX	AO/$\overline{CS}$ Hold after $\overline{RD}$/$\overline{INTA}$↑	5		0		0		ns	
TRLRH	$\overline{RD}$ Pulse Width	420		235		160		ns	
TAHWL	AO/$\overline{CS}$ Setup to $\overline{WR}$↓	50		0		0		ns	
TWHAX	AO/$\overline{CS}$ Hold after $\overline{WR}$↑	20		0		0		ns	
TWLWH	$\overline{WR}$ Pulse Width	400		290		190		ns	
TDVWH	Data Setup to $\overline{WR}$↑	300		240		160		ns	
TWHDX	Data Hold after $\overline{WR}$↑	40		0		0		ns	
TJLJH	Interrupt Request Width (Low)	100		100		100		ns	See Note 1
TCVIAL	Cascade Setup to Second or Third $\overline{INTA}$↓ (Slave Only)	55		55		40		ns	
TRHRL	End of $\overline{RD}$ to next $\overline{RD}$ End of $\overline{INTA}$ to next $\overline{INTA}$ within an $\overline{INTA}$ sequence only	160		160		160		ns	
TWHWL	End of $\overline{WR}$ to next $\overline{WR}$	190		190		190		ns	

A.C. CHARACTERISTICS (Continued)

Symbol	Parameter	8259A-8		8259A		8259A-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
*TCHCL	End of Command to next Command (Not same command type) End of $\overline{INTA}$ sequence to next $\overline{INTA}$ sequence.	500		500		500		ns	

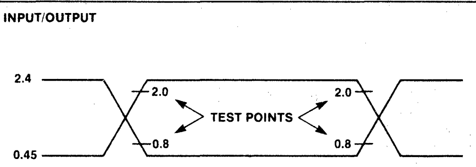
*Worst case timing for TCHCL in an actual microprocessor system is typically much greater than 500 ns (i.e. 8085A = 1.6μs, 8085A-2 = 1μs, 8086 = 1μs, 8086-2 = 625 ns)

NOTE: This is the low time required to clear the input latch in the edge triggered mode.

TIMING RESPONSES

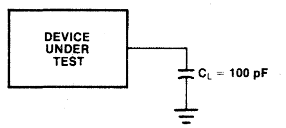
Symbol	Parameter	8259A-8		8259A		8259A-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TRLDV	Data Valid from $\overline{RD}$/$\overline{INTA}$↓		300		200		120	ns	C of Data Bus= 100 pF
TRHDZ	Data Float after $\overline{RD}$/$\overline{INTA}$↑	10	200	10	100	10	85	ns	C of Data Bus Max text C = 100 pF Min. test C = 15 pF
TJHIH	Interrupt Output Delay		400		350		300	ns	
TIALCV	Cascade Valid from First $\overline{INTA}$↓ (Master Only)		565		565		360	ns	C_{INT} = 100 pF
TRLEL	Enable Active from $\overline{RD}$↓ or $\overline{INTA}$↓		160		125		100	ns	$C_{CASCADE}$ = 100 pF
TRHEH	Enable Inactive from $\overline{RD}$↑ or $\overline{INTA}$↑		325		150		150	ns	
TAHDV	Data Valid from Stable Address		350		200		200	ns	
TCVDV	Cascade Valid to Valid Data		300		300		200	ns	

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0.8V FOR A LOGIC "0."

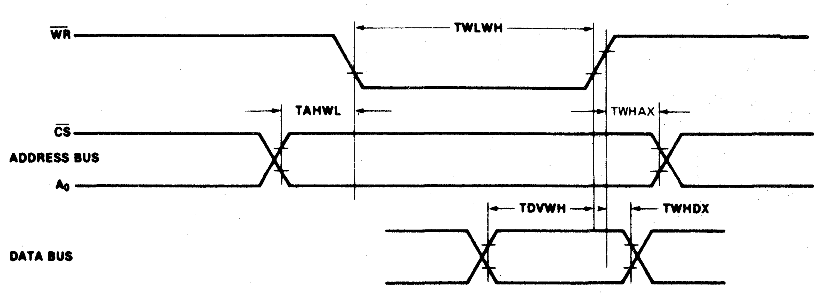
A.C. TESTING LOAD CIRCUIT



C_L = 100 pF
C_L INCLUDES JIG CAPACITANCE

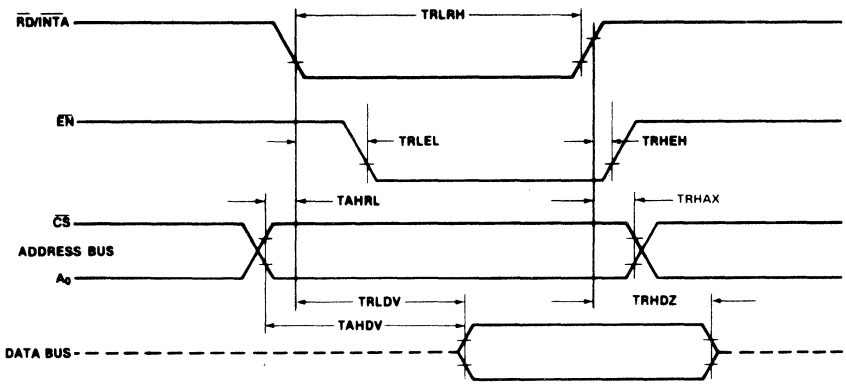
WAVEFORMS

WRITE

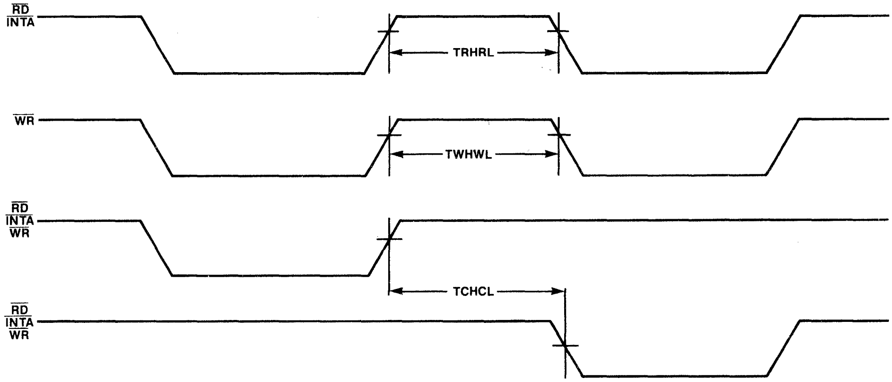


WAVEFORMS (Continued)

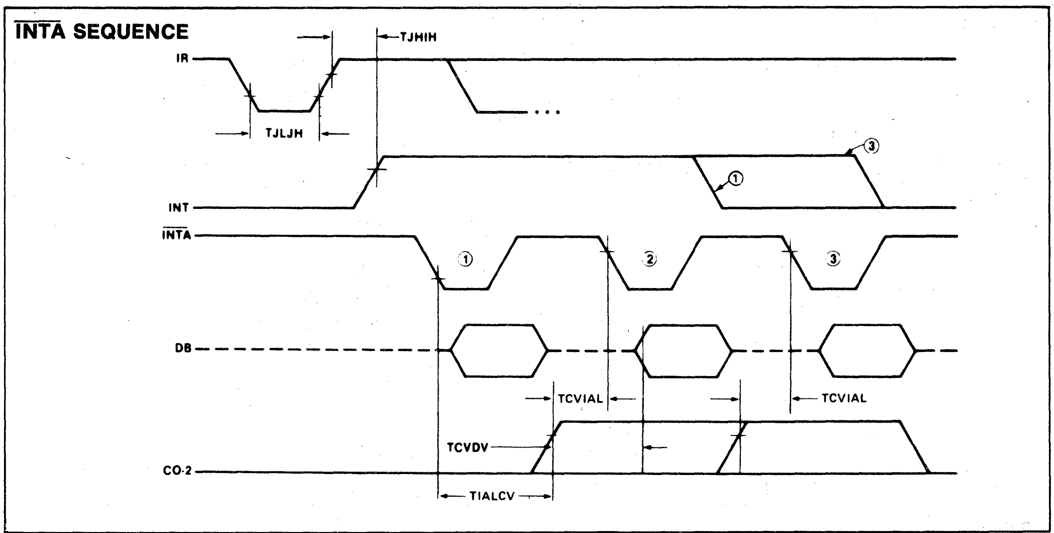
READ/INTA



OTHER TIMING



WAVEFORMS (Continued)



NOTES: Interrupt output must remain HIGH at least until leading edge of first INTA.

1. Cycle 1 in iAPX 86, iAPX 88 systems, the Data Bus is not active.

82C59A-2
CHMOS Programmable Interrupt Controller

- Pin Compatible with NMOS 8259A-2
- Eight-Level Priority Controller
- Expandable to 64 levels
- Programmable Interrupt Modes
- Low Standby Power—10 μA
- Individual Request Mask Capability
- 80C86/88 and 8080/85/86/88 Compatible
- Fully Static Design
- Single 5V Power Supply
- Will Be Available in 28-Lead Plastic DIP and 28-Lead PLCC Packages
  (See Packaging Spec., Order #231369)

The Intel 82C59A-2 is a high performance CHMOS Version of the NMOS 8259A-2 Priority Interrupt Controller. The 82C59A is designed to relieve the system CPU from the task of polling in a multi-level priority interrupt system. The high speed and industry standard configuration of the 82C59A-2, make it compatible with microprocessors such as the 80C86/88, 8086/88 and 8080/85.

The 82C59A-2 can handle up to 8 vectored priority interrupts for the CPU and is cascadable to 64 without additional circuitry. It is designed to minimize the software and real time overhead in handling multi-level priority interrupts. Two modes of operation make the 82C59A-2 optimal for a variety of system requirements. Static CHMOS circuit design, requiring no clock input, insures low operating power. It is packaged in a 28-pin plastic DIP.

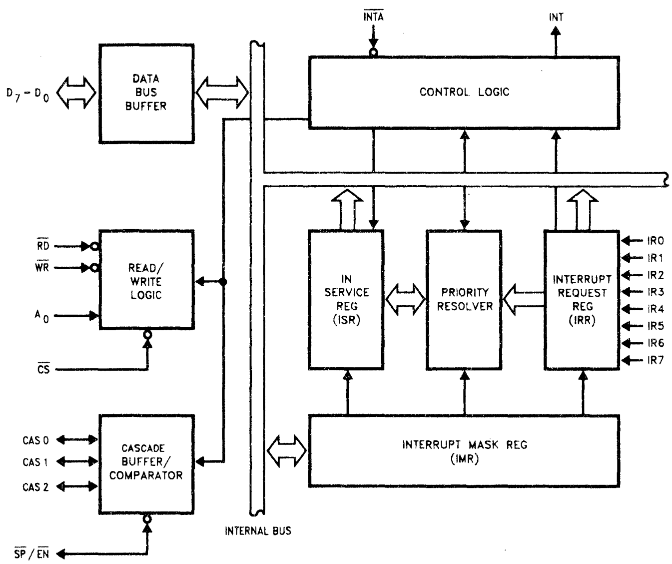


Figure 1. Block Diagram

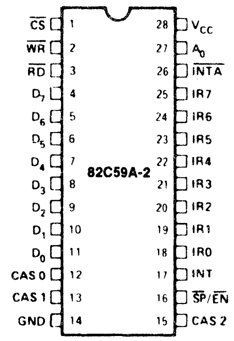


Figure 2a. 28-Lead DIP Configuration

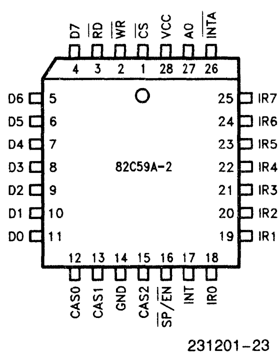


Figure 2b. 28-Lead PLCC Configuration

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}	28	I	**SUPPLY:** +5V Supply.
GND	14	I	**GROUND.**
$\overline{CS}$	1	I	**CHIP SELECT:** A low on this pin enables $\overline{RD}$ and $\overline{WR}$ communication between the CPU and the 82C59A-2. INTA functions are independent of CS.
$\overline{WR}$	2	I	**WRITE:** A low on this pin when $\overline{CS}$ is low enables the 82C59A-2 to accept command words from the CPU.
$\overline{RD}$	3	I	**READ:** A low on this pin when $\overline{CS}$ is low enables the 82C59A-2 to release status onto the data bus for the CPU.
D_7-D_0	4–11	I/O	**BIDIRECTIONAL DATA BUS:** Control, status and interrupt-vector information is transferred via this bus.
CAS_0-CAS_2	12, 13, 15	I/O	**CASCADE LINES:** The CAS lines form a private 82C59A-2 bus to control a multiple 82C59A-2 structure. These pins are outputs for a master 82C59A-2 and inputs for a slave 82C59A-2.
$\overline{SP}/\overline{EN}$	16	I/O	**SLAVE PROGRAM/ENABLE BUFFER:** This is a dual function pin. When in the Buffered Mode it can be used as an output to control buffer transceivers (EN). When not in the buffered mode it is used as an input to designate a master (SP = 1) or slave (SP = 0).
INT	17	O	**INTERRUPT:** This pin goes high whenever a valid interrupt request is asserted. It is used to interrupt the CPU, thus it is connected to the CPU's interrupt pin.
IR_0-IR_7	18–25	I	**INTERRUPT REQUESTS:** Asynchronous inputs. An interrupt request is executed by raising an IR input (low to high), and holding it high until it is acknowledged (Edge Triggered Mode), or just by a high level on an IR input (Level Triggered Mode). Internal pull-up resistors are implemented on IR0–7.
$\overline{INTA}$	26	I	**INTERRUPT ACKNOWLEDGE:** This pin is used to enable 82C59A-2 interrupt-vector data onto the data bus by a sequence of interrupt acknowledge pulses issued by the CPU.
A_0	27	I	**AO ADDRESS LINE:** This pin acts in conjunction with the $\overline{CS}$, $\overline{WR}$, and $\overline{RD}$ pins. It is used by the 82C59A-2 to decipher various Command Words the CPU writes and status the CPU wishes to read. It is typically connected to the CPU A0 address line (A1 for 80C86, 80C88).

FUNCTIONAL DESCRIPTION

Interrupts in Microcomputer Systems

Microcomputer system design requires that I/O devices such as keyboards, displays, sensors and other components receive servicing in an efficient manner so that large amounts of the total system tasks can be assumed by the microcomputer with little or no effect on throughput.

The most common method of servicing such devices is the *Polled* approach. This is where the processor must test each device in sequence and in effect "ask" each one if it needs servicing. It is easy to see that a large portion of the main program is looping through this continuous polling cycle and that such a method would have a serious, detrimental effect on system throughput, thus limiting the tasks that could be assumed by the microcomputer and reducing the cost effectiveness of using such devices.

A more desirable method would be one that would allow the microprocessor to be executing its main program and only stop to service peripheral devices when it is told to do so by the device itself. In effect, the method would provide an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed and fetch a new routine that will service the requesting device. Once this servicing is complete, however, the processor would resume exactly where it left off.

This method is called *Interrupt*. It is easy to see that system throughput would drastically increase, and thus more tasks could be assumed by the microcomputer to further enhance its cost effectiveness.

The Programmable Interrupt Controller (PIC) functions as an overall manager in an Interrupt-Driven system environment. It accepts requests from the peripheral equipment, determines which of the incoming requests is of the highest importance (priority), ascertains whether the incoming request has a higher priority value than the level currently being serviced, and issues an interrupt to the CPU based on this determination.

Each peripheral device or structure usually has a special program or "routine" that is associated with its specific functional or operational requirements; this is referred to as a "service routine". The PIC, after issuing an Interrupt to the CPU, must somehow input information into the CPU that can "point" the Program Counter to the service routine associated with the requesting device. This "pointer" is an address in a vectoring table and will often be referred to, in this document, as vectoring data.

The 82C59A-2

The 82C59A-2 is a device specifically designed for use in real time, interrupt driven microcomputer sys-

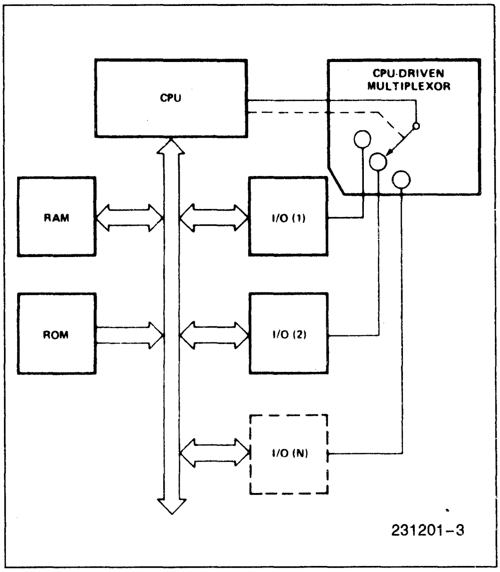


Figure 3a. Polled Method

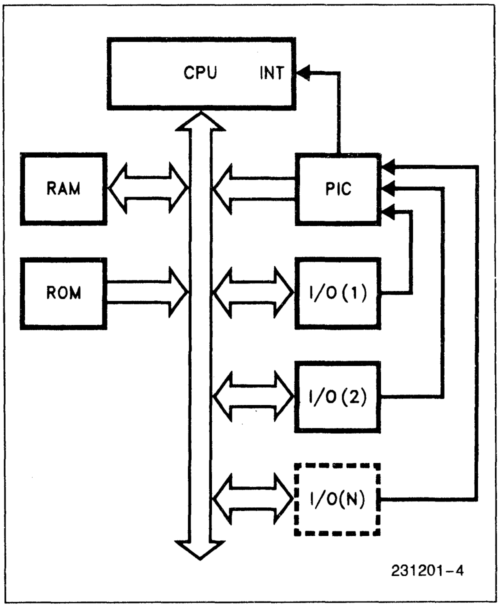


Figure 3b. Interrupt Method

tems. It manages eight levels or requests and has built-in features for expandability to other 82C59A-2's (up to 64 levels). It is programmed by the system's software as an I/O peripheral. A selection of priority modes is available to the programmer so that the manner in which the requests are processed by the 82C59A-2 can be configured to match system requirements. The priority modes can be changed or reconfigured dynamically at any time during the main program. This means that the complete interrupt structure can be defined as required, based on the total system environment.

INTERRUPT REQUEST REGISTER (IRR) AND IN-SERVICE REGISTER (ISR)

The interrupts at the IR input lines are handled by two registers in cascade, the Interrupt Request Register (IRR) and the In-Service Register (ISR). The IRR is used to store all the interrupt levels which are requesting service; and the ISR is used to store all the interrupt levels which are being serviced.

PRIORITY RESOLVER

This logic block determines the priorities of the bits set in the IRR. The highest priority is selected and strobed into the corresponding bit of the ISR during $\overline{\text{INTA}}$ pulse.

INTERRUPT MASK REGISTER (IMR)

The IMR stores the bits which mask the interrupt lines to be masked. The IMR operates on the IRR. Masking of a higher priority input will not affect the interrupt request lines of lower priority.

INT (INTERRUPT)

This output goes directly to the CPU interrupt input. The V_{OH} level on this line is designed to be fully compatible with the 8080A, 8085A, 80C88 and 80C86 input levels.

$\overline{\text{INTA}}$ (INTERRUPT ACKNOWLEDGE)

$\overline{\text{INTA}}$ pulses will cause the 82C59A-2 to release vectoring information onto the data bus. The format of this data depends on the system mode (μPM) of the 82C59A-2.

DATA BUS BUFFER

This 3-state, bidirectional 8-bit buffer is used to interface the 82C59A-2 to the system Data Bus. Control words and status information are transferred through the Data Bus Buffer.

READ/WRITE CONTROL LOGIC

The function of this block is to accept OUTput commands from the CPU. It contains the Initialization Command Word (ICW) registers and Operation Command Word (OCW) registers which store the various control formats for device operation. This function block also allows the status of the 82C59A-2 to be transferred onto the Data Bus.

$\overline{\text{CS}}$ (CHIP SELECT)

A LOW on this input enables the 82C59A-2. No reading or writing of the chip will occur unless the device is selected.

$\overline{\text{WR}}$ (WRITE)

A LOW on this input enables the CPU to write control words (ICWs and OCWs) to the 82C59A-2.

$\overline{\text{RD}}$ (READ)

A LOW on this input enables the 82C59A-2 to send the status of the Interrupt Request Register (IRR), In Service Register (ISR), the Interrupt Mask Register (IMR), or the Interrupt level onto the Data Bus.

A_0

This input signal is used in conjunction with $\overline{\text{WR}}$ and $\overline{\text{RD}}$ signals to write commands into the various command registers, as well as reading the various status registers of the chip. This line can be tied directly to one of the address lines.

THE CASCADE BUFFER/COMPARATOR

This function block stores and compares the IDs of all 82C59A-2's used in the system. The associated three I/O pins (CAS0–2) are outputs when the 82C59A-2 is used as a master and are inputs when the 82C59A-2 is used as a slave. As a master, the 82C59A-2 sends the ID of the interrupting slave device onto the CAS0–2 lines. The slave thus selected will send its preprogrammed subroutine address onto the Data Bus during the next one or two consecutive $\overline{\text{INTA}}$ pulses. (See section "Cascading the 82C59A-2".)

INTERRUPT SEQUENCE

The powerful features of the 82C59A-2 in a microcomputer system are its programmability and the interrupt routine addressing capability. The latter allows direct or indirect jumping to the specific interrupt routine requested without any polling of the interrupting devices. The normal sequence of events during an interrupt depends on the type of CPU being used.

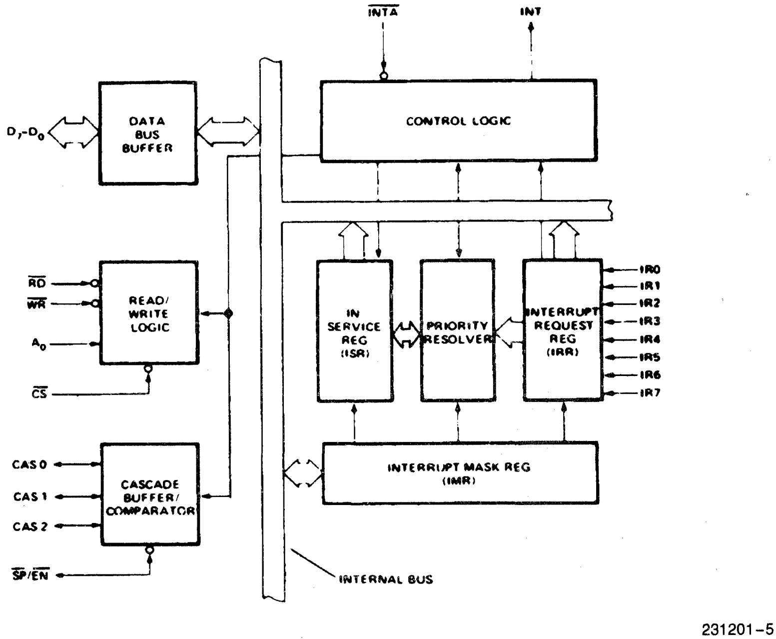


Figure 4. 82C59A-2 Block Diagram

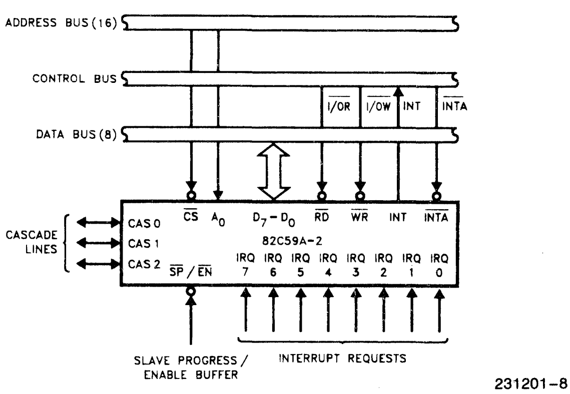


Figure 5. 82C59A-2 Interface to Standard System Bus

The events occur as follows in an MCS-80/85 system:

1. One or more of the INTERRUPT REQUEST Lines (IR7–0) are raised high, setting the corresponding IRR bit(s).
2. The 82C59A-2 evaluates these requests, and sends an INT to the CPU, if appropriate.
3. The CPU acknowledges the INT and responds with an $\overline{INTA}$ pulse.
4. Upon receiving an $\overline{INTA}$ from the CPU group, the highest priority ISR bit is set, and the corresponding IRR bit is reset. The 82C59A-2 will also release a CALL instruction code (11001101) onto the 8-bit Data Bus through its D7–0 pins.
5. This CALL instruction will initiate two more $\overline{INTA}$ pulses to be sent to the 82C59A-2 from the CPU group.
6. These two $\overline{INTA}$ pulses allow the 82C59A-2 to release its preprogrammed subroutine address onto the Data Bus. The lower 8-bit address is released at the first $\overline{INTA}$ pulse and the higher 8-bit address is released at the second $\overline{INTA}$ pulse.
7. This completes the 3-byte CALL instruction released by the 82C59A-2. In the AEOI mode the ISR bit is reset at the end of the third $\overline{INTA}$ pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt sequence.

The events occurring in an 80C86 system are the same until step 4.

4. Upon receiving an $\overline{INTA}$ from the CPU group, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 82C59A-2 does not drive the Data Bus during this cycle.
5. The 80C86 will initiate a second $\overline{INTA}$ pulse. During this pulse, the 82C59A-2 releases an 8-bit pointer onto the Data Bus where it is read by the CPU.
6. This completes the interrupt cycle. In the AEOI mode the ISR bit is reset at the end of the second $\overline{INTA}$ pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt subroutine.

If no interrupt is present at step 4 of either sequence (i.e., the request was too short in duration) the 82C59A-2 will issue an interrupt level 7. Both the vectoring bytes and the CAS lines will look like an interrupt level 7 was requested.

INTERRUPT SEQUENCE OUTPUTS

MCS®-80, MCS-85

This sequence is timed by three $\overline{INTA}$ pulses. During the first $\overline{INTA}$ pulse the CALL opcode is enabled onto the data bus.

Content of First Interrupt Vector Byte

	D7	D6	D5	D4	D3	D2	D1	D0
CALL CODE	1	1	0	0	1	1	0	1

During the second $\overline{INTA}$ pulse the lower address of the appropriate service routine is enabled onto the data bus. When Interval = 4 bits A_5–A_7 are programmed, while A_0–A_4 are automatically inserted by the 82C59A-2. When Interval = 8 only A_6 and A_7 are programmed, while A_0–A_5 are automatically inserted.

Content of Second Interrupt Vector Byte

IR	Interval = 4							
	D7	D6	D5	D4	D3	D2	D1	D0
7	A7	A6	A5	1	1	1	0	0
6	A7	A6	A5	1	1	0	0	0
5	A7	A6	A5	1	0	1	0	0
4	A7	A6	A5	1	0	0	0	0
3	A7	A6	A5	0	1	1	0	0
2	A7	A6	A5	0	1	0	0	0
1	A7	A6	A5	0	0	1	0	0
0	A7	A6	A5	0	0	0	0	0

IR	Interval = 8							
	D7	D6	D5	D4	D3	D2	D1	D0
7	A7	A6	1	1	1	0	0	0
6	A7	A6	1	1	0	0	0	0
5	A7	A6	1	0	1	0	0	0
4	A7	A6	1	0	0	0	0	0
3	A7	A6	0	1	1	0	0	0
2	A7	A6	0	1	0	0	0	0
1	A7	A6	0	0	1	0	0	0
0	A7	A6	0	0	0	0	0	0

During the third INTA pulse the higher address of the appropriate service routine, which was programmed as byte 2 of the initialization sequence (A_8 – A_{15}), is enabled onto the bus.

Content of Third Interrupt Vector Byte

D7	D6	D5	D4	D3	D2	D1	D0
A15	A14	A13	A12	A11	A10	A9	A8

80C86, 80C88

80C86, 80C88 mode is similar to MCS-80 mode except that only two Interrupt Acknowledge cycles are issued by the processor and no CALL opcode is sent to the processor. The first interrupt acknowledge cycle is similar to that of MCS-80, 85 systems in that the 82C59A-2 uses it to internally freeze the state of the interrupts for priority resolution and as a master it issues the interrupt code on the cascade lines at the end of the INTA pulse. On this first cycle it does not issue any data to the processor and leaves its data bus buffers disabled. On the second interrupt acknowledge cycle in 80C86, 80C88 mode the master (or slave if so programmed) will send a byte of data to the processor with the acknowledged interrupt code composed as follows (note the state of the ADI mode control is ignored and A_5–A_{11} are unused in 80C86, 80C88 mode):

Content of Interrupt Vector Byte for 80C86, 80C88 System Mode

	D7	D6	D5	D4	D3	D2	D1	D0
IR7	T7	T6	T5	T4	T3	1	1	1
IR6	T7	T6	T5	T4	T3	1	1	0
IR5	T7	T6	T5	T4	T3	1	0	1
IR4	T7	T6	T5	T4	T3	1	0	0
IR3	T7	T6	T5	T4	T3	0	1	1
IR2	T7	T6	T5	T4	T3	0	1	0
IR1	T7	T6	T5	T4	T3	0	0	1
IR0	T7	T6	T5	T4	T3	0	0	0

PROGRAMMING THE 82C59A-2

The 82C59A-2 accepts two types of command words generated by the CPU:

1. *Initialization Command Words (ICWs)*: Before normal operation can begin, each 82C59A-2 in the system must be brought to a starting point — by a sequence of 2 to 4 bytes timed by $\overline{WR}$ pulses.
2. *Operation Command Words (OCWs)*: These are the command words which command the 82C59A-2 to operate in various interrupt modes. These modes are:
   a. Fully nested mode
   b. Rotating priority mode
   c. Special mask mode
   d. Polled mode

The OCWs can be written into the 82C59A-2 anytime after initialization.

INITIALIZATION COMMAND WORDS (ICWS)

GENERAL

Whenever a command is issued with A0 = 0 and D4 = 1, this is interpreted as Initialization Command Word 1 (ICW1). ICW1 starts the initialization sequence during which the following automatically occur.

a. The edge sense circuit is reset, which means that following initialization, an interrupt request (IR) input must make a low-to-high transition to generate an interrupt.

b. The Interrupt Mask Register is cleared.

c. IR7 input is assigned priority 7.

d. The slave mode address is set to 7.

e. Special Mask Mode is cleared and Status Read is set to IRR.

f. If IC4 = 0, then all functions selected in ICW4 are set to zero. (Non-Buffered mode*, no Auto-EOI, MCS-80, 85 system).

***NOTE:**
Master/Slave in ICW4 is only used in the buffered mode.

INITIALIZATION COMMAND WORDS 1 AND 2 (ICW1, ICW2)

A_5–A_{15}: *Page starting address of service routines.* In an MCS 80/85 system, the 8 request levels will generate CALLs to 8 locations equally spaced in memory. These can be programmed to be spaced at intervals of 4 or 8 memory locations, thus the 8 routines will occupy a page of 32 or 64 bytes, respectively.

The address format is 2 bytes long (A_0–A_{15}). When the routine interval is 4, A_0–A_4 are automatically inserted by the 82C59A-2, while A_5–A_{15} are programmed externally. When the routine interval is 8, A_0–A_5 are automatically inserted by the 82C59A-2, while A_6–A_{15} are programmed externally.

The 8-byte interval will maintain compatibility with current software, while the 4-byte interval is best for a compact jump table.

In an 80C86, 80C88 system A_{15}–A_{11} are inserted in the five most significant bits of the vectoring byte and the 82C59A-2 sets the three least significant bits according to the interrupt level. $A_{10}-A_5$ are ignored and ADI (Address Interval) has no effect:

LTIM: If LTIM = 1, then the 82C59A-2 will operate in the level interrupt mode. Edge detect logic on the interrupt inputs will be disabled.

ADI: CALL address interval. ADI = 1 then interval = 4; ADI = 0 then interval = 8.

SNGL: Single. Means that this is the only 82C59A-2 in the system. If SNGL = 1 no ICW3 will be issued.

IC4: If this bit is set — ICW4 has to be read. If ICW4 is not needed, set IC4 = 0.

INITIALIZATION COMMAND WORD 3 (ICW3)

This word is read only when there is more than one 82C59A-2 in the system and cascading is used, in which case SNGL = 0. It will load the 8-bit slave register. The functions of this register are:

a. In the master mode (either when SP = 1, or in buffered mode when M/S = 1 in ICW4) a "1" is set for each slave in the system. The master then will release byte 1 of the call sequence (for MCS-80/85 system) and will enable the corresponding slave to release bytes 2 and 3 (for 80C86, 80C88 only byte 2) through the cascade lines.

b. In the slave mode (either when $\overline{SP}$ = 0, or if BUF = 1 and M/S = 0 in ICW4) bits 2-0 identify the slave. The slave compares its cascade input with these bits and, if they are equal, bytes 2 and 3 of the call sequence (or just byte 2 for 80C86, 80C88 are released by it on the Data Bus.

INITIALIZATION COMMAND WORD 4 (ICW4)

SFNM: If SFNM = 1 the special fully nested mode is programmed.

BUF: If BUF = 1 the buffered mode is programmed. In buffered mode $\overline{SP}/\overline{EN}$ becomes an enable output and the master/slave determination is by M/S.

M/S: If buffered mode is selected: M/S = 1 means the 82C59A-2 is programmed to be a master, M/S = 0 means the 82C59A-2 is programmed to be a slave. If BUF = 0, M/S has no function.

AEOI: If AEOI = 1 the automatic end of interrupt mode is programmed.

μPM: Microprocessor mode: μPM = 0 sets the 82C59A-2 for MCS-80, 85 system operation, μPM = 1 sets the 82C59A-2 for 80C86 system operation.

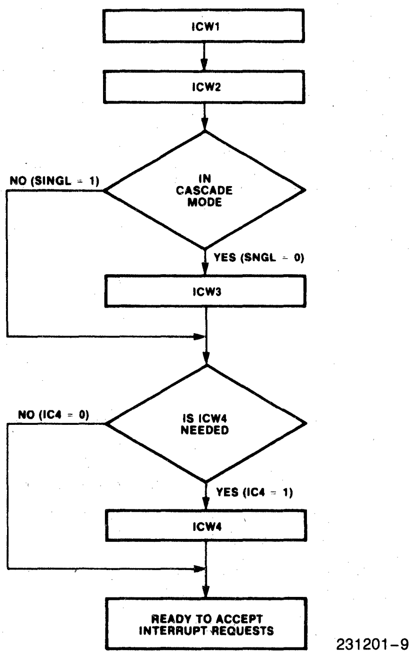
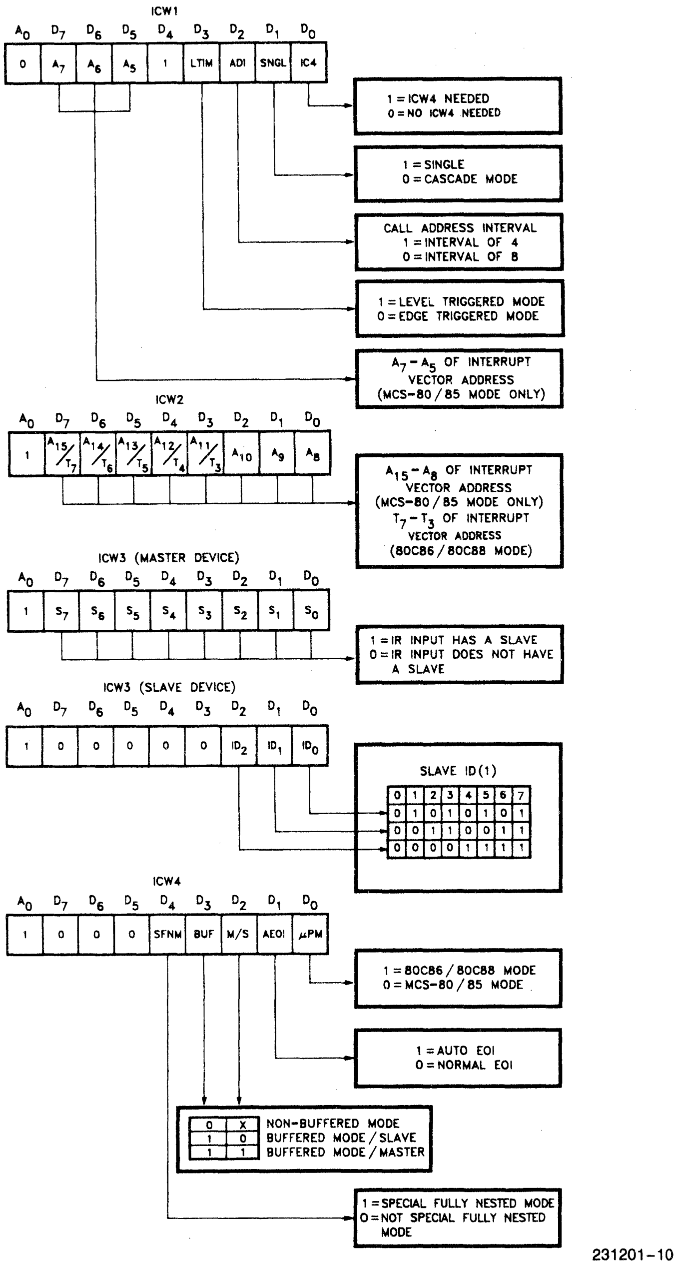


Figure 6. Initialization Sequence



NOTE:
Slave ID is equal to the corresponding master IR input.

Figure 7. Initialization Command Word Format

OPERATION COMMAND WORDS (OCWs)

After the initialization Command Words (ICWs) are programmed into the 82C59A-2, the chip is ready to accept interrupt requests at its input lines. However, during the 82C59A-2 operation, a selection of algorithms can command the 82C59A-2 to operate in various modes through the Operation Command Words (OCWs).

OPERATION CONTROL WORDS (OCWs)

OCW1

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	M7	M6	M5	M4	M3	M2	M1	M0

OCW2

A0	D7	D6	D5	D4	D3	D2	D1	D0
0	R	SL	EOI	0	0	L2	L1	L0

OCW3

A0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	ESMM	SMM	0	1	P	RR	RIS

OPERATION CONTROL WORD 1 (OCW1)

OCW1 sets and clears the mask bits in the interrupt Mask Register (IMR). $M_7 - M_0$ represent the eight mask bits. M = 1 indicates the channel is masked (inhibited), M = 0 indicates the channel is enabled.

OPERATION CONTROL WORD 2 (OCW2)

R, SL, EOI — These three bits control the Rotate and End of Interrupt modes and combinations of the two. A chart of these combinations can be found on the Operation Command Word Format.

L_2, L_1, L_0—These bits determine the interrupt level acted upon when the SL bit is active.

OPERATION CONTROL WORD 3 (OCW3)

ESMM — Enable Special Mask Mode. When this bit is set to 1 it enables the SMM bit to set or reset the Special Mask Mode. When ESMM = 0 the SMM bit becomes a "don't care".

SMM — Special Mask Mode. If ESMM = 1 and SMM = 1 the 82C59A-2 will enter Special Mask Mode. If ESMM = 1 and SMM = 0 the 82C59A-2 will revert to normal mask mode. When ESMM = 0, SMM has no effect.

FULLY NESTED MODE

This mode is entered after initialization unless another mode is programmed. The interrupt requests are ordered in priority form 0 through 7 (0 highest). When an interrupt is acknowledged the highest priority request is determined and its vector placed on the bus. Additionally, a bit of the Interrupt Service register (ISO-7) is set. This bit remains set until the microprocessor issues an End of Interrupt (EOI) command immediately before returning from the service routine, or if AEOI (Automatic. End of Interrupt) bit is set, until the trailing edge of the last INTA. While the IS bit is set, all further interrupts of the same or lower priority are inhibited, while higher levels will generate an interrupt (which will be acknowledged only if the microprocessor internal interrupt enable flip-flop has been re-enabled through software).

After the initialization sequence, IR0 has the highest priority and IR7 the lowest. Priorities can be changed, as will be explained, in the rotating priority mode.

END OF INTERRUPT (EOI)

The In Service (IS) bit can be reset either automatically following the trailing edge of the last in sequence $\overline{INTA}$ pulse (when AEOI bit in ICW4 is set) or by a command word that must be issued to the 82C59A-2 before returning from a service routine (EOI command). An EOI command must be issued twice if in the Cascade mode, once for the master and once for the corresponding slave.

There are two forms of EOI command: Specific and Non-Specific. When the 82C59A-2 is operated in modes which preserve the fully nested structure, it can determine which IS bit to reset on EOI. When a Non-Specific EOI command is issued the 82C59A-2 will automatically reset the highest IS bit of those that are set, since in the fully nested mode the highest IS level was necessarily the last level acknowledged and serviced. A non-specific EOI can be issued with OCW2 (EOI = 1, SL = 0, R = 0).

When a mode is used which may disturb the fully nested structure, the 82C59A-2 may no longer be able to determine the last level acknowledged. In this case a Specific End of Interrupt must be issued which includes as part of the command the IS level to be reset. A specific EOI can be issued with OCW2 (EOI = 1, SL = 1, R = 0, and L0-L2 is the binary level of the IS bit to be reset).

It should be noted that an IS bit that is masked by an IMR bit will not be cleared by a non-specific EOI if the 82C59A-2 is in the Special Mask Mode.

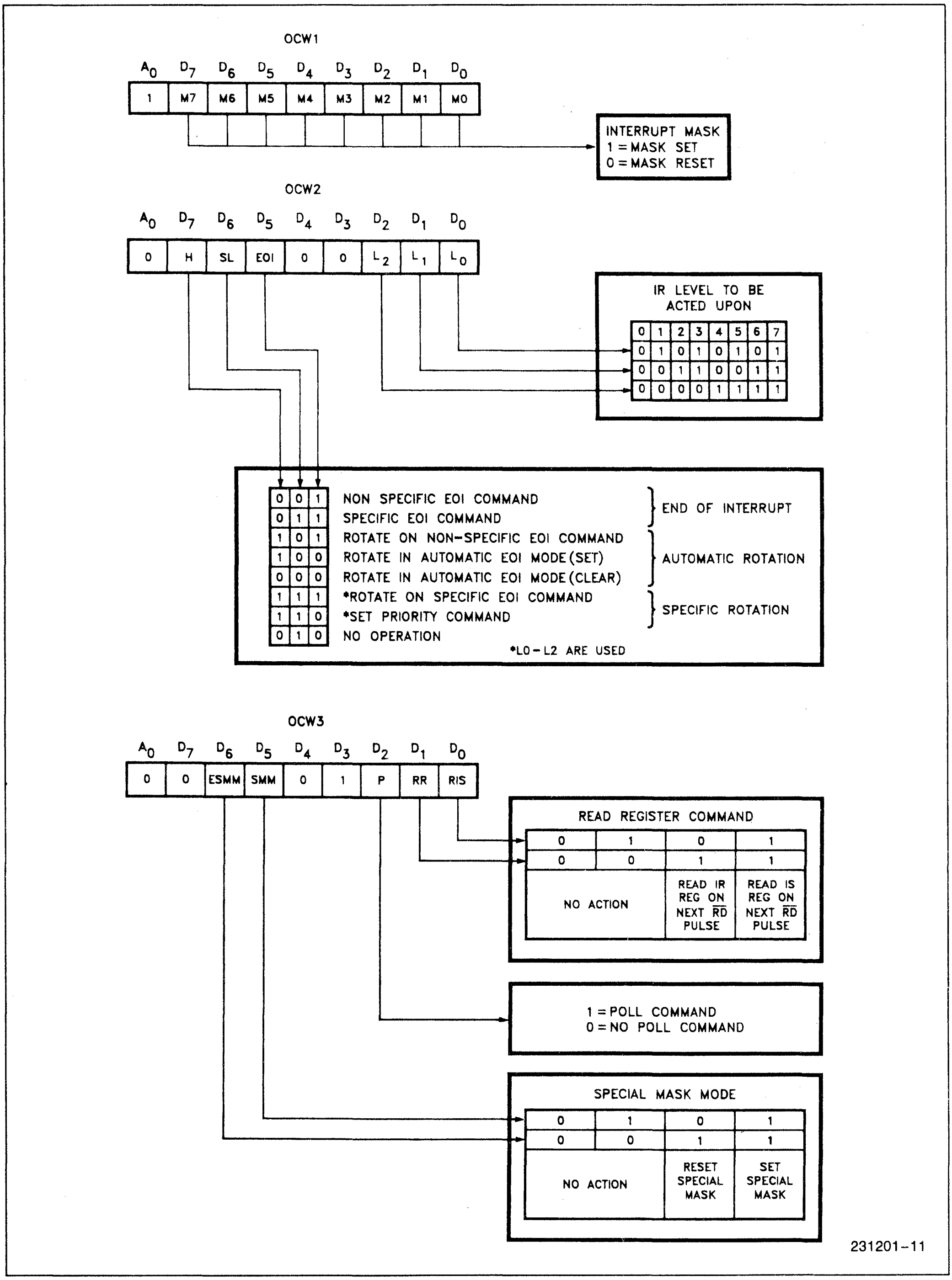


Figure 8. Operation Command Word Format

AUTOMATIC END OF INTERRUPT (AEOI) MODE

If AEOI = 1 in ICW4, then the 82C59A-2 will operate in AEOI mode continuously until reprogrammed by ICW4. In this mode the 82C59A-2 will automatically perform a non-specific EOI operation at the trailing edge of the last interrupt acknowledge pulse (third pulse in MCS-80/85, second in 80C86/88). Note that from a system standpoint, this mode should be used only when a nested multilevel interrupt structure is not required within a single 82C59A.

The AEOI mode can only be used in a master 82C59A and not a slave.

AUTOMATIC ROTATION

(Equal Priority Devices)

In some applications there are a number of interrupting devices of equal priority. In this mode a device, after being serviced, receives the lowest priority, so a device requesting an interrupt will have to wait, in the worst case until each of 7 other devices are serviced at most *once*. For example, if the priority and "in service" status is:

Before Rotate (IR4 the highest priority requiring service)

	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
"IS" Status	0	1	0	1	0	0	0	0

	Lowest Priority ↓							Highest Priority ↓
Priority Status	7	6	5	4	3	2	1	0

After Rotate (IR4 was serviced, all other priorities rotated correspondingly)

	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
"IS" Status	0	1	0	0	0	0	0	0

			Highest Priority ↓	Lowest Priority ↓				
Priority Status	2	1	0	7	6	5	4	3

There are two ways to accomplish Automatic Rotation using OCW2, the Rotation on Non-Specific EOI Command (R = 1, SL = 0, EOI = 1) and the Rotate in Automatic EOI Mode which is set by (R = 1, SL = 0, EOI = 0) and cleared by (R = 0, SL = 0, EOI = 0).

SPECIFIC ROTATION

(Specific Priority)

The programmer can change priorities by programming the bottom priority and thus fixing all other priorities; i.e., if IR5 is programmed as the bottom priority device, then IR6 will have the highest one.

The Set Priority command is issued in OCW2 where: R = 1, SL = 1; LO–L2 is the binary priority level code of the bottom priority device.

Observe that in this mode internal status is updated by software control during OCW2. However, it is independent of the End of Interrupt (EOI) command (also executed by OCW2). Priority changes can be executed during an EOI command by using the Rotate on Specific EOI command in OCW2 (R = 1, SL = 1, EOI = 1 and LO–L2 = IR level to receive bottom priority).

INTERRUPT MASKS

Each Interrupt Request input can be masked individually by the Interrupt Mask Register (IMR) programmed through OCW1. Each bit in the IMR masks one interrupt channel if it is set (1). Bit 0 masks IR0, Bit 1 masks IR1 and so forth. Masking an IR channel does not affect the other channels operation.

SPECIAL MASK MODE

Some applications may require an interrupt service routine to dynamically alter the system priority structure during its execution under software control. For example, the routine may wish to inhibit lower priority requests for a portion of its execution but enable some of them for another portion.

The difficulty here is that if an interrupt Request is acknowledged and an End of Interrupt command did not reset its IS bit (i.e., while executing a service routine), the 82C59A-2 would have inhibited all lower priority requests with no easy way for the routine to enable them.

That is where the Special Mask Mode comes in. In the special Mask Mode, when a mask bit is set in OCW1, it inhibits further interrupts at that level *and enables* interrupts from *all other* levels (lower as well as higher) that are not masked.

Thus, any interrupts may be selectivity enabled by loading the mask register.

The special Mask Mode is set by OCW3 where: SSMM = 1, SMM = 1, and cleared where SSMM = 1, SMM = 0.

POLL COMMAND

In this mode the INT output is not used or the microprocessor internal Interrupt Enable flip-flop is reset, disabling its interrupt input. Service to devices is achieved by software using a Poll command.

The Poll command is issued by setting P = "1" in OCW3. The 82C59A-2 treats the next $\overline{RD}$ pulse to the 82C59A-2 (i.e., $\overline{RD}$ = 0, $\overline{CS}$ = 0) as an interrupt acknowledge, sets the appropriate IS bit if there is a request, and reads the priority level. Interrupt is frozen from $\overline{WR}$ to $\overline{RD}$.

The word enabled onto the data bus during $\overline{RD}$ is:

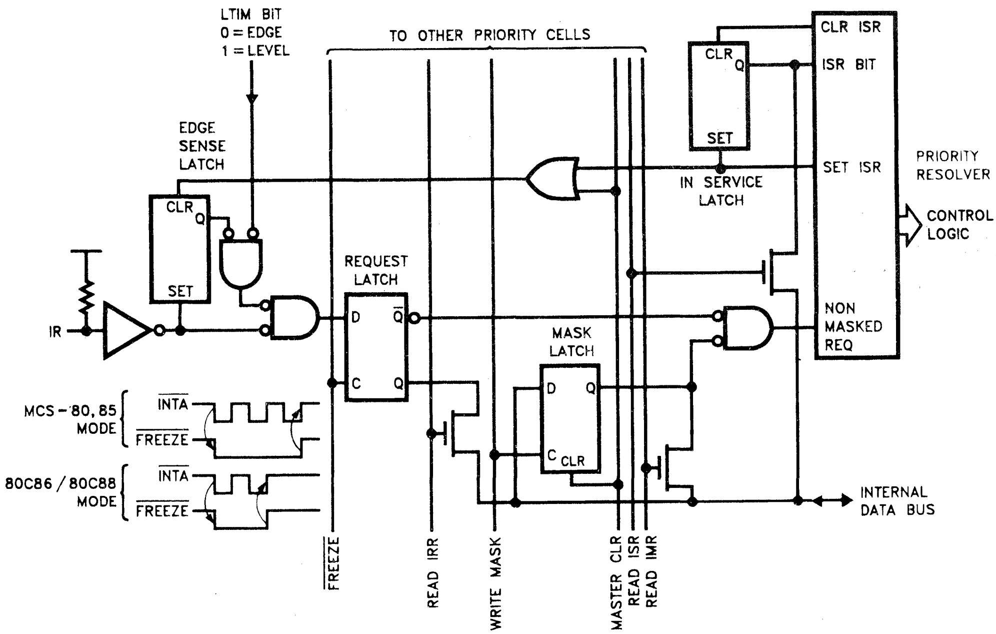
D7	D6	D5	D4	D3	D2	D1	D0
I	—	—	—	—	W2	W1	W0

WO–W2:

Binary code of the highest priority level requesting service.

1: Equal to a "1" if there is an interrupt.

This mode is useful if there is a routine command common to several levels so that the $\overline{INTA}$ sequence is not needed (saves ROM space). Another application is to use the poll mode to expand the number of priority levels to more than 64.



231201-12

NOTES:
1. Master Clear active only during ICW1
2. Freeze/ is active during $\overline{INTA}$/and poll sequences only
3. Truth Table for D-Latch

C	D	Q	OPERATION
1	Di	Di	FOLLOW
0	X	Qn-1	HOLD

Figure 9. Priority Cell—Simplified Logic Diagram

READING THE 82C59A-2 STATUS

The input status of several internal registers can be read to update the user information on the system. The following registers can be read via OCW3 (IRR and ISR or OCW1 [IMR]).

Interrupt Request Register (IRR): 8-bit register which contains the levels requesting an interrupt to be acknowledged. The highest request level is reset from the IRR when an interrupt is acknowledged. (Not affected by IMR).

In-Service Register (ISR): 8-bit register which contains the priority levels that are being serviced. The ISR is updated when an End of Interrupt Command is issued.

Interrupt Mask Register: 8-bit register which contains the interrupt request lines which are masked.

The IRR can be read when, prior to the RD pulse, a Read Register Command is issued with OCW3 (RR = 1, RIS = 0.)

The ISR can be read when, prior to the RD pulse, a Read Register Command is issued with OCW3 (RR = 1, RIS = 1):

There is no need to write an OCW3 before every status read operation, as long as the status read corresponds with the previous one; i.e., the 82C59A-2 "remembers" whether the IRR or ISR has been previously selected by the OCW3. This is not true when poll is used.

After initialization the 82C59A-2 is set to IRR.

For reading the IMR, no OCW3 is needed. The output data bus will contain the IMR whenever $\overline{RD}$ is active and AO = 1 (OCW1).

Polling overrides status read when P = 1, RR = 1 in OCW3.

EDGE AND LEVEL TRIGGERED MODES

This mode is programmed using bit 3 in ICW1.

If LTIM = '0', an interrupt request will be recognized by a low to high transition on an IR input. The IR input can remain high without generating another interrupt.

If LTIM = '1', an interrupt request will be recognized by a 'high' level on IR Input, and there is no need for an edge detection. The interrupt request must be removed before the EOI command is issued or the CPU interrupt is enabled to prevent a second interrupt from occurring.

The priority cell diagram shows a conceptual circuit of the level sensitive and edge sensitive input circuitry of the 82C59A-2. Be sure to note that the request latch is a transparent D type latch.

In both the edge and level triggered modes the IR inputs must remain high until after the falling edge of the first INTA. If the IR input goes low before this time a DEFAULT IR7 will occur when the CPU acknowledges the interrupt. This can be a useful safeguard for detecting interrupts caused by spurious noise glitches on the IR inputs. To implement this feature the IR7 routine is used for "clean up" simply executing a return instruction, thus ignoring the interrupt. If IR7 is needed for other purposes a default IR7 can still be detected by reading the ISR. A normal IR7 interrupt will set the corresponding ISR bit, a default IR7 won't. If a default IR7 routine occurs during a normal IR7 routine, however, the ISR will remain set. In this case it is necessary to keep track of whether or not the IR7 routine was previously entered. If another IR7 occurs it is a default.

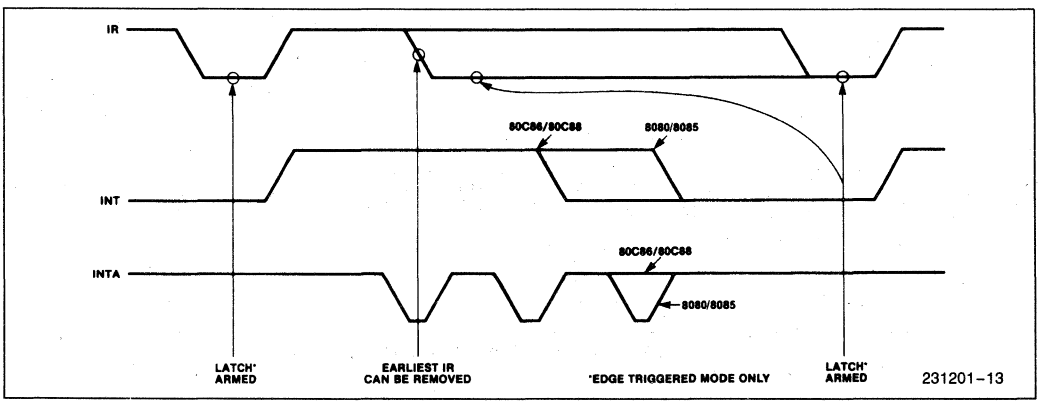


Figure 10. IR Triggering Timing Requirements

THE SPECIAL FULLY NESTED MODE

This mode will be used in the case of a big system where cascading is used, and the priority has to be conserved within each slave. In this case the fully nested mode will be programmed to the master (using ICW4). This mode is similar to the normal nested mode with the following exceptions:

a. When an interrupt request from a certain slave is in service this slave is not locked out from the master's priority logic and further interrupt requests from higher priority IR's within the slave will be recognized by the master and will initiate interrupts to the processor. (In the normal nestled mode a slave is masked out when its request is in service and no higher requests from the same slave can be serviced.)

b. When exiting the Interrupt Service routine the software has to check whether the interrupt serviced was the only one from that slave. This is done by sending a non-specific End of Interrupt (EOI) command to the slave and then reading its In-Service register and checking for zero. If it is empty, a non-specific EOI can be sent to the master too. If not, no EOI should be sent.

BUFFERED MODE

When the 82C59A-2 is used in a large system where bus driving buffers are required on the data bus and the cascading mode is used, there exists the problem of enabling buffers.

The buffered mode will structure the 82C59A-2 to send an enable signal on $\overline{\text{SP}}/\overline{\text{EN}}$ to enable the buffers. In this mode, whenever the 82C59A-2's data bus outputs are enabled, the $\overline{\text{SP}}/\overline{\text{EN}}$ output becomes active.

This modification forces the use of software programming to determine whether the 82C59A-2 is a master or a slave. Bit 3 in ICW4 programs the buffered mode, and bit 2 in ICW3 determines whether it is a master or a slave.

CASCADE MODE

The 82C59A-2 can be easily interconnected in a system of one master with up to eight slaves to handle up to 64 priority levels.

The master controls the slaves through the 3 line cascade bus. The cascade bus acts like chip selects to the slaves during the $\overline{\text{INTA}}$ sequence.

In a cascade configuration, the slave interrupt outputs are connected to the master interrupt request inputs. When a slave request line is activated and afterwards acknowledged, the master will enable the corresponding slave to release the device routine address during bytes 2 and 3 of INTA. (Byte 2 only for 80C86/80C88).

The cascade bus lines are normally low and will contain the slave address code from the trailing edge of the first INTA pulse to the trailing edge of the third pulse. Each 82C59A-2 in the system must follow a separate initialization sequence and can be programmed to work in a different mode. An EOI command must be issued twice: once for the master and once for the corresponding slave. An address decoder is required to activate the Chip Select (CS) input of each 82C59A-2.

The cascade lines of the Master 82C59A-2 are activated only for slave inputs, non slave inputs leave the cascade line inactive (low).

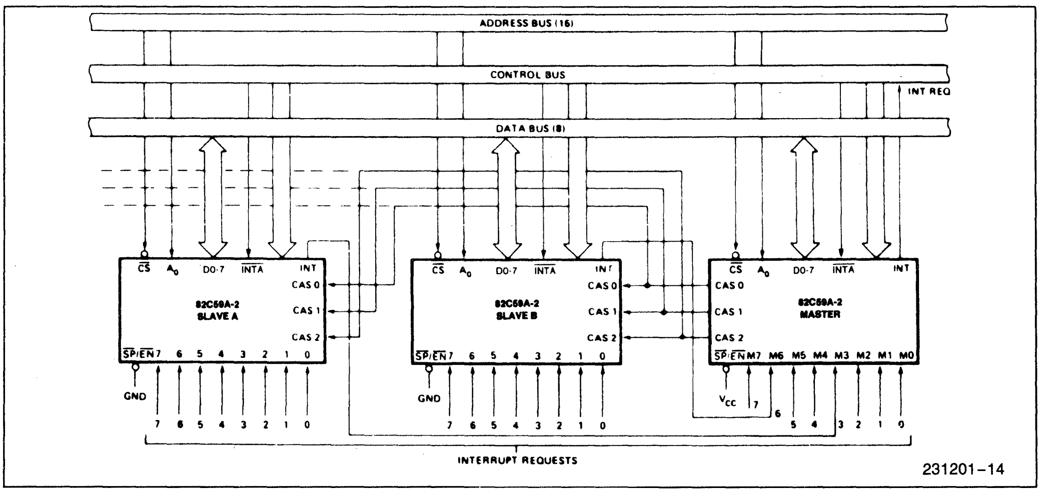


Figure 11. Cascading the 82C59A-2

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ...... 0°C to 70°C
Storage Temperature .... −65°C to + 150°C
Supply Voltage (w.r.t. ground) .... −0.5 to 7.0V
Input Voltage (w.r.t. ground) ... −0.5 to V_{CC} + 0.5V
Output Voltage (w.r.t. ground).. −0.5 to V_{CC} + 0.5V
Power Dissipation .... 0.9 Watt

Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±10%

Symbol	Parameter	Min	Max	Units	Test Conditions
I_{CCS}	Standby Supply Current		10	μA	$V_{IN} = V_{CC}$ or GND All IR = V_{CC} Outputs Unloaded V_{CC} = 5.5V
I_{CC}	Operating Supply Current		5	mA	(Note)
V_{IH}	Input High Voltage	2.2	V_{CC} + 0.5	V	
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 2.5 mA
V_{OH}	Output High Voltage	3.0 V_{CC} −0.4		V	I_{OH} = −2.5 mA I_{OH} = −100 μA
I_{LI}	Input Leakage Current		±1.0	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current		±10	μA	$0V \leq V_{OUT} \leq V_{CC}$
I_{LIR}	IR Input Leakage Current		−300 +10	μA	V_{IN} = 0 $V_{IN} = V_{CC}$

NOTE:
Repeated data input with 80C86-2 timings.

CAPACITANCE T_A = 25°C; V_{CC} = GND = 0V

Symbol	Parameter	Min	Max	Units	Test Conditions	
C_{IN}	Input Capacitance		7	pF		fc = 1 MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins at GND	
C_{OUT}	Output Capacitance		15	pF		

A.C. CHARACTERISTICS $T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$

TIMING REQUIREMENTS

Symbol	Parameter	82C59A-2		Units	Test Conditions
		Min	Max		
TAHRL	AO/$\overline{CS}$ Setup to $\overline{RD}$/$\overline{INTA}$ $\downarrow$	10		ns	
TRHAX	AO/$\overline{CS}$ Hold after $\overline{RD}$/$\overline{INTA}$ $\uparrow$	5		ns	
TRLRH	$\overline{RD}$/$\overline{INTA}$ Pulse Width	160		ns	
TAHWL	AO/$\overline{CS}$ Setup to $\overline{WR}$ $\downarrow$	0		ns	
TWHAX	AO/$\overline{CS}$ Hold after $\overline{WR}$ $\uparrow$	0		ns	
TWLWH	$\overline{WR}$ Pulse Width	190		ns	
TDVWH	Data Setup to $\overline{WR}$ $\uparrow$	160		ns	
TWHDX	Data Hold after $\overline{WR}$ $\uparrow$	0		ns	
TJLJH	Interrupt Request Width (Low)	100		ns	(See Note)
TCVIAL	Cascade Setup to Second or Third $\overline{INTA}$ $\downarrow$ (Slave Only)	40		ns	
TRHRL	End of $\overline{RD}$ to next $\overline{RD}$ End of $\overline{INTA}$ to next $\overline{INTA}$ within an $\overline{INTA}$ sequence only	160		ns	
TWHWL	End of $\overline{WR}$ to next $\overline{WR}$	190		ns	
*TCHCL	End of Command to next Command (Not same command type) End of $\overline{INTA}$ sequence to next $\overline{INTA}$ sequence.	400		ns	

*Worst case timing for TCHCL in an actual microprocessor system is typically much greater than 400 ns (i.e. 8085A = 1.6 μs, 8085-A2 = 1 μs, 80C86 = 1 μs, 80C86-2 = 625 ns)

NOTE:
This is the low time required to clear the input latch in the edge triggered mode.

TIMING RESPONSES

Symbol	Parameter	8259A-2		Units	Test Conditions**
		Min	Max		
TRLDV	Data Valid from $\overline{\text{RD}}/\overline{\text{INTA}}$ ↓		120	ns	1
TRHDZ	Data Float after $\overline{\text{RD}}/\overline{\text{INTA}}$ ↑	10	85	ns	2
TJHIH	Interrupt Output Delay		300	ns	1
TIALCV	Cascade Valid from First $\overline{\text{INTA}}$ ↓ (Master Only)		360	ns	1
TRLEL	Enable Active from $\overline{\text{RD}}$ ↓ or $\overline{\text{INTA}}$ ↓		110	ns	1
TRHEH	Enable Inactive from $\overline{\text{RD}}$ ↑ or $\overline{\text{INTA}}$ ↑		150	ns	1
TAHDV	Data Valid from Stable Address		200	ns	1
TCVDV	Cascade Valid to Valid Data		200	ns	1

**Test Condition Definition Table

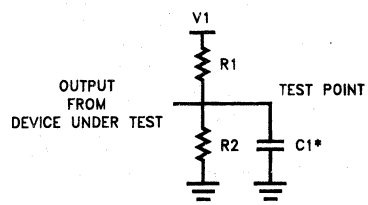
TEST CONDITION	V1	R1	R2	C1
1	1.7V	523Ω	OPEN	100 pf
2	4.5V	1.8 kΩ	1.8 kΩ	30 pf

A.C. TESTING INPUT, OUTPUT WAVEFORM

231201–15

A.C. Testing: All input signals must switch between $V_{IL} - 0.4V$ and $V_{IH} + 0.4V$. Input Rise and Fall Times must be ≤ 15 ns. All timing measurements are made at 2.4V and 0.45V.

A.C. TESTING LOAD CIRCUIT

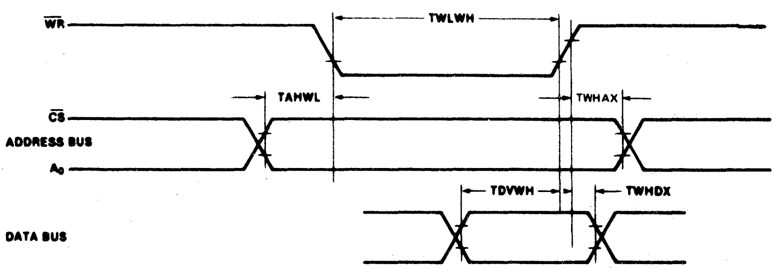


231201–16

*Includes Stray and Jig Capacitance

WAVEFORMS

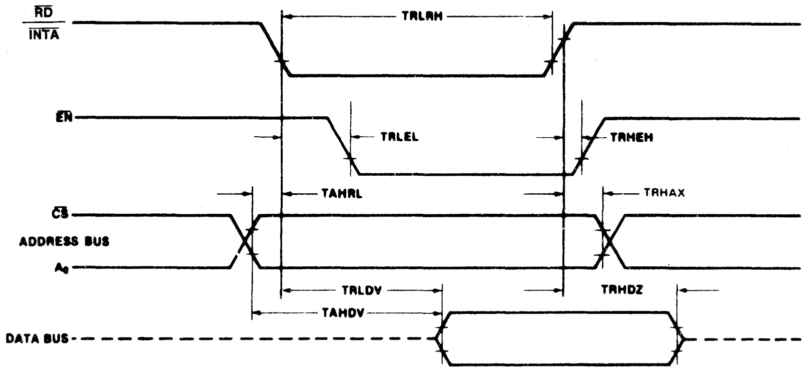
WRITE



231201–17

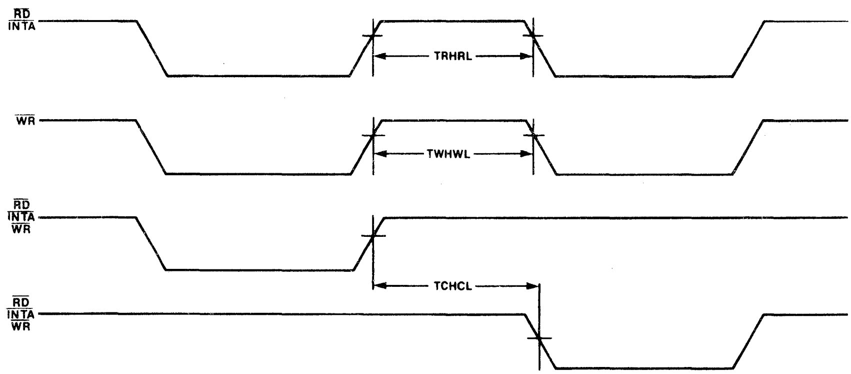
WAVEFORMS (Continued)

READ/INTA



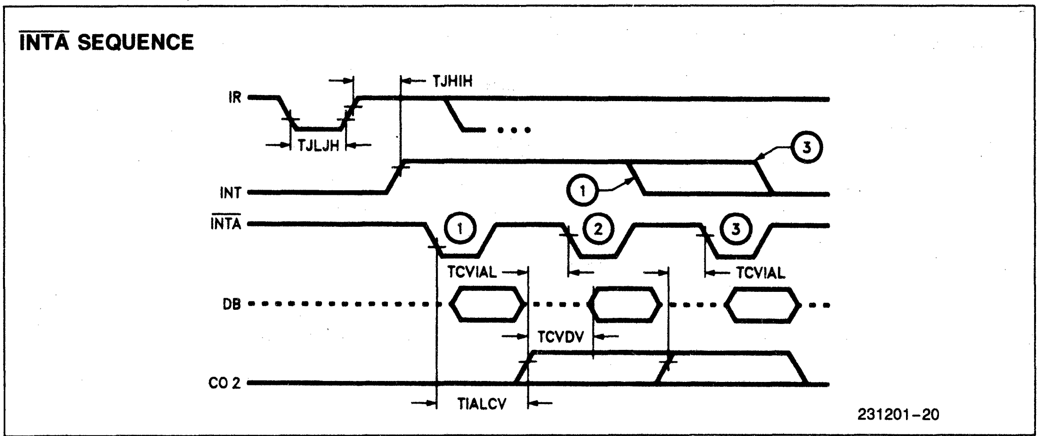
231201-18

OTHER TIMING



231201-19

WAVEFORMS (Continued)



NOTES:
Interrupt output must remain HIGH at least until leading edge of first INTA.
1. Cycle 1 in 80C86 and 80C88 systems, the Data Bus is not active.

intel®

8755A/8755A-2
16,384-BIT EPROM WITH I/O

- 2048 Words × 8 Bits
- Single +5V Power Supply (V_{CC})
- Directly Compatible with 8085A and 8088 Microprocessors
- U.V. Erasable and Electrically Reprogrammable
- Internal Address Latch
- 2 General Purpose 8-Bit I/O Ports
- Each I/O Port Line Individually Programmable as Input or Output
- Multiplexed Address and Data Bus
- 40-Pin DIP
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range

The Intel® 8755A is an erasable and electrically reprogrammable ROM (EPROM) and I/O chip to be used in the 8085AH and iAPX 88 microprocessor systems. The EPROM portion is organized as 2048 words by 8 bits. It has a maximum access time of 450 ns to permit use with no wait states in an 8085AH CPU.

The I/O portion consists of 2 general purpose I/O ports. Each I/O port has 8 port lines, and each I/O port line is individually programmable as input or output.

The 8755A-2 is a high speed selected version of the 8755A compatible with the 5 MHz 8085AH-2 and the 5 MHz iAPX 88 microprocessor.

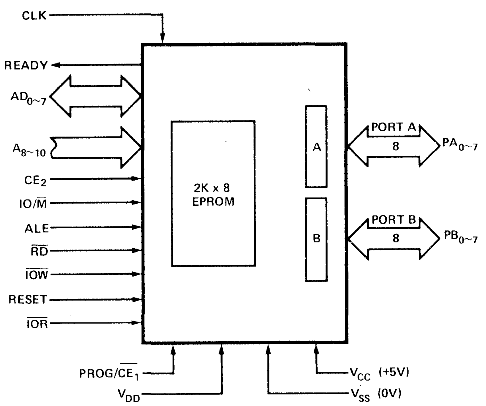


Figure 1. Block Diagram

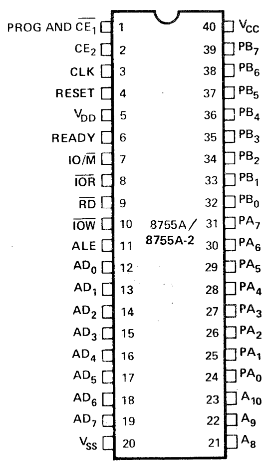


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Type	Name and Function
ALE	I	**Address Latch Enable:** When Address Latch Enable goes *high*, AD_{0-7}, IO/$\overline{M}$, A_{8-10}, CE_2, and $\overline{CE}_1$ enter the addrss latches. The signals (AD, IO/$\overline{M}$ AD_{8-10}, CE_2, $\overline{CE}_1$) are latched in at the trailing edge of ALE.
AD_{0-7}	I	**Bidirectional Address/Data Bus:** The lower 8-bits of the PROM or I/O address are applied to the bus lines when ALE is high. During an I/O cycle, Port A or B is selected based on the latched value of AD_0. IF $\overline{RD}$ or $\overline{IOR}$ is low when the latched Chip Enables are active, the output buffers present data on the bus.
A_{8-10}	I	**Address Bus:** These are the high order bits of the PROM address. They do not affect I/O operations.
PROG/$\overline{CE}_1$ CE_2	I	**Chip Enable Inputs:** $\overline{CE}_1$ is active low and CE_2 is active high. The 8755A can be accessed only when *both* Chip Enables are active at the time the ALE signal latches them up. If either Chip Enable input is not active, the AD_{0-7} and READY outputs will be in a high impedance state. $\overline{CE}_1$ is also used as a programming pin. (See section on programming.)
IO/$\overline{M}$	I	**I/O Memory:** If the latched IO/$\overline{M}$ is high when $\overline{RD}$ is low, the output data comes from an I/O port. If it is low the output data comes from the PROM.
$\overline{RD}$	I	**Read:** If the latched Chip Enables are active when $\overline{RD}$ goes low, the AD_{0-7} output buffers are enabled and output either the selected PROM location or I/O port. When both $\overline{RD}$ and $\overline{IOR}$ are high, the AD_{0-7} output buffers are 3-stated.
$\overline{IOW}$	I	**I/O Write:** If the latched Chip Enables are active, a low on $\overline{IOW}$ causes the output port pointed to by the latched value of AD_0 to be written with the data on AD_{0-7}. The state of IO/$\overline{M}$ is ignored.
CLK	I	**Clock:** The CLK is used to force the READY into its high impedance state after it has been forced low by $\overline{CE}_1$ low, CE_2 high, and ALE high.
READY	O	**Ready** is a 3-state output controlled by $\overline{CE}_1$, CE_2, ALE and CLK. READY is forced low when the Chip Enables are active during the time ALE is high, and remains low until the rising edge of the next CLK. (See Figure 6c.)
PA_{0-7}	I/O	**Port A:** These are general purpose I/O pins. Their input/output direction is determined by the contents of Data Direction Register (DDR). Port A is selected for write operations when the Chip Enables are active and $\overline{IOW}$ is low and a 0 was previously latched from AD_0, AD_1. Read Operation is selected by either $\overline{IOR}$ low and active Chip Enables and AD_0 and AD_1 low, *or* IO/$\overline{M}$ high, $\overline{RD}$ low, active Chip Enables, and AD_0 and AD_1 low.
PB_{0-7}	I/O	**Port B:** This general purpose I/O port is identical to Port A except that it is selected by a 1 latched from AD_0 and a 0 from AD_1.
RESET	I	**Reset:** In normal operation, an input high on RESET causes all pins in Ports A and B to assume input mode (clear DDR register).
$\overline{IOR}$	I	**I/O Read:** When the Chip Enables are active, a low on $\overline{IOR}$ will output the selected I/O port onto the AD bus. $\overline{IOR}$ low performs the same function as the combination of IO/$\overline{M}$ high and $\overline{RD}$ low. When $\overline{IOR}$ is not used in a system, $\overline{IOR}$ should be tied to V_{CC} ("1").
V_{CC}		**Power:** +5 volt supply.
V_{SS}		**Ground:** Reference.
V_{DD}		**Power Supply:** V_{DD} is a programming voltage, and must be tied to V_{CC} when the 8755A is being read. For programming, a high voltage is supplied with V_{DD} = 25V, typical. (See section on programming.)

FUNCTIONAL DESCRIPTION

PROM Section

The 8755A contains an 8-bit address latch which allows it to interface directly to MCS-48, MCS-85 and iAPX 88/10 Microcomputers without additional hardware.

The PROM section of the chip is addressed by the 11-bit address and the Chip Enables. The address, $\overline{CE_1}$ and CE_2 are latched into the address latches on the falling edge of ALE. If the latched Chip Enables are active and IO/$\overline{M}$ is low when $\overline{RD}$ goes low, the contents of the PROM location addressed by the latched address are put out on the AD_{0-7} lines (provided that V_{DD} is tied to V_{CC}.)

I/O Section

The I/O section of the chip is addressed by the latched value of AD_{0-1}. Two 8-bit Data Direction Registers (DDR) in 8755A determine the input/output status of each pin in the corresponding ports. A "0" in a particular bit position of a DDR signifies that the corresponding I/O port bi is in the input mode. A "1" in a particular bit position signifies that the corresponding I/O port bit is in the output mode. In this manner the I/O ports of the 8755A are bit-by-bit programmable as inputs or outputs. The table summarizes port and DDR designation. DDR's cannot be read.

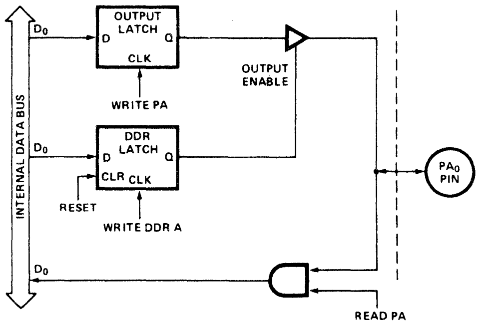
AD_1	AD_0	Selection
0	0	Port A
0	1	Port B
1	0	Port A Data Direction Register (DDR A)
1	1	Port B Data Direction Register (DDR B)

When $\overline{IOW}$ goes low and the Chip Enables are active, the data on the AD_{0-7} is written into I/O port selected by the latched value of AD_{0-1}. During this operation all I/O bits of the selected port are affected, regardless of their I/O mode and the state of IO/$\overline{M}$. The actual output level does not change until $\overline{IOW}$ returns high. (glitch free output)

A port can be read out when the latched Chip Enables are active and either $\overline{RD}$ goes low with IO/$\overline{M}$ high, or $\overline{IOR}$ goes low. Both input and output mode bits of a selected port will appear on lines AD_{0-7}.

To clarify the function of the I/O Ports and Data Direction Registers, the following diagram shows the configuration of one bit of PORT A and DDR A. The same logic applies to PORT B and DDR B.

8755A
ONE BIT OF PORT A AND DDR A:



WRITE PA = ($\overline{IOW}$=0) • (CHIP ENABLES ACTIVE) • (PORT A ADDRESS SELECTED)
WRITE DDR A = ($\overline{IOW}$=0) • (CHIP ENABLES ACTIVE) • (DDR A ADDRESS SELECTED)
READ PA = {[(IO/$\overline{M}$=1) • ($\overline{RD}$=0)] + ($\overline{IOR}$=0)} • (CHIP ENABLES ACTIVE) • (PORT A ADDRESS SELECTED)
NOTE: WRITE PA IS NOT QUALIFIED BY IO/$\overline{M}$.

Note that hardware RESET or writing a zero to the DDR latch will cause the output latch's output buffer to be disabled, preventing the data in the Output Latch from being passed through to the pin. This is equivalent to putting the port in the input mode. Note also that the data can be written to the Output Latch even though the Output Buffer has been disabled. This enables a port to be initialized with a value prior to enabling the output.

The diagram also shows that the contents of PORT A and PORT B can be read even when the ports are configured as outputs.

TABLE 1. 8755A PROGRAMMING MODULE CROSS REFERENCE

MODULE NAME	USE WITH
UPP 955	UPP(4)
UPP UP2(2)	UPP 855
PROMPT 975	PROMPT 80/85(3)
PROMPT 475	PROMPT 48(1)

NOTES:
1. Described on p. 13-34 of 1978 Data Catalog.
2. Special adaptor socket.
3. Described on p. 13-39 of 1978 Data Catalog.
4. Described on p. 13-71 of 1978 Data Catalog.

ERASURE CHARACTERISTICS

The erasure characteristics of the 8755A are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000-4000Å range. Data show that constant exposure to room level fluorescent lighting could erase the typical 8755A in approximately 3 years while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 8755A is to be exposed to these types of lighting conditions for extended periods of time, opaque labels are available from Intel which should be placed over the 8755 window to prevent unintentional erasure.

The recommended erasure procedure for the 8755A is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms (Å). The integrated dose (i.e., UV intensity X exposure time) for erasure should be a minimum of 15W-sec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000μW/cm^2 power rating. The 8755A should be placed within one inch from the lamp tubes during erasure. Some lamps have a filter on their tubes and this filter should be removed before erasure.

PROGRAMMING

Initially, and after each erasure, all bits of the EPROM portions of the 8755A are in the "1" state. Information is introduced by selectively programming "0" into the desired bit locations. A programmed "0" can only be changed to a "1" by UV erasure.

The 8755A can be programmed on the Intel® Universal PROM Programmer (UPP), and the PROMPT™ 80/85 and PROMPT-48™ design aids. The appropriate programming modules and adapters for use in programming both 8755A's and 8755's are shown in Table 1.

The program mode itself consists of programming a single address at a time, giving a single 50 msec pulse for every address. Generally, it is desirable to have a verify cycle after a program cycle for the same address as shown in the attached timing diagram. In the verify cycle (i.e., normal memory read cycle) 'V_{DD}' should be at +5V.

Preliminary timing diagrams and parameter values pertaining to the 8755A programming operation are contained in Figure 7.

SYSTEM APPLICATIONS

System Interface with 8085AH and iAPX 88

A system using the 8755A can use either one of the two I/O Interface techniques:

- Standard I/O
- Memory Mapped I/O

If a standard I/O technique is used, the system can use the feature of both CE_2 and $\overline{CE_1}$. By using a combination of unused address lines A_{11-15} and the Chip Enable inputs, the 8085AH system can use up to 5 each 8755A's without requiring a CE decoder. See Figure 4a and 4b.

If a memory mapped I/O approach is used the 8755A will be selected by the combination of both the Chip Enables and IO/$\overline{M}$ using AD_{8-15} address lines. See Figure 3.

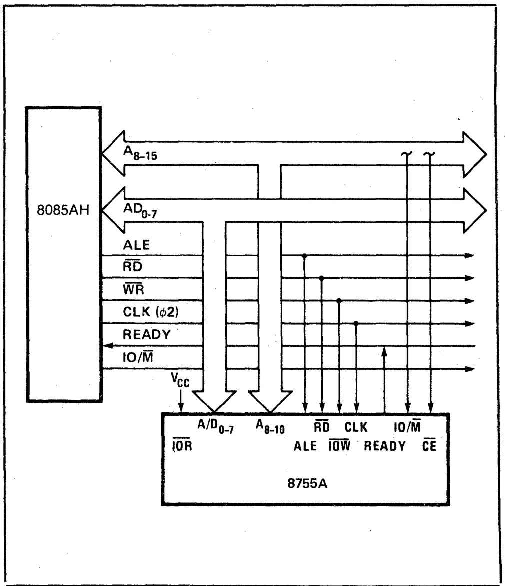


Figure 3. 8755A in 8085AH System (Memory-Mapped I/O)

iAPX 88 FIVE CHIP SYSTEM

Figure 4 shows a five chip system containing:

- 1.25K Bytes RAM
- 2K Bytes EPROM
- 38 I/O Pins
- 1 Interval Timer
- 2 Interrupt Levels

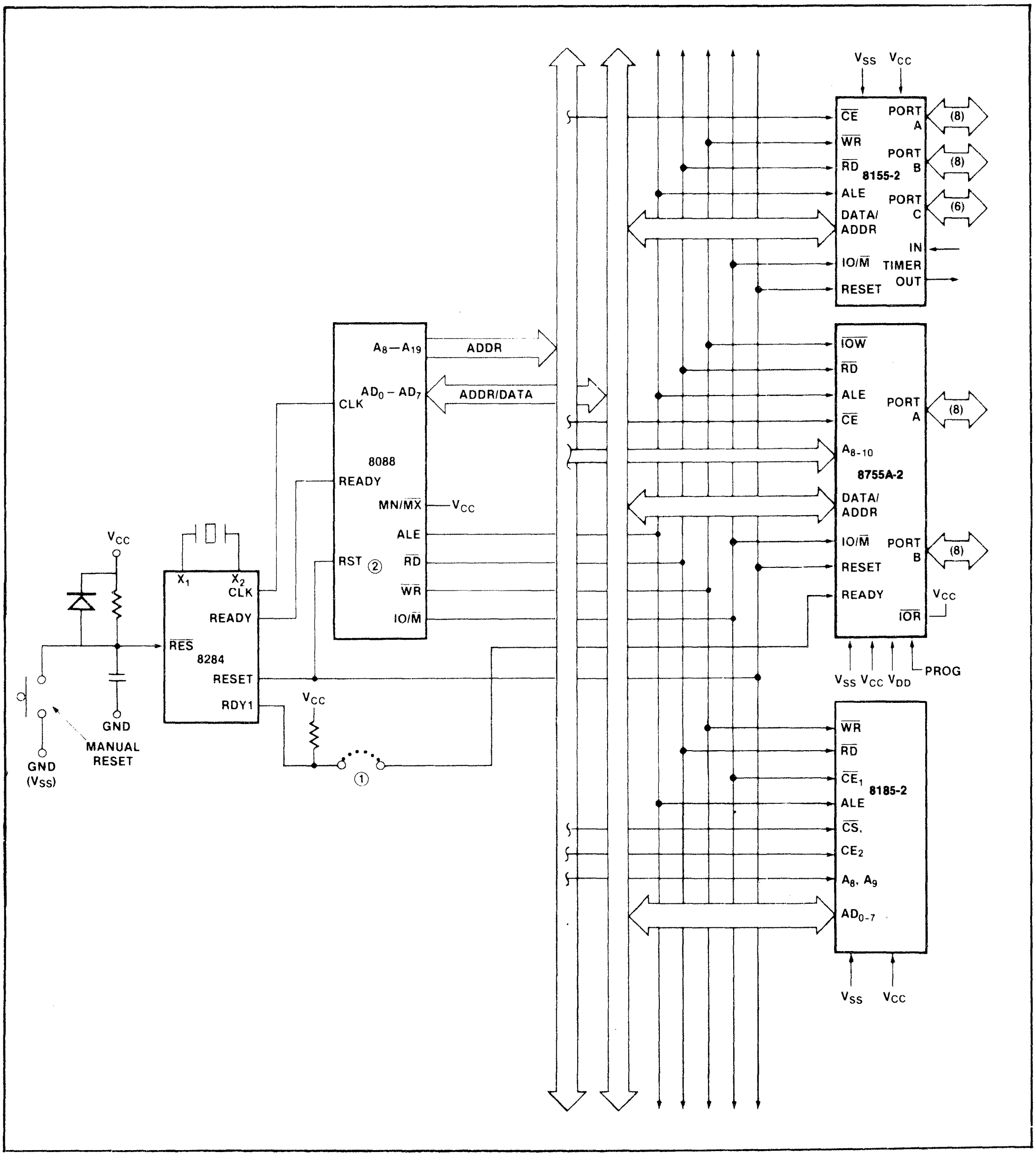
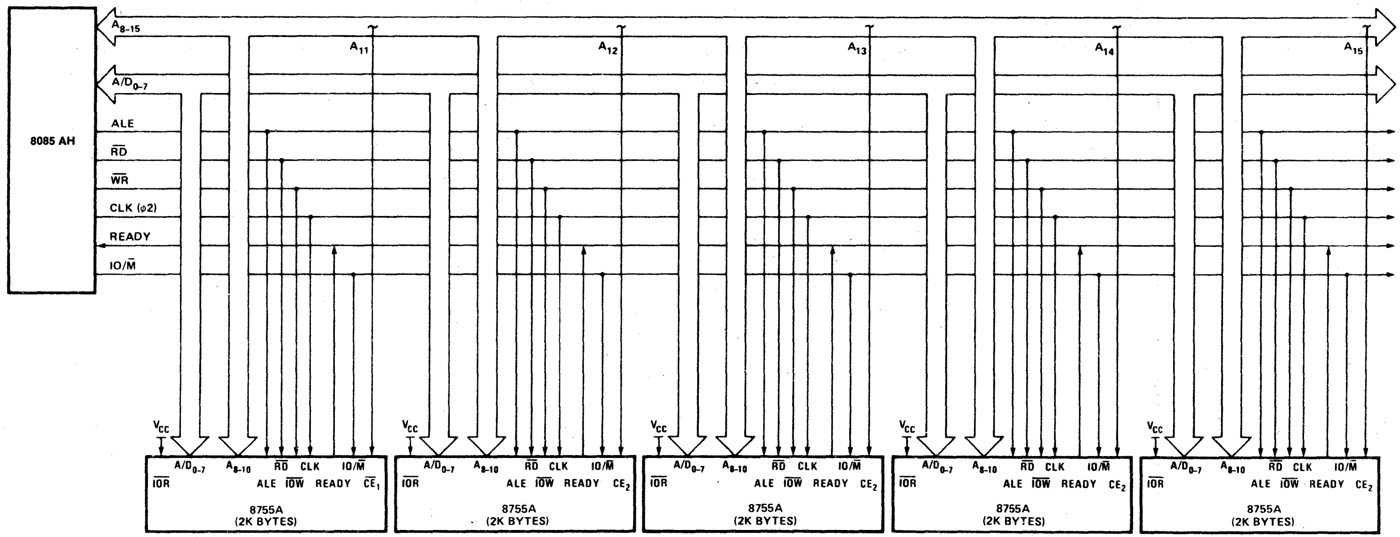


Figure 4a. iAPX 88 Five Chip System Configuration



Note: Use $\overline{CE}_1$ for the first 8755A in the system, and CE_2 for the other 8755A's. Permits up to 5-8755A's in a system without CE decoder.

Figure 4b. 8755A in 8085A System (Standard I/O)

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... 0°C to +70°C
Storage Temperature .... -65°C to +150°C
Voltage on Any Pin
With Respect to Ground .... -0.5V to +7V
Power Dissipation 1.5W

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°, V_{CC} = V_{DD} = 5V ± 5%; V_{CC} = V_{DD} = 5V ±10% for 8755A-2)

SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
V_{IL}	Input Low Voltage	-0.5	0.8	V	V_{CC} = 5.0V
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	V_{CC} = 5.0V
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -400μA
I_{IL}	Input Leakage		10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current		±10	μA	0.45V ≤ $V_{OUT} \leq V_{CC}$
I_{CC}	V_{CC} Supply Current		180	mA	
I_{DD}	V_{DD} Supply Current		30	mA	$V_{DD} = V_{CC}$
C_{IN}	Capacitance of Input Buffer		10	pF	f_C = 1μHz
$C_{I/O}$	Capacitance of I/O Buffer		15	pF	f_C = 1μHz

D.C. CHARACTERISTICS — PROGRAMMING (T_A = 0°C to 70°, V_{CC} = 5V ± 5%, V_{SS} = 0V, V_{DD} = 25V ±1V; V_{CC} = V_{DD} = 5V ±10% for 8755A-2)

Symbol	Parameter	Min.	Typ.	Max.	Unit
V_{DD}	Programming Voltage (during Write to EPROM)	24	25	26	V
I_{DD}	Prog Supply Current		15	30	mA

A.C. CHARACTERISTICS (T_A = 0°C to 70°, V_{CC} = 5V ± 5%; V_{CC} = V_{DD} = 5V ±10% for 8755A-2)

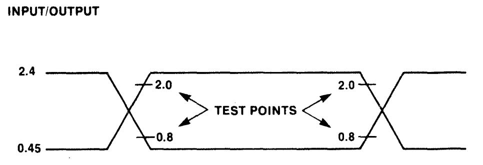
Symbol	Parameter	8755A Min.	8755A Max.	8755A-2 (Preliminary) Min.	8755A-2 (Preliminary) Max.	Units
t_{CYC}	Clock Cycle Time	320		200		ns
T_1	CLK Pulse Width	80		40		ns
T_2	CLK Pulse Width	120		70		ns
t_f,t_r	CLK Rise and Fall Time		30		30	ns
t_{AL}	Address to Latch Set Up Time	50		30		ns
t_{LA}	Address Hold Time after Latch	80		45		ns
t_{LC}	Latch to READ/WRITE Control	100		40		ns
t_{RD}	Valid Data Out Delay from READ Control*		170		140	ns
t_{AD}	Address Stable to Data Out Valid**		450		300	ns
t_{LL}	Latch Enable Width	100		70		ns
t_{RDF}	Data Bus Float after READ	0	100	0	85	ns
t_{CL}	READ/WRITE Control to Latch Enable	20		10		ns
t_{CC}	READ/WRITE Control Width	250		200		ns
t_{DW}	Data In to Write Set Up Time	150		150		ns
t_{WD}	Data In Hold Time After WRITE	30		10		ns
t_{WP}	WRITE to Port Output		400		300	ns
t_{PR}	Port Input Set Up Time	50		50		ns
t_{RP}	Port Input Hold Time to Control	50		50		ns
t_{RYH}	READY HOLD Time to Control	0	160	0	160	ns
t_{ARY}	ADDRESS (CE) to READY		160		160	ns
t_{RV}	Recovery Time Between Controls	300		200		ns
t_{RDE}	READ Control to Data Bus Enable	10		10		ns

NOTE:
C_{LOAD} = 150pF.
*Or $T_{AD} - (T_{AL} + T_{LC})$, whichever is greater.
**Defines ALE to Data Out Valid in conjunction with T_{AL}.

A.C. CHARACTERISTICS—PROGRAMMING (T_A = 0°C to 70°, V_{CC} = 5V ± 5%, V_{SS} = 0V, V_{DD} = 25V ±1V; V_{CC} = V_{DD} = 5V ±10% for 8755A-2)

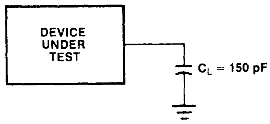
Symbol	Parameter	Min.	Typ.	Max.	Unit
t_{PS}	Data Setup Time	10			ns
t_{PD}	Data Hold Time	0			ns
t_S	Prog Pulse Setup Time	2			μs
t_H	Prog Pulse Hold Time	2			μs
t_{PR}	Prog Pulse Rise Time	0.01	2		μs
t_{PF}	Prog Pulse Fall Time	0.01	2		μs
t_{PRG}	Prog Pulse Width	45	50		msec

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0.8V FOR A LOGIC "0."

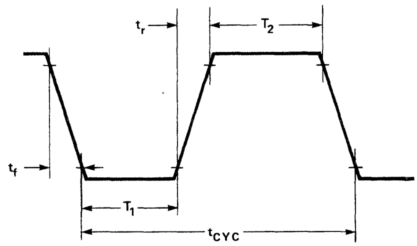
A.C. TESTING LOAD CIRCUIT



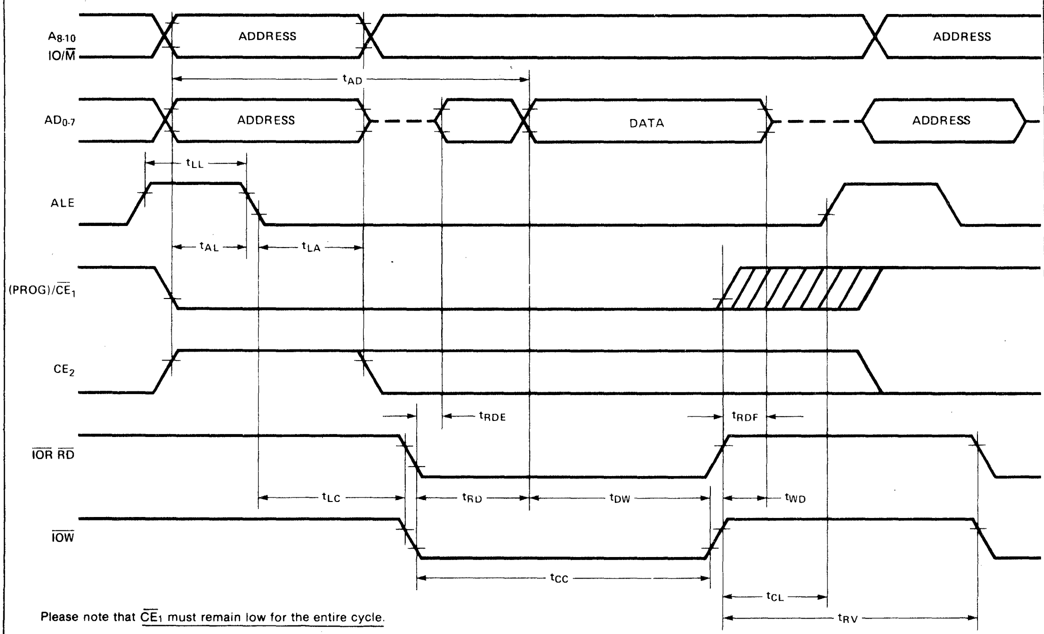
C_L = 150 pF
C_L INCLUDES JIG CAPACITANCE

WAVEFORMS

CLOCK SPECIFICATION FOR 8755A



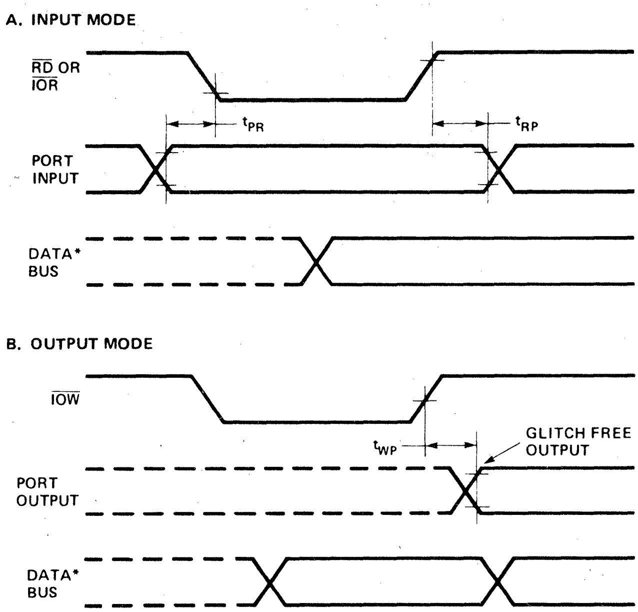
PROM READ, I/O READ AND WRITE



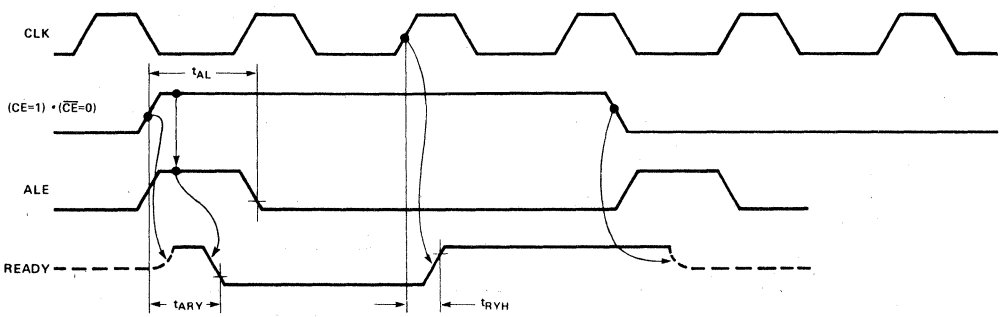
Please note that $\overline{CE_1}$ must remain low for the entire cycle.

WAVEFORMS (Continued)

I/O PORT

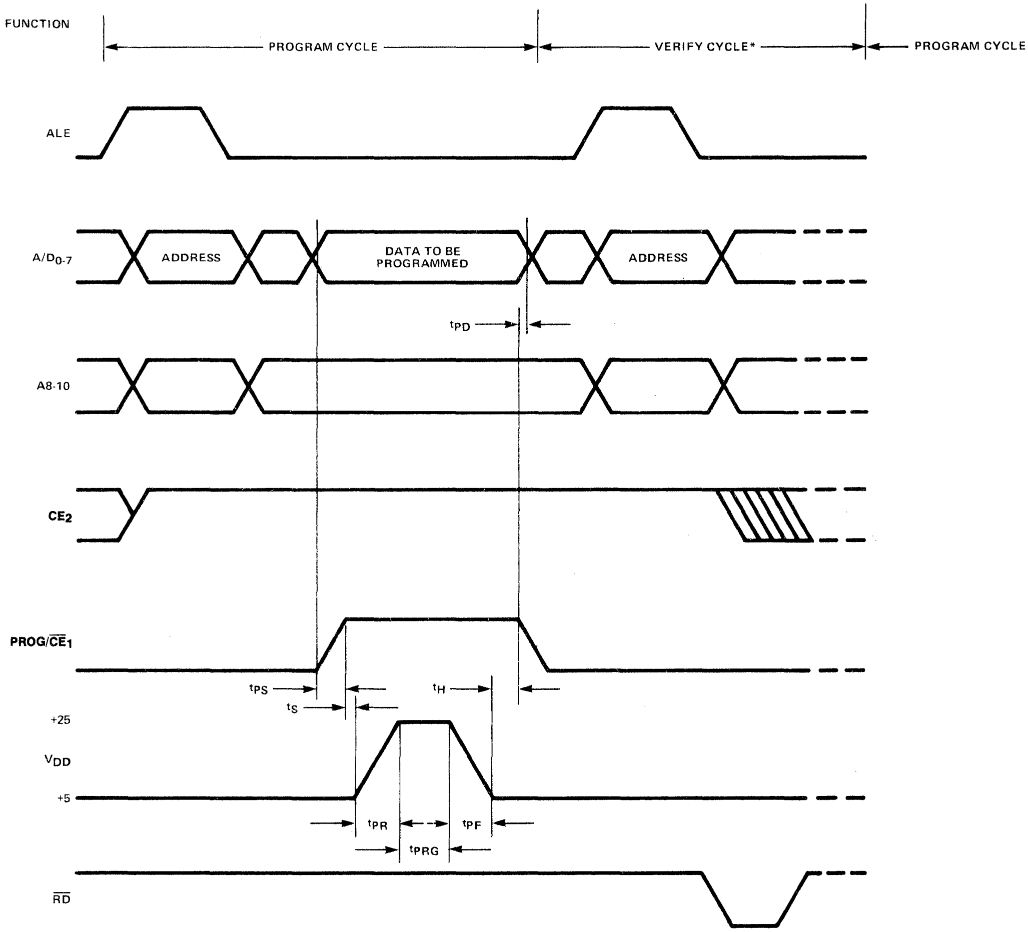


WAIT STATE (READY = 0)



WAVEFORMS (Continued)

8755A PROGRAM MODE



*VERIFY CYCLE IS A REGULAR MEMORY READ CYCLE (WITH V_{DD} = +5V FOR 8755A)

intel®

APPLICATION NOTE

AP-59

September 1979

Using the 8259A Programmable Interrupt Controller

Robin Jigour
Microcomputer Applications

121500-001

INTRODUCTION

The Intel 8259A is a Programmable Interrupt Controller (PIC) designed for use in real-time interrupt driven microcomputer systems. The 8259A manages eight levels of interrupts and has built-in features for expansion up to 64 levels with additional 8259A's. Its versatile design allows it to be used within MCS-80, MCS-85, MCS-86, and MCS-88 microcomputer systems. Being fully programmable, the 8259A provides a wide variety of modes and commands to tailor 8259A interrupt processing for the specific needs of the user. These modes and commands control a number of interrupt oriented functions such as interrupt priority selection and masking of interrupts. The 8259A programming may be dynamically changed by the software at any time, thus allowing complete interrupt control throughout program execution.

The 8259A is an enhanced, fully compatible revision of its predecessor, the 8259. This means the 8259A can use all hardware and software originally designed for the 8259 without any changes. Furthermore, it provides additional modes that increase its flexibility in MCS-80 and MCS-85 systems and allow it to work in MCS-86 and MCS-88 systems. These modes are:

- MCS-86/88 Mode
- Automatic End of Interrupt Mode
- Level Triggered Mode
- Special Fully Nested Mode
- Buffered Mode

Each of these are covered in depth further in this application note.

This application note was written to explain completely how to use the 8259A within MCS-80, MCS-85, MCS-86, and MCS-88 microcomputer systems. It is divided into five sections. The first section, "Concepts", explains the concepts of interrupts and presents an overview of how the 8259A works with each microcomputer system mentioned above. The second section, "Functional Block Diagram", describes the internal functions of the 8259A in block diagram form and provides a detailed functional description of each device pin. "Operation of the 8259A", the third section, explains in depth the operation and use of each of the 8259A modes and commands. For clarity of explanation, this section doesn't make reference to the actual programming of the 8259A. Instead, all programming is covered in the fourth section, "Programming the 8259A". This section explains how to program the 8259A with the modes and commands mentioned in the previous section. These two sections are referenced in Appendix A. The fifth and final section "Application Examples", shows the 8259A in three typical applications. These applications are fully explained with reference to both hardware and software.

The reader should note that some of the terminology used throughout this application note may differ slightly from existing data sheets. This is done to better clarify and explain the operation and programming of the 8259A.

1. CONCEPTS

In microcomputer systems there is usually a need for the processor to communicate with various Input/Output (I/O) devices such as keyboards, displays, sensors, and other peripherals. From the system viewpoint, the processor should spend as little time as possible servicing the peripherals since the time required for these I/O chores directly affects the amount of time available for other tasks. In other words, the system should be designed so that I/O servicing has little or no effect on the total system throughput. There are two basic methods of handling the I/O chores in a system: status polling and interrupt servicing.

The status poll method of I/O servicing essentially involves having the processor "ask" each peripheral if it needs servicing by testing the peripheral's status line. If the peripheral requires service, the processor branches to the appropriate service routine; if not, the processor continues with the main program. Clearly, there are several problems in implementing such an approach. First, how often a peripheral is polled is an important constraint. Some idea of the "frequency-of-service" required by each peripheral must be known and any software written for the system must accommodate this time dependence by "scheduling" when a device is polled. Second, there will obviously be times when a device is polled that is not ready for service, wasting the processor time that it took to do the poll. And other times, a ready device would have to wait until the processor "makes its rounds" before it could be serviced, slowing down the peripheral.

Other problems arise when certain peripherals are more important than others. The only way to implement the "priority" of devices is to poll the high priority devices more frequently than lower priority ones. It may even be necessary to poll the high priority devices while in a low priority device service routine. It is easy to see that the polled approach can be inefficient both time-wise and software-wise. Overall, the polled method of I/O servicing can have a detrimental effect on system throughput, thus limiting the tasks that can be performed by the processor.

A more desirable approach in most systems would allow the processor to be executing its main program and only stop to service the I/O when told to do so by the I/O itself. This is called the interrupt service method. In effect, the device would asynchronously signal the processor when it required service. The processor would finish its current instruction and then vector to the service routine for the device requesting service. Once the service routine is complete, the processor would resume exactly where it left off. Using the interrupt service method, no processor time is spent testing devices, scheduling is not needed, and priority schemes are readily implemented. It is easy to see that, using the interrupt service approach, system throughput would increase, allowing more tasks to be handled by the processor.

However, to implement the interrupt service method between processor and peripherals, additional hardware is usually required. This is because, after interrupting the processor, the device must supply information for vectoring program execution. Depending on the processor used, this can be accomplished by the device taking control of the data bus and "jamming" an instruction(s) onto it. The instruction(s) then vectors the program to the proper service routine. This of course requires additional control logic for each interrupt requesting device. Yet the implementation so far is only in the most basic form. What if certain peripherals are to be of higher priority than others? What if certain interrupts must be disabled while others are to be enabled? The possible variations go on, but they all add up to one theme; to provide greater flexibility using the interrupt service method, hardware requirements increase.

So, we're caught in the middle. The status poll method is a less desirable way of servicing I/O in terms of throughput, but its hardware requirements are minimal. On the other hand, the interrupt service method is most desirable in terms of flexibility and throughput, but additional hardware is required.

The perfect situation would be to have the flexibility and throughput of the interrupt method in an implementation with minimal hardware requirements. The 8259A Programmable Interrupt Controller (PIC) makes this all possible.

The 8259A Programmable Interrupt Controller (PIC) was designed to function as an overall manager of an interrupt driven system. No additional hardware is required. The 8259A alone can handle eight prioritized interrupt levels, controlling the complete interface between peripherals and processor. Additional 8259A's can be "cascaded" to increase the number of interrupt levels processed. A wide variety of modes and commands for programming the 8259A give it enough flexibility for almost any interrupt controlled structure. Thus, the 8259A is the feasible answer to handling I/O servicing in microcomputer systems.

Now, before explaining exactly how to use the 8259A, let's go over interrupt structures of the MCS-80, MCS-85, MCS-86, and MCS-88 systems, and how they interact with the 8259A. Figure 1 shows a block diagram of the 8259A interfacing with a standard system bus. This may prove useful as reference throughout the rest of the "Concepts" section.

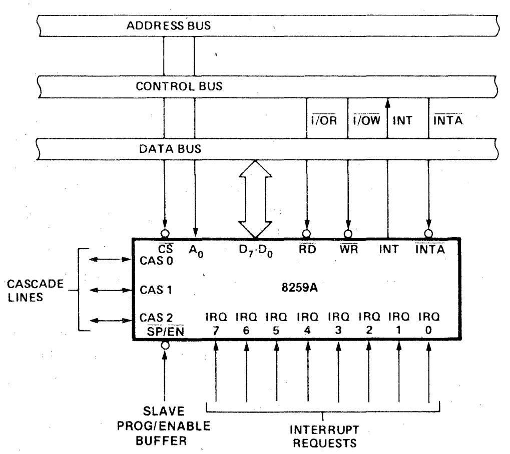


Figure 1. 8259A Interface to Standard System Bus

1.1 MCS-80 —8259A OVERVIEW

In an MCS-80—8259A interrupt configuration, as in Figure 2, a device may cause an interrupt by pulling one of the 8259A's interrupt request pins (IR0-IR7) high. If the 8259A accepts the interrupt request (this depends on its programmed condition), the 8259A's INT (interrupt) pin will go high, driving the 8080A's INT pin high.

The 8080A can receive an interrupt request any time, since its INT input is asynchronous. The 8080A, however, doesn't always have to acknowledge an interrupt request immediately. It can accept or disregard requests under software control using the EI (Enable Interrupt) or DI (Disable Interrupt) instructions. These instructions either set or reset an internal interrupt enable flip-flop. The output of this flip-flop controls the state of the INTE (Interrupt Enabled) pin. Upon reset, the 8080A interrupts are disabled, making INTE low.

At the end of each instruction cycle, the 8080A examines the state of its INT pin. If an interrupt request is present and interrupts are enabled, the 8080A enters an interrupt machine cycle. During the interrupt machine cycle the 8080A resets the internal interrupt enable flip-flop, disabling further interrupts until an EI instruction is executed. Unlike normal machine cycles, the interrupt machine cycle doesn't increment the program counter. This ensures that the 8080A can return to the pre-interrupt program location after the interrupt is completed. The 8080A then issues an $\overline{INTA}$ (Interrupt Acknowledge) pulse via the 8228 System Controller Bus Driver. This $\overline{INTA}$ pulse signals the 8259A that the 8080A is honoring the request and is ready to process the interrupt.

The 8259A can now vector program execution to the corresponding service routine. This is done during a sequence of the three $\overline{INTA}$ pulses from the 8080A via the 8228. Upon receiving the first $\overline{INTA}$ pulse the 8259A places the opcode for a CALL instruction on the data bus. This causes the contents of the program counter to be pushed onto the stack. In addition, the CALL instruction causes two more $\overline{INTA}$ pulses to be issued, allowing the 8259A to place onto the data bus the starting address of the corresponding service routine. This address is called the interrupt-vector address. The lower 8 bits (LSB) of the interrupt-vector address are released during the second $\overline{INTA}$ pulse and the upper 8 bits (MSB) during the third $\overline{INTA}$ pulse. Once this sequence is completed, program execution then vectors to the service routine at the interrupt-vector address.

If the same registers are used by both the main program and the interrupt service routine, their contents should be saved when entering the service routine. This includes the Program Status Word (PSW) which consists of the accumulator and flags. The best way to do this is to "PUSH" each register used onto the stack. The service routine can then "POP" each register off the stack in the reverse order when it is completed. This prevents any ambiguous operation when returning to the main program.

Once the service routine is completed, the main program may be re-entered by using a normal RET (Return) instruction. This will "POP" the original contents of the program counter back off the stack to resume program execution where it left off. Note, that because interrupts are disabled during the interrupt acknowledge sequence, the EI instruction must be executed either during the service routine or the main program before further interrupts can be processed.

For additional information on the 8080A interrupt structure and operation, refer to the MCS-80 User's Manual.

1.2 MCS-85 —8259A OVERVIEW

An MCS-85—8259A configuration processes interrupts in much the same format as an MCS-80—8259A configuration. When an interrupt occurs, a sequence of three $\overline{INTA}$ pulses causes the 8259A to release onto the data bus a CALL instruction and an interrupt-vector address for the corresponding service routine. Other events that occur during the 8080A interrupt machine cycle, such as disabling interrupts and not incrementing the program counter, also occur in the 8085A interrupt acknowledge machine cycle. Additionally, the instructions for saving registers, enabling or disabling of interrupts, and returning from service routines are literally the same.

The 8085A, however, has a different interrupt hardware scheme as shown in Figure 3. For one, the 8085A supplies its own $\overline{INTA}$ output pin rather than using an addi-

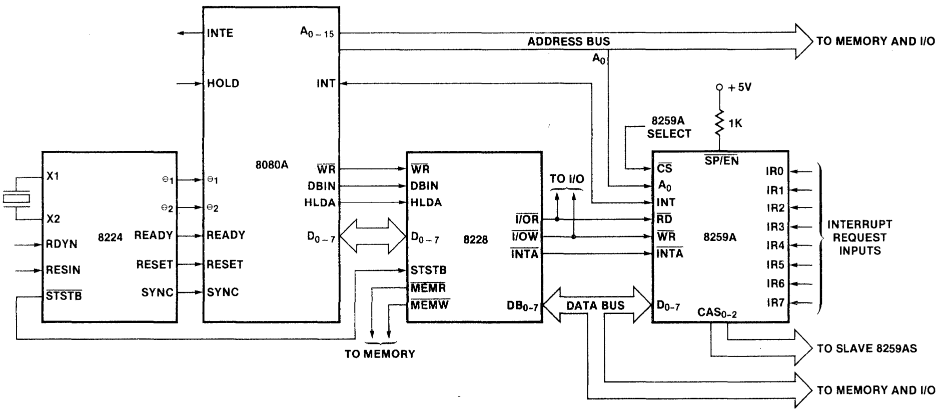


Figure 2. MCS-80 8259A Basic Configuration Example

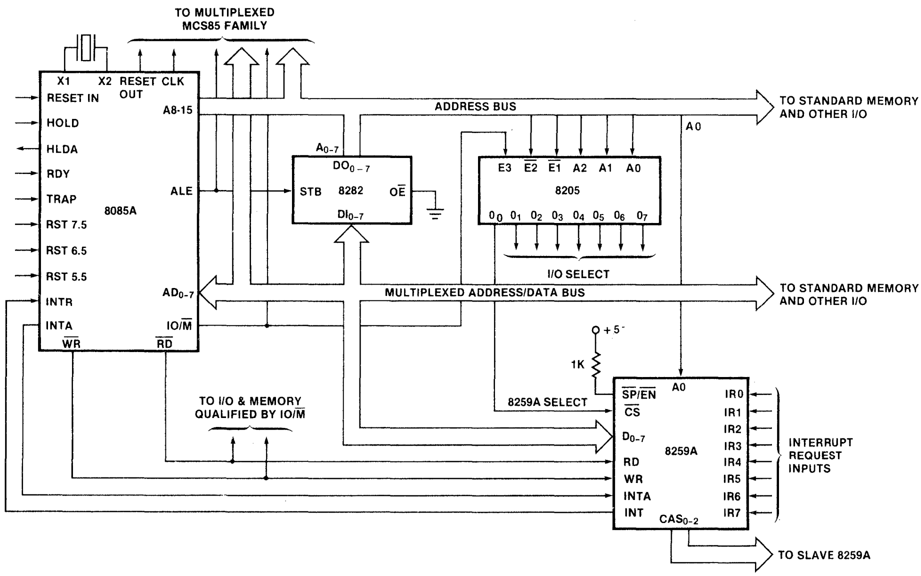


Figure 3. MCS-85 8259A Basic Configuration Example

tional chip, as the 8080A uses the 8228 System Controller Bus Driver. Another hardware difference is the 8085A has five hardware interrupt pins: INTR, RST 7.5, RST 6.5, RST 5.5, and TRAP. The INTR (Interrupt Request) pin is the equivalent to the 8080A's INT pin. The RST (Restart) pins and TRAP pin are all restart interrupts which vector program execution to an individual dedicated address when asserted. The important factor associating these interrupts is their relative priority, as shown below:

TRAP	Highest Priority
RST 7.5	
RST 6.5	
RST 5.5	
INTR	Lowest Priority

The INTR pin has lowest priority among the other 8085A hardware interrupts. Thus, precautions to prevent interrupting 8259A service routines may be necessary. This, of course, depends on how the 8085A interrupts are being used in a particular application. Such precautions can be implemented, however, by masking the RST pins using the SIM instruction. The TRAP pin on the other hand is non-maskable; all interrupt pins but TRAP can be controlled by the EI (Enable Interrupt) and DI (Disable Interrupt) instructions.

For a complete description of the 8085A interrupt structure, refer to the MCS-85 User's Manual.

1.3 MCS-86/88 —8259A OVERVIEW

Operation of an MCS-86/88—8259A configuration has basic similarities of the MCS-80/85—8259A configurations. That is, a device can cause an interrupt by pulling one of the 8259A's interrupt request pins (IR0-IR7) high. If the 8259A honors the request, its INT pin will go high, driving the 8086/8088's INTR pin high. Like the 8080A and 8085A, the INTR pin of the 8086/8088 is asynchronous, thus it can receive an interrupt any time. The 8086/8088 can also accept or disregard requests on INTR under software control using the STI (Set Interrupt) or CLI (Clear Interrupt) instructions. These instructions set or clear the interrupt-enabled flag IF. Upon 8086/8088 reset the IF flag is cleared, disabling external interrupts on INTR. Beside the INTR pin, the 8086/8088 provides an NMI (Non-Maskable Interrupt) pin. The NMI functions similar to the 8085A's TRAP; it can't be disabled or masked. NMI has higher priority than INTR.

Figure 4 shows an MCS-86 MAX Mode system interfacing with an 8259A on the local bus. This MCS-86—8259A configuration is also representative of an MCS-88—8259A configuration except for the data bus which is 16 bits for 8086 and 8 bits for 8088. In the MCS-86 system the 8259A must be on the lower 8 bits of the data bus. Note that the 8259A could also be interfaced on the system bus.

Although there are some basic similarities, the actual processing of interrupts with an 8086/8088 is different than an 8080A or 8085A. When an interrupt request is present and interrupts are enabled, the 8086/8088 enters its interrupt acknowledge machine cycle. The interrupt acknowledge machine cycle pushes the flag registers onto the stack (as in a PUSHF instruction). It then clears the IF flag which disables interrupts. The contents of

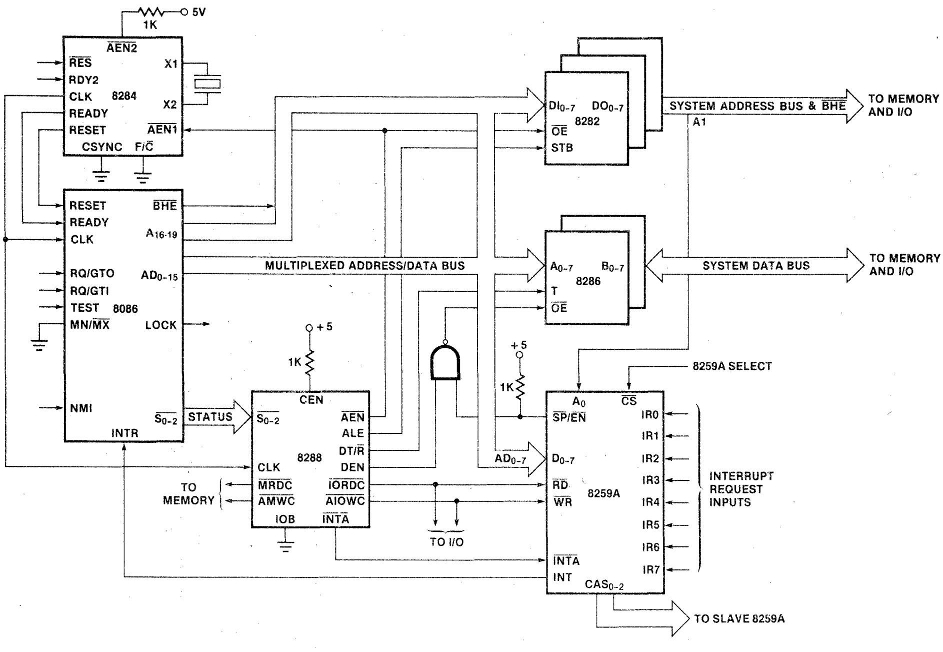


Figure 4. MSC-86 8259A Basic Configuration Example (8086 In Max. Mode)

both the code segment and the instruction pointer are then also pushed onto the stack. Thus, the stack retains the pre-interrupt flag status and pre-interrupt program location which are used to return from the service routine. The 8086/8088 then issues the first of two $\overline{INTA}$ pulses which signal the 8259A that the 8086/8088 has honored its interrupt request. If the 8086/8088 is used in its "MIN Mode" the $\overline{INTA}$ signal is available from the 8086/8088 on its $\overline{INTA}$ pin. If the 8086/8088 is used in the "MAX Mode" the $\overline{INTA}$ signal is available via the 8288 Bus Controller $\overline{INTA}$ pin. Additionally, in the "MAX Mode" the 8086/8088 LOCK pin goes low during the interrupt acknowledge sequence. The LOCK signal can be used to indicate to other system bus masters not to gain control of the system bus during the interrupt acknowledge sequence. A "HOLD" request won't be honored while LOCK is low.

The 8259A is now ready to vector program execution to the corresponding service routine. This is done during the sequence of the two $\overline{INTA}$ pulses issued by the 8086/8088. Unlike operation with the 8080A or 8085A, the 8259A doesn't place a CALL instruction and the starting address of the service routine on the data bus. Instead, the first $\overline{INTA}$ pulse is used only to signal the 8259A of the honored request. The second $\overline{INTA}$ pulse causes the 8259A to place a single interrupt-vector byte onto the data bus. Not used as a direct address, this interrupt-vector byte pertains to one of 256 interrupt "types" supported by the 8086/8088 memory. Program execution is vectored to the corresponding service routine by the contents of a specified interrupt type.

All 256 interrupt types are located in absolute memory locations 0 through 3FFH which make up the 8086/8088's interrupt-vector table. Each type in the interrupt-vector table requires 4 bytes of memory and stores a code segment address and an instruction pointer address. Figure 5 shows a block diagram of the interrupt-vector table. Locations 0 through 3FFH should be reserved for the interrupt-vector table alone. Furthermore, memory locations 00 through 7FH (types 0–31) are reserved for use by Intel Corporation for Intel hardware and software products. To maintain compatibility with present and future Intel products, these locations should not be used.

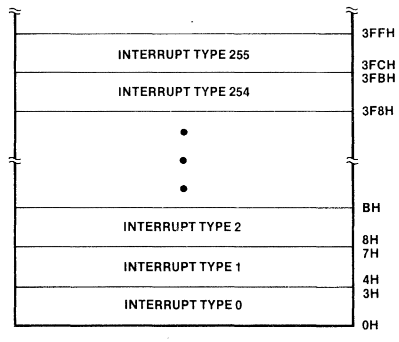


Figure 5. 8086/8088 Interrupt Vector Table

When the 8086/8088 receives an interrupt-vector byte from the 8259A, it multiplies its value by four to acquire the address of the interrupt type. For example, if the interrupt-vector byte specifies type 128 (80H), the vectored address in 8086/8088 memory is $4 \times 80H$, which equals 200H. Program execution is then vectored to the service routine whose address is specified by the code segment and instruction pointer values within type 128 located at 200H. To show how this is done, let's assume interrupt type 128 is to vector data to 8086/8088 memory location 2FF5FH. Figure 6 shows two possible ways to set values of the code segment and instruction pointer for vectoring to location 2FF5FH. Address generation by the code segment and instruction pointer is accomplished by an offset (they overlap). Of the total 20-bit address capability, the code segment can designate the upper 16 bits, the instruction pointer can designate the lower 16 bits.

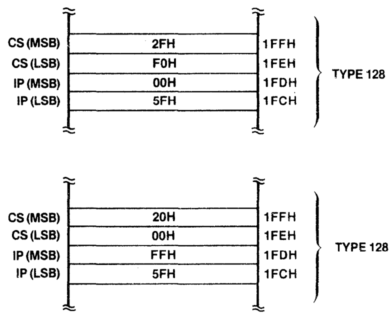


Figure 6. Two Examples of 8086/8088 Interrupt Type 128 Vectoring to Location 2FF5FH

When entering an interrupt service routine, those registers that are mutually used between the main program and service routine should be saved. The best way to do this is to "PUSH" each register used onto the stack immediately. The service routine can then "POP" each register off the stack in the same order when it is completed.

Once the service routine is completed the main program may be re-entered by using a IRET (Interrupt Return) instruction. The IRET instruction will pop the pre-interrupt instruction pointer, code segment and flags off the stack. Thus the main program will resume where it was interrupted with the same flag status regardless of changes in the service routine. Note especially that this includes the state of the IF flag, thus interrupts are re-enabled automatically when returning from the service routine.

Beside external interrupt generation from the INTR pin, the 8086/8088 is also able to invoke interrupts by software. Three interrupt instructions are provided: INT, INT (Type 3), and INTO. INT is a two byte instruction, the second byte selects the interrupt type. INT (Type 3) is a one byte instruction which selects interrupt Type 3. INTO is a conditional one byte interrupt instruction which selects interrupt Type 4 if the OF flag (trap on overflow) is set. All the software interrupts vector program execution as the hardware interrupts do.

For further information on 8086/8088 interrupt operation and internal interrupt structure refer to the MCS-86 User's Manual and the 8086 System Design application note.

2. 8259A FUNCTIONAL BLOCK DIAGRAM

A block diagram of the 8259A is shown in Figure 7. As can be seen from this figure, the 8259A consists of eight major blocks: the Interrupt Request Register (IRR), the In-Service Register (ISR), the Interrupt Mask Register (IMR), the Priority Resolver (PR), the cascade buffer/comparator, the data bus buffer, and logic blocks for control and read/write. We'll first go over the blocks directly related to interrupt handling, the IRR, ISR, IMR, PR, and the control logic. The remaining functional blocks are then discussed.

2.1 INTERRUPT REGISTERS AND CONTROL LOGIC

Basically, interrupt requests are handled by three "cascaded" registers: the Interrupt Request Register (IRR) is use to store all the interrupt levels requesting service; the In-Service Register (ISR) stores all the levels which are being serviced; and the Interrupt Mask Register (IMR) stores the bits of the interrupt lines to be masked. The Priority Resolver (PR) looks at the IRR, ISR and IMR, and determines whether an INT should be issued by the the control logic to the processor.

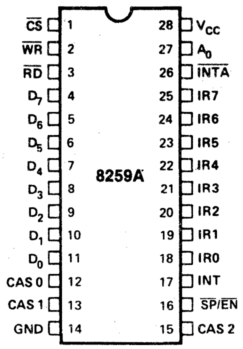
Figure 8 shows conceptually how the Interrupt Request (IR) input handles an interrupt request and how the various interrupt registers interact. The figure represents one of eight "daisy-chained" priority cells, one for each IR input.

The best way to explain the operation of the priority cell is to go through the sequence of internal events that happen when an interrupt request occurs. However, first, notice that the input circuitry of the priority cell allows for both level sensitive and edge sensitive IR inputs. Deciding which method to use is dependent on the particular application and will be discussed in more detail later.

When the IR input is in an inactive state (LOW), the edge sense latch is set. If edge sensitive triggering is selected, the "Q" output of the edge sense latch will arm the input gate to the request latch. This input gate will be disarmed after the IR input goes active (HIGH) and the interrupt request has been acknowledged. This disables the input from generating any further interrupts until it has returned low to re-arm the edge sense latch. If level sensitive triggering is selected, the "Q" output of the edge sense latch is rendered useless. This means the level of the IR input is in complete control of interrupt generation; the input won't be disarmed once acknowledged.

When an interrupt occurs on the IR input, it propagates through the request latch and to the PR (assuming the input isn't masked). The PR looks at the incoming requests and the currently in-service interrupts to ascertain whether an interrupt should be issued to the processor. Let's assume that the request is the only one incoming and no requests are presently in service. The PR then causes the control logic to pull the INT line to the processor high.

PIN CONFIGURATION



PIN NAMES

D_7-D_0	DATA BUS (BI-DIRECTIONAL)
RD	READ INPUT
WR	WRITE INPUT
A_0	COMMAND SELECT ADDRESS
CS	CHIP SELECT
CAS1-CAS0	CASCADE LINES
SP/EN	SLAVE PROGRAM/ENABLE BUFFER
INT	INTERRUPT OUTPUT
INTA	INTERRUPT ACKNOWLEDGE INPUT
IR0-IR7	INTERRUPT REQUEST INPUTS

BLOCK DIAGRAM

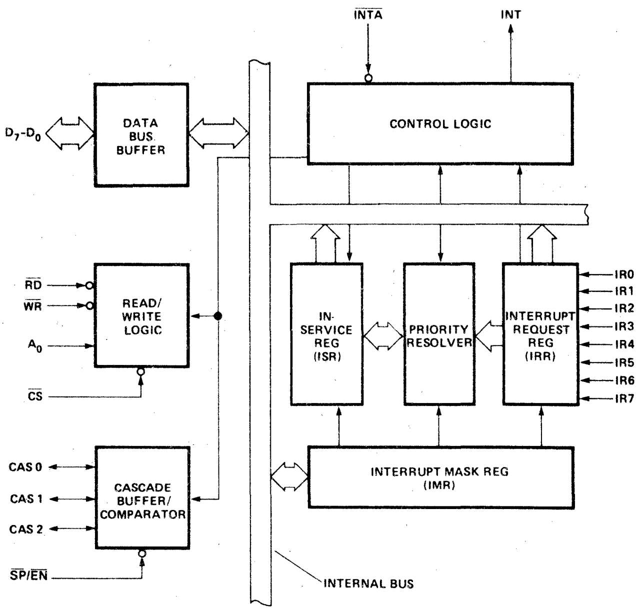
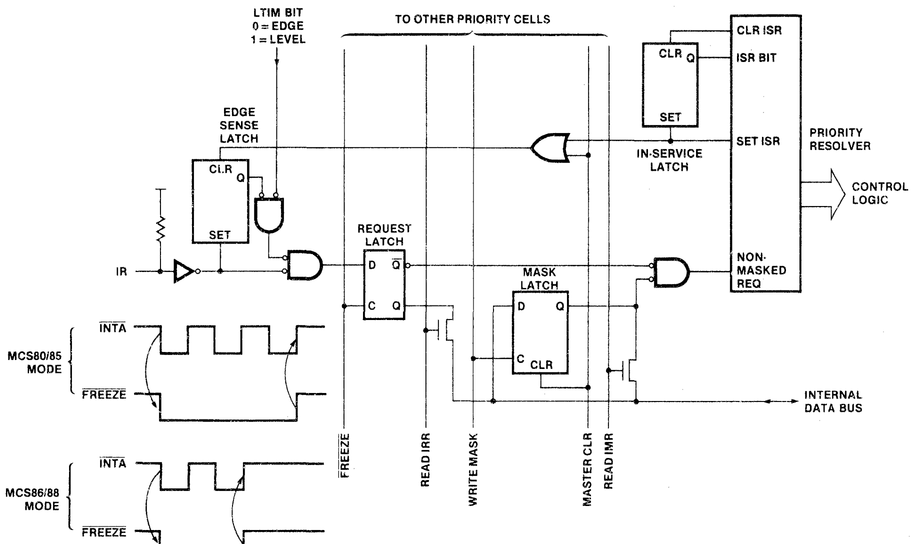


Figure 7. 8259A Block Diagram and Pin Configuration



NOTES
1. MASTER CLEAR ACTIVE ONLY DURING ICW1
2. FREEZE/ IS ACTIVE DURING INTA/ AND POLL SEQUENCES ONLY
3. TRUTH TABLE FOR D-LATCH

C	D	Q	OPERATION
1	Di	Di	FOLLOW
0	X	Qn−1	HOLD

Figure 8. Priority Cell

When the processor honors the INT pulse, it sends a sequence of INTA pulses to the 8259A (three for 8080A/8085A, two for 8086/8088). During this sequence the state of the request latch is frozen (note the INTA-freeze request timing diagram). Priority is again resolved by the PR to determine the appropriate interrupt vectoring which is conveyed to the processor via the data bus.

Immediately after the interrupt acknowledge sequence, the PR sets the corresponding bit in the ISR which simultaneously clears the edge sense latch. if edge sensitive triggering is used, clearing the edge sense latch also disarms the request latch. This inhibits the possibility of a still active IR input from propagating through the priority cell. The IR input must return to an inactive state, setting the edge sense latch, before another interrupt request can be recognized. If level sensitive triggering is used, however, clearing the edge sense latch has no affect on the request latch. The state of the request latch is entirely dependent upon the IR input level. Another interrupt will be generated immediately if the IR level is left active after its ISR bit has been reset. An ISR bit gets reset with an End-of-Interrupt (EOI) command issued in the service routine. End-of-interrupts will be covered in more detail later.

2.2 OTHER FUNCTIONAL BLOCKS

Data Bus Buffer

This three-state, bidirectional 8-bit buffer is used to interface the 8259A to the processor system data bus (via DB0–DB7). Control words, status information, and interrupt-vector data are transferred through the data bus buffer.

Read/Write Control Logic

The function of this block is to control the programming of the 8259A by accepting OUTput commands from the processor. It also controls the releasing of status onto the data bus by accepting INput commands from the processor. The initialization and operation command word registers which store the various control formats are located in this block. The $\overline{RD}$, $\overline{WR}$, $\overline{A0}$, and $\overline{CS}$ pins are used to control access to this block by the processor.

Cascade Buffer/Comparator

As mentioned earlier, multiple 8259A's can be combined to expand the number of interrupt levels. A master-slave relationship of cascaded 8259A's is used for the expansion. The $\overline{SP}/\overline{EN}$ and the CAS0–2 pins are used for operation of this block. The cascading of 8259A's is covered in depth in the "Operation of the 8259A" section of this application note.

2.3 PIN FUNCTIONS

Name	Pin #	I/O	Function
V_{CC}	28	I	+5V supply
GND	14	I	Ground

Name	Pin #	I/O	Function
$\overline{CS}$	1	I	*Chip Select:* A low on this pin enables $\overline{RD}$ and $\overline{WR}$ communication between the CPU and the 8259A. $\overline{INTA}$ functions are independent of $\overline{CS}$.
$\overline{WR}$	2	I	*Write:* A low on this pin when $\overline{CS}$ is low enables the 8259A to accept command words from the CPU.
$\overline{RD}$	3	I	*Read:* A low on this pin when $\overline{CS}$ is low enables the 8259A to release status onto the data bus for the CPU.
D7-D0	4-11	I/O	*Bidirectional Data Bus:* Control, status and interrupt-vector information is transferred via this bus.
CAS0-CAS2	12,13, 15	I/O	*Cascade Lines:* The CAS lines form a private 8259A bus to control a multiple 8259A structure. These pins are outputs for a master 8259A and inputs for a slave 8259A.
$\overline{SP}/\overline{EN}$	16	I/O	*Slave Program/Enable Buffer:* This is a dual function pin. When in the buffered mode it can be used as an output to control buffer transceivers ($\overline{EN}$). When not in the buffered mode it is used as an input to designate a master ($\overline{SP}$ = 1) or slave ($\overline{SP}$ = 0).
INT	17	O	*Interrupt:* This pin goes high whenever a valid interrupt request is asserted. It is used to interrupt the CPU, thus it is connected to the CPU's interrupt pin.
IR0-IR7	18-25	I	*Interrupt Requests:* Asynchronous inputs. An interrupt request can be generated by raising an IR input (low to high) and holding it high until it is acknowledged (edge triggered mode), or just by a high level on an IR input (level triggered mode).
$\overline{INTA}$	26	I	*Interrupt Acknowledge:* This pin is used to enable 8259A interrupt-vector data onto the data bus. This is done by a sequence of interrupt acknowledge pulses issued by the CPU.
A0	27	I	*A0 Address Line:* This pin acts in conjunction with the $\overline{CS}$, $\overline{WR}$, and $\overline{RD}$ pins. It is used by the 8259A to decipher between various command words the CPU writes and status the CPU wishes to read. It is typically connected to the CPU A0 address line (A1 for 8086/8088).

3. OPERATION OF THE 8259A

Interrupt operation of the 8259A falls under five main categories: vectoring, priorities, triggering, status, and cascading. Each of these categories use various modes and commands. This section will explain the operation of these modes and commands. For clarity of explanation, however, the actual programming of the 8259A isn't covered in this section but in "Programming the 8259A". Appendix A is provided as a cross reference between these two sections.

3.1 INTERRUPT VECTORING

Each IR input of the 8259A has an individual interrupt-vector address in memory associated with it. Designation of each address depends upon the initial programming of the 8259A. As stated earlier, the interrupt sequence and addressing of an MCS-80 and MCS-85 system differs from that of an MCS-86 and MCS-88 system. Thus, the 8259A must be initially programmed in either a MCS-80/85 or MCS-86/88 mode of operation to insure the correct interrupt vectoring.

MCS-80/85™ Mode

When programmed in the MCS-80/85 mode, the 8259A should only be used within an 8080A or an 8085A system. In this mode the 8080A/8085A will handle interrupts in the format described in the "MCS-80—8259A or MCS-85—8259A Overviews."

Upon interrupt request in the MCS-80/85 mode, the 8259A will output to the data bus the opcode for a CALL instruction and the address of the desired routine. This is in response to a sequence of three $\overline{INTA}$ pulses issued by the 8080A/8085A after the 8259A has raised INT high.

The first INTA pulse to the 8259A enables the CALL opcode "CD_H" onto the data bus. It also resolves IR priorities and effects operation in the cascade mode, which will be covered later. Contents of the first interrupt-vector byte are shown in Figure 9A.

During the second and third $\overline{INTA}$ pulses, the 8259A conveys a 16-bit interrupt-vector address to the 8080A/8085A. The interrupt-vector addresses for all eight levels are selected when initially programming the 8259A. However, only one address is needed for programming. Interrupt-vector addresses of IR0-IR7 are automatically set at equally spaced intervals based on the one programmed address. Address intervals are user definable to 4 or 8 bytes apart. If the service routine for a device is short it may be possible to fit the entire routine within an 8-byte interval. Usually, though, the service routines require more than 8 bytes. So, a 4-byte interval is used to store a Jump (JMP) instruction which directs the 8080A/8085A to the appropriate routine. The 8-byte interval maintains compatibility with current 8080A/8085A Restart (RST) instruction software, while the 4-byte interval is best for a compact jump table. If the 4-byte interval is selected, then the 8259A will automatically insert bits A0-A4. This leaves A5-A15 to be programmed by the user. If the 8-byte interval is selected, the 8259A will automatically insert bits A0-A5. This leaves only A6-A15 to be programmed by the user.

The LSB of the interrupt-vector address is placed on the data bus during the second $\overline{INTA}$ pulse. Figure 9B shows the contents of the second interrupt-vector byte for both 4 and 8-byte intervals.

The MSB of the interrupt-vector address is placed on the data bus during the third $\overline{INTA}$ pulse. Contents of the third interrupt-vector byte is shown in Figure 9C.

	D7	D6	D5	D4	D3	D2	D1	D0
CALL CODE	1	1	0	0	1	1	0	1

A. FIRST INTERRUPT VECTOR BYTE, MCS80/85 MODE

IR	Interval = 4							
	D7	D6	D5	D4	D3	D2	D1	D0
7	A7	A6	A5	1	1	1	0	0
6	A7	A6	A5	1	1	0	0	0
5	A7	A6	A5	1	0	1	0	0
4	A7	A6	A5	1	0	0	0	0
3	A7	A6	A5	0	1	1	0	0
2	A7	A6	A5	0	1	0	0	0
1	A7	A6	A5	0	0	1	0	0
0	A7	A6	A5	0	0	0	0	0

IR	Interval = 8							
	D7	D6	D5	D4	D3	D2	D1	D0
7	A7	A6	1	1	1	0	0	0
6	A7	A6	1	1	0	0	0	0
5	A7	A6	1	0	1	0	0	0
4	A7	A6	1	0	0	0	0	0
3	A7	A6	0	1	1	0	0	0
2	A7	A6	0	1	0	0	0	0
1	A7	A6	0	0	1	0	0	0
0	A7	A6	0	0	0	0	0	0

B. SECOND INTERRUPT VECTOR BYTE, MCS80/85 MODE

D7	D6	D5	D4	D3	D2	D1	D0
A15	A14	A13	A12	A11	A10	A9	A8

C. THIRD INTERRUPT VECTOR BYTE, MCS80/85 MODE

Figure 9. 9A–C. Interrupt-Vector Bytes for 8259A, MCS 80/85 Mode

MCS-86/88 Mode

When programmed in the MCS-86/88 mode, the 8259A should only be used within an MCS-86 or MCS-88 system. In this mode, the 8086/8088 will handle interrupts in the format described earlier in the "8259A—8086/8088 Overview".

Upon interrupt in the MCS-86/88 mode, the 8259A will output a single interrupt-vector byte to the data bus. This is in response to only two $\overline{INTA}$ pulses issued by the 8086/8088 after the 8259A has raised INT high.

The first $\overline{INTA}$ pulse is used only for set-up purposes internal to the 8259A. As in the MCS-80/85 mode, this set-up includes priority resolution and cascade mode operations which will be covered later. Unlike the MCS-80/85 mode, no CALL opcode is placed on the data bus.

The second $\overline{INTA}$ pulse is used to enable the single interrupt-vector byte onto the data bus. The 8086/8088 uses this interrupt-vector byte to select one of 256 interrupt "types" in 8086/8088 memory. Interrupt type selection for all eight IR levels is made when initially programming the 8259A. However, reference to only one interrupt type is needed for programming. The upper 5 bits of the interrupt vector byte are user definable. The lower 3 bits are automatically inserted by the 8259A depending upon the IR level.

Contents of the interrupt-vector byte for 8086/8088 type selection is put on the data bus during the second $\overline{INTA}$ pulse and is shown in Figure 10.

IR	D7	D6	D5	D4	D3	D2	D1	D0
7	T7	T6	T5	T4	T3	1	1	1
6	T7	T6	T5	T4	T3	1	1	0
5	T7	T6	T5	T4	T3	1	0	1
4	T7	T6	T5	T4	T3	1	0	0
3	T7	T6	T5	T4	T3	0	1	1
2	T7	T6	T5	T4	T3	0	1	0
1	T7	T6	T5	T4	T3	0	0	1
0	T7	T6	T5	T4	T3	0	0	0

Figure 10. Interrupt Vector Byte, MCS 86/88™ Mode

3.2 INTERRUPT PRIORITIES

A variety of modes and commands are available for controlling interrupt priorities of the 8259A. All of them are programmable, that is, they may be changed dynamically under software control. With these modes and commands, many possibilities are conceivable, giving the user enough versatility for almost any interrupt controlled application.

Fully Nested Mode

The fully nested mode of operation is a general purpose priority mode. This mode supports a multilevel-interrupt structure in which priority order of all eight IR inputs are arranged from highest to lowest.

Unless otherwise programmed, the fully nested mode is entered by default upon initialization. At this time, IR0 is assigned the highest priority through IR7 the lowest. The fully nested mode, however, is not confined to this IR structure alone. Once past initialization, other IR inputs can be assigned highest priority also, keeping the multilevel-interrupt structure of the fully nested mode. Figure 11A-C shows some variations of the priority structures in the fully nested mode.

IR LEVELS	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
PRIORITY	7	6	5	4	3	2	1	0

A

IR LEVELS	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
PRIORITY	4	3	2	1	0	7	6	5

B

IR LEVELS	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
PRIORITY	1	0	7	6	5	4	3	2

C

Figure 11. A-C. Some Variations of Priority Structure in the Fully Nested Mode

Further explanation of the fully nested mode, in this section, is linked with information of general 8259A interrupt operations. This is done to ease explanation to the user in both areas.

In general, when an interrupt is acknowledged, the highest priority request is determined from the IRR (Interrupt Request Register). The interrupt vector is then placed on the data bus. In addition, the corresponding bit in the ISR (In-Service Register) is set to designate the routine in service. This ISR bit remains set until an EOI (End-Of-Interrupt) command is issued to the 8259A. EOI's will be explained in greater detail shortly.

In the fully nested mode, while an ISR bit is set, all further requests of the same or lower priority are inhibited from generating an interrupt to the microprocessor. A higher priority request, though, can generate an interrupt, thus vectoring program execution to its service routine. Interrupts are only acknowledged, however, if the microprocessor has previously executed an "Enable Interrupts" instruction. This is because the interrupt request pin on the microprocessor gets disabled automatically after acknowledgement of any interrupt. The assembly language instructions used to enable interrupts are "EI" for 8080A/8085A and "STI" for 8086/8088. Interrupts can be disabled by using the instruction "DI" for 8080A/ 8085A and "CLI" for 8086/8088. When a routine is completed a "return" instruction is executed, "RET" for 8080A/8085A and "IRET" for 8086/8088.

Figure 12 illustrates the correct usage of interrupt related instructions and the interaction of interrupt levels in the fully nested mode.

Assuming the IR priority assignment for the example in Figure 12 is IR0 the highest through IR7 the lowest, the sequence is as follows. During the main program, IR3 makes a request. Since interrupts are enabled, the microprocessor is vectored to the IR3 service routine. During the IR3 routine, IR1 asserts a request. Since IR1 has higher priority than IR3, an interrupt is generated. However, it is not acknowledged because the microprocessor disabled interrupts in response to the IR3 interrupt. The IR1 interrupt is not acknowledged until the "Enable Interrupts" instruction is executed. Thus the IR3 routine has a "protected" section of code over which no interrupts (except non-maskable) are allowed. The IR1 routine has no such "protected" section since an "Enable Interrupts" instruction is the first one in its service routine. Note that in this example the IR1 request must stay high until it is acknowledged. This is covered in more depth in the "Interrupt Triggering" section.

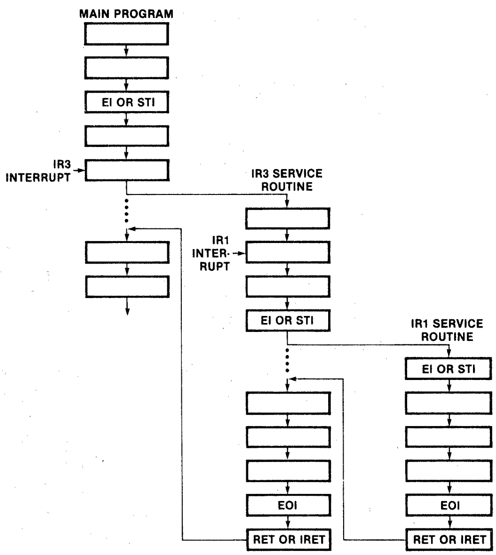


Figure 12. Fully Nested Mode Example (MCS 80/85 or MCS 86/88)

What is happening to the ISR register? While in the main program, no ISR bits are set since there aren't any interrupts in service. When the IR3 interrupt is acknowledged, the ISR3 bit is set. When the IR1 interrupt is acknowledged, both the ISR1 and the ISR3 bits are set, indicating that neither routine is complete. At this time, only IR0 could generate an interrupt since it is the only input with a higher priority than those previously in service. To terminate the IR1 routine, the routine must inform the 8259A that it is complete by resetting its ISR bit. It does this by executing an EOI command. A "return" instruction then transfers execution back to the IR3 routine. This allows IR0-IR2 to interrupt the IR3 routine again, since ISR3 is the highest ISR bit set. No further interrupts occur in the example so the EOI command resets ISR3 and the "return" instruction causes the main program to resume at its pre-interrupt location, ending the example.

A single 8259A is essentially always in the fully nested mode unless certain programming conditions disturb it. The following programming conditions can cause the 8259A to go out of the high to low priority structure of the fully nested mode.

- The automatic EOI mode
- The special mask mode
- A slave with a master not in the special fully nested mode

These modes will be covered in more detail later, however, they are mentioned now so the user can be aware of them. As long as these program conditions aren't inacted, the fully nested mode remains undisturbed.

End of Interrupt

Upon completion of an interrupt service routine the 8259A needs to be notified so its ISR can be updated. This is done to keep track of which interrupt levels are in the process of being serviced and their relative priorities. Three different End-Of-Interrupt (EOI) formats are available for the user. These are: the non-specific EOI command, the specific EOI command, and the automatic EOI Mode. Selection of which EOI to use is dependent upon the interrupt operations the user wishes to perform.

Non-Specific EOI Command

A non-specific EOI command sent from the microprocessor lets the 8259A know when a service routine has been completed, without specification of its exact interrupt level. The 8259A automatically determines the interrupt level and resets the correct bit in the ISR.

To take advantage of the non-specific EOI the 8259A must be in a mode of operation in which it can predetermine in-service routine levels. For this reason the non-specific EOI command should only be used when the most recent level acknowledged and serviced is always the highest priority level. When the 8259A receives a non-specific EOI command, it simply resets the highest priority ISR bit, thus confirming to the 8259A that the highest priority routine of the routines in service is finished.

The main advantage of using the non-specific EOI command is that IR level specification isn't necessary as in the "Specific EOI Command", covered shortly. However, special consideration should be taken when deciding to use the non-specific EOI. Here are two program conditions in which it is best not used:

- Using the set priority command within an interrupt service routine.
- Using a special mask mode.

These conditions are covered in more detail in their own sections, but are listed here for the users reference.

Specific EOI Command

A specific EOI command sent from the microprocessor lets the 8259A know when a service routine of a particular interrupt level is completed. Unlike a non-specific EOI command, which automatically resets the highest priority ISR bit, a specific EOI command specifies an exact ISR bit to be reset. One of the eight IR levels of the 8259A can be specified in the command.

The reason the specific EOI command is needed, is to reset the ISR bit of a completed service routine whenever the 8259A isn't able to automatically determine it. An example of this type of situation might be if the priorities of the interrupt levels were changed during an interrupt routine ("Specific Rotation"). In this case, if any other routines were in service at the same time, a non-specific EOI might reset the wrong ISR bit. Thus the specific EOI command is the best bet in this case, or for that matter, any time in which confusion of interrupt priorities may exist. The specific EOI command can be used in all conditions of 8259A operation, including those that prohibit non-specific EOI command usage.

Automatic EOI Mode

When programmed in the automatic EOI mode, the microprocessor no longer needs to issue a command to notify the 8259A it has completed an interrupt routine. The 8259A accomplishes this by performing a non-specific EOI automatically at the trailing edge of the last $\overline{INTA}$ pulse (third pulse in MCS-80/85, second in MCS-86).

The obvious advantage of the automatic EOI mode over the other EOI command is no command has to be issued. In general, this simplifies programming and lowers code requirements within interrupt routines.

However, special consideration should be taken when deciding to use the automatic EOI mode because it disturbs the fully nested mode. In the automatic EOI mode the ISR bit of a routine in service is reset right after it's acknowledged, thus leaving no designation in the ISR that a sevice routine is being executed. If any interrupt request occurs during this time (and interrupts are enabled) it will get serviced regardless of its priority, low or high. The problem of "over nesting" may also happen in this situation. "Over nesting" is when an IR input keeps interrupting its own routine, resulting in unnecessary stack pushes which could fill the stack in a worst case condition. This is not usually a desired form of operation!

So what good is the automatic EOI mode with problems like those just covered? Well, again, like the other EOIs, selection is dependent upon the application. If interrupts are controlled at a predetermined rate, so as not to cause the problems mentioned above, the automatic EOI mode works perfect just the way it is. However, if interrupts happen sporadically at an indeterminate rate, the automatic EOI mode should only be used under the following guideline:

- When using the automatic EOI mode with an indeterminate interrupt rate, the microprocessor should keep its interrupt request input disabled during execution of service routines.

By doing this, higher priority interrupt levels will be serviced only after the completion of a routine in service. This guideline restores the fully nested structure in regards to the IRR; however, a routine in-service can't be interrupted.

Automatic Rotation — Equal Priority

Automatic rotation of priorities serves in applications where the interrupting devices are of equal priority, such as communications channels. The concept is that once a peripheral is serviced, all other equal priority peripherals should be given a chance to be serviced before the original peripheral is serviced again. This is accomplished by automatically assigning a peripheral the lowest priority after being serviced Thus, in worst case, the device would have to wait until all other devices are serviced before being serviced again.

There are two methods of accomplishing automatic rotation. One is used in conjunction with the non-specific EOI, "rotate on non-specific EOI command". The other is used with the automatic EOI mode, "rotate in automatic EOI mode".

Rotate on Non-Specific EOI Command

When the rotate on non-specific EOI command is issued, the highest ISR bit is reset as in a normal non-specific EOI command. After it's reset though, the corresponding IR level is assigned lowest priority. Other IR priorities rotate to conform to the fully nested mode based on the newly assigned low priority

Figures 13A and B show how the rotate on non-specific EOI command effects the interrupt priorities. Let's assume the IR priorities were assigned with IR0 the highest and IR7 the lowest, as in 13A. IR6 and IR4 are already in service but neither is completed. Being the higher priority routine, IR4 is necessarily the routine being executed. During the IR4 routine a rotate on non-specific EOI command is executed. When this happens, bit 4 in the ISR is reset. IR4 then becomes the lowest priority and IR5 becomes the highest as in 13B.

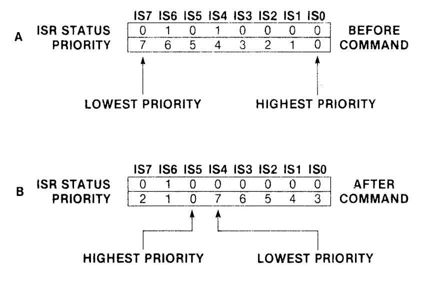


Figure 13. A-B. Rotate on Non-specific EOI Command Example

Rotate in Automatic EOI Mode

The rotate in automatic EOI mode works much like the rotate on non-specific EOI command. The main difference is that priority rotation is done automatically after the last $\overline{\text{INTA}}$ pulse of an interrupt request. To enter or exit this mode a rotate-in-automatic-EOI set command and rotate-in-automatic-EOI clear command is provided. After that, no commands are needed as with the normal automatic EOI mode. However, it must be remembered, when using any form of the automatic EOI mode, special consideration should be taken. Thus, the guideline for the automatic EOI mode also stands for the rotate in automatic EOI mode.

Specific Rotation — Specific Priority

Specific rotation gives the user versatile capabilities in interrupt controlled operations. It serves in those applications in which a specific device's interrupt priority must be altered. As opposed to automatic rotation which automatically sets priorities, specific rotation is completely user controlled. That is, the user selects which interrupt level is to receive lowest or highest priority. This can be done during the main program or within interrupt routines. Two specific rotation commands are available to the user, the "set priority command" and the "rotate on specific EOI command."

Set Priority Command

The set priority command allows the programmer to assign an IR level the lowest priority. All other interrupt levels will conform to the fully nested mode based on the newly assigned low priority.

An example of how the set priority command works is shown in Figures 14A and 14B. These figures show the status of the ISR and the relative priorities of the interrupt levels before and after the set priority command. Two interrupt routines are shown to be in service in Figure 14A. Since IR2 is the highest priority, it is necessarily the routine being executed. During the IR2 routine, priorities are altered so that IR5 is the highest. This is done simply by issuing the set priority command to the 8259A. In this case, the command specifies IR4 as being the lowest priority. The result of this set priority command is shown in Figure 14B. Even though IR7 now has higher priority than IR2, it won't be acknowledged until the IR2 routine is finished (via EOI). This is because priorities are only resolved upon an interrupt request or an interrupt acknowledge sequence. If a higher priority request occurs during the IR2 routine, then priorities are resolved and the highest will be acknowledged.

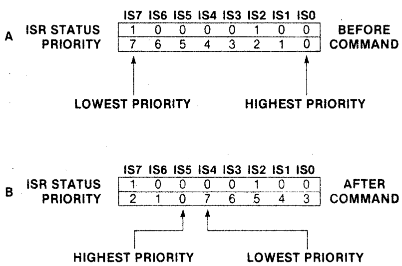


Figure 14. A-B. Set Priority Command Example

When completing a service routine in which the set priority command is used, the correct EOI must be issued. The non-specific EOI command shouldn't be used in the same routine as a set priority command. This is because the non-specific EOI command resets the highest ISR bit, which, when using the set priority command, is not always the most recent routine in service. The automatic EOI mode, on the other hand, can be used with the set priority command. This is because it automatically performs a non-specific EOI before the set priority command can be issued. The specific EOI command is the best bet in most cases when using the set priority command within a routine. By resetting the specific ISR bit of a routine being completed, confusion is eliminated.

Rotate on Specific EOI Command

The rotate on specific EOI command is literally a combination of the set priority command and the specific EOI command. Like the set priority command, a specified IR level is assigned lowest priority. Like the specific EOI command, a specified level will be reset in the ISR. Thus the rotate on specific EOI command accomplishes both tasks in only one command.

If it is not necessary to change IR priorities prior to the end of an interrupt routine, then this command is advantageous. For an EOI command must be executed anyway (unless in the automatic EOI mode), so why not do both at the same time?

Interrupt Masking

Disabling or enabling interrupts can be done by other means than just controlling the microprocessor's interrupt request pin. The 8259A has an IMR (Interrupt Mask Register) which enhances interrupt control capabilities. Rather than all interrupts being disabled or enabled at the same time, the IMR allows individual IR masking. The IMR is an 8-bit register, bits 0–7 directly correspond to IR0–IR7. Any IR input can be masked by writing to the IMR and setting the appropriate bit. Likewise, any IR input can be enabled by clearing the correct IMR bit.

There are various uses for masking off individual IR inputs. One example is when a portion of a main routine wishes only to be interrupted by specific interrupts. Another might be disabling higher priority interrupts for a portion of a lower priority service routine. The possibilities are many.

When an interrupt occurs while its IMR bit is set, it isn't necessarily forgotten. For, as stated earlier, the IMR acts only on the output of the IRR. Even with an IR input masked it is still possible to set the IRR. Thus, when resetting an IMR, if its IRR bit is set it will then generate an interrupt. This is providing, of course, that other priority factors are taken into consideration and the IR request remains active. If the IR request is removed before the IMR is reset, no interrupt will be acknowledged.

Special Mask Mode

In various cases, it may be desirable to enable interrupts of a lower priority than the routine in service. Or, in other words, allow lower priority devices to generate interrupts. However, in the fully nested mode, all IR levels of priority below the routine in service are inhibited. So what can be done to enable them?

Well, one method could be using an EOI command before the actual completion of a routine in service. But beware, doing this may cause an "over nesting" problem, similar to in the automatic EOI mode. In addition, resetting an ISR bit is irreversible by software control, so lower priority IR levels could only be later disabled by setting the IMR.

A much better solution is the special mask mode. Working in conjunction with the IMR, the special mask mode enables interrupts from all levels except the level in service. This is done by masking the level that is in service and then issuing the special mask mode command. Once the special mask mode is set, it remains in effect until reset.

Figure 15 shows how to enable lower priority interrupts by using the Special Mask Mode (SMM). Assume that IR0 has highest priority when the main program is interrupted by IR4. In the IR4 service routine an enable interrupt instruction is executed. This only allows higher priority interrupt requests to interrupt IR4 in the normal fully nested mode. Further in the IR4 routine, bit 4 of the IMR is masked and the special mask mode is entered. Priority operation is no longer in the fully nested mode. All interrupt levels are enabled except for IR4. To leave the special mask mode, the sequence is executed in reverse.

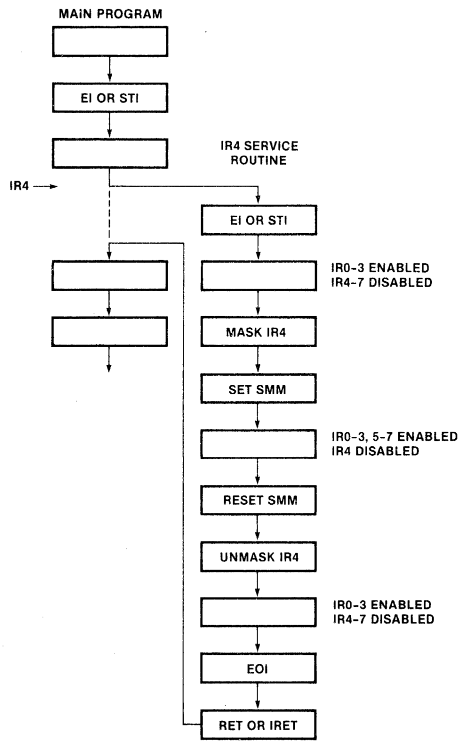


Figure 15. Special Mask Mode Example (MCS 80/85 or MCS 86/88)

Precautions must be taken when exiting an interrupt service routine which has used the special mask mode. A non-specific EOI command can't be used when in the special mask mode. This is because a non-specific won't clear an ISR bit of an interrupt which is masked when in the special mask mode. In fact, the bit will appear invisible. If the special mask mode is cleared before an EOI command is issued a non-specific EOI command can be used. This could be the case in the example shown in Figure 15, but, to avoid any confusion it's best to use the specific EOI whenever using the special mask mode.

It must be remembered that the special mask mode applies to all masked levels when set. Take, for instance, IR1 interrupting IR4 in the previous example. If this happened while in the special mask mode, and the IR1 routine masked itself, all interrupts would be enabled except IR1 and IR4 which are masked.

3.3 INTERRUPT TRIGGERING

There are two classical ways of sensing an active interrupt request: a level sensitive input or an edge sensitive input. The 8259A gives the user the capability for either method with the edge triggered mode and the level triggered mode. Selection of one of these interrupt triggering methods is done during the programmed initialization of the 8259A.

Level Triggered Mode

When in the level triggered mode the 8259A will recognize any active (high) level on an IR input as an interrupt request. If the IR input remains active after an EOI command has been issued (resetting its ISR bit), another interrupt will be generated. This is providing of course, the processor INT pin is enabled. Unless repetitious interrupt generation is desired, the IR input must be brought to an inactive state before an EOI command is issued in its service routine. However, it must not go inactive so soon that it disobeys the necessary timing requirements shown in Figure 16. Note that the request on the IR input must remain until after the falling edge of the first $\overline{INTA}$ pulse. If on any IR input, the request goes inactive before the first $\overline{INTA}$ pulse, the 8259A will respond as if IR7 was active. In any design in which there's a possibility of this happening, the IR7 default feature can be used as a safeguard. This can be accomplished by using the IR7 routine as a "clean-up routine" which might recheck the 8259A status or merely return program execution to its pre-interrupt location.

Depending upon the particular design and application, the level triggered mode has a number of uses. For one, it provides for repetitious interrupt generation. This is useful in cases when a service routine needs to be continually executed until the interrupt request goes inactive. Another possible advantage of the level triggered mode is it allows for "wire-OR'ed" interrupt requests. That is, a number of interrupt requests using the same IR input. This can't be done in the edge triggered mode, for if a device makes an interrupt request while the IR input is high (from another request), its transition will be "shadowed". Thus the 8259A won't recognize further interrupt requests because its IR input is already high. Note that when a "wire-OR'ed" scheme is used, the ac-

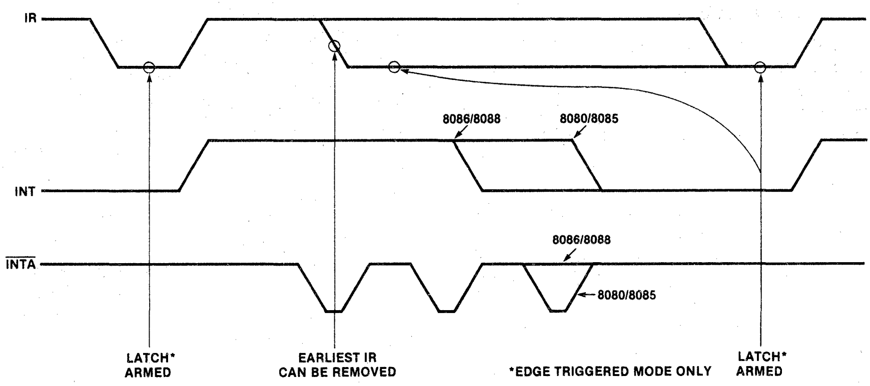


Figure 16. IR Triggering Timing Requirements

tual requesting device has to be determined by the software in the service routine.

Caution should be taken when using the automatic EOI mode and the level triggered mode together. Since in the automatic EOI mode an EOI is automatically performed at the end of the interrupt acknowledge sequence, if the processor enables interrupts while an IR input is still high, an interrupt will occur immediately. To avoid this situation interrupts should be kept disabled until the end of the service routine or until the IR input returns low.

Edge Triggered Mode

When in the edge triggered mode, the 8259A will only recognize interrupts if generated by an inactive (low) to active (high) transition on an IR input. The edge triggered mode incorporates an edge lockout method of operation. This means that after the rising edge of an interrupt request and the acknowledgement of the request, the positive level of the IR input won't generate further interrupts on this level. The user needn't worry about quickly removing the request after acknowledgement in fear of generating further interrupts as might be the case in the level triggered mode. Before another interrupt can be generated the IR input must return to the inactive state.

Referring back to Figure 16, the timing requirements for interrupt triggering is shown. Like the level triggered mode, in the edge triggered mode the request on the IR input must remain active until after the falling edge of the first $\overline{\text{INTA}}$ pulse for that particular interrupt. Unlike the level triggered mode, though, after the interrupt request is acknowledged its IRR latch is disarmed. Only after the IR input goes inactive will the IRR latch again become armed, making it ready to receive another interrupt request (in the level triggered mode, the IRR latch is always armed). Because of the way the edge triggered mode functions, it is best to use a positive level with a negative pulse to trigger the IR requests. With this type of input, the trailing edge of the pulse causes the interrupt and the maintained positive level meets the necessary timing requirements (remaining high until after the interrupt acknowledge occurs). Note that the IR7 default feature mentioned in the "level triggered mode" section also works for the edge triggered mode.

Depending upon the particular design and application, the edge triggered mode has various uses. Because of its edge lockout operation, it is best used in those applications where repetitious interrupt generation isn't desired. It is also very useful in systems where the interrupt request is a pulse (this should be in the form of a negative pulse to the 8259A). Another possible advantage is that it can be used with the automatic EOI mode without the cautions in the level triggered mode. Overall, in most cases, the edge triggered mode simplifies operation for the user, since the duration of the interrupt request at a positive level is not usually a factor.

3.4 INTERRUPT STATUS

By means of software control, the user can interrogate the status of the 8259A. This allows the reading of the internal interrupt registers, which may prove useful for interrupt control during service routines. It also provides for a modified status poll method of device monitoring, by using the poll command. This makes the status of the internal IR inputs available to the user via software control. The poll command offers an alternative to the interrupt vector method, especially for those cases when more than 64 interrupts are needed.

Reading Interrupt Registers

The contents of each 8-bit interrupt register, IRR, ISR, and IMR, can be read to update the user's program on the present status of the 8259A. This can be a versatile tool in the decision making process of a service routine, giving the user more control over interrupt operations. Before delving into the actual process of reading the registers, let's briefly review their general descriptions:

IRR (Interrupt Request Register)	Specifies all interrupt levels requesting service.
ISR (In-Service Register)	Specifies all interrupt levels which are being serviced.
IMR (Interrupt Mask Register)	Specifies all interrupt levels that are masked.

To read the contents of the IRR or ISR, the user must first issue the appropriate read register command (read IRR or read ISR) to the 8259A. Then by applying a $\overline{RD}$ pulse to the 8259A (an INput instruction), the contents of the desired register can be acquired. There is no need to issue a read register command every time the IRR or ISR is to be read. Once a read register command is received by the 8259A, it "remembers" which register has been selected. Thus, all that is necessary to read the contents of the same register more than once is the $\overline{RD}$ pulse and the correct addressing (A0 = 0, explained in "Programming the 8259A"). Upon initialization, the selection of registers defaults to the IRR. Some caution should be taken when using the read register command in a system that supports several levels of interrupts. If the higher priority routine causes an interrupt between the read register command and the actual input of the register contents, there's no guarantee that the same register will be selected when it returns. Thus it is best in such cases to disable interrupts during the operation.

Reading the contents of the IMR is different than reading the IRR or ISR. A read register command is not necessary when reading the IMR. This is because the IMR can be addressed directly for both reading and writing. Thus all that the 8259A requires for reading the IMR is a $\overline{RD}$ pulse and the correct addressing (A0 = 1, explained in "Programming the 8259A").

Poll Command

As mentioned towards the beginning of this application note, there are two methods of servicing peripherals: status polling and interrupt servicing. For most applications the interrupt service method is best. This is because it requires the least amount of CPU time, thus increasing system throughput. However, for certain applications, the status poll method may be desirable.

For this reason, the 8259A supports polling operations with the poll command. As opposed to the conventional method of polling, the poll command offers improved device servicing and increased throughput. Rather than having the processor poll each peripheral in order to find the actual device requiring service, the processor polls the 8259A. This allows the use of all the previously mentioned priority modes and commands. Additionally, both polled and interrupt methods can be used within the same program.

To use the poll command the processor must first have its interrupt request pin disabled. Once the poll command is issued, the 8259A will treat the next ($\overline{CS}$ qualified) $\overline{RD}$ pulse issued to it (an INput instruction) as an interrupt acknowledge. It will then set the appropriate bit in the ISR, if there was an interrupt request, and enable a special word onto the data bus. This word shows whether an interrupt request has occurred and the highest priority level requesting service. Figure 17 shows the contents of the "poll word" which is read by the processor. Bits W0-W2 convey the binary code of the highest priority level requesting service. Bit I designates whether or not an interrupt request is present. If an interrupt request is present, bit I will equal 1. If there isn't an interrupt request at all, bit I will equal 0 and bits W0-W2 will be set to ones. Service to the requesting device is achieved by software decoding the poll word and branching to the appropriate service routine. Each time the 8259A is to be polled, the poll command must be written before reading the poll word.

The poll command is useful in various situations. For instance, it's a good alternative when memory is very limited, because an interrupt-vector table isn't needed. Another use for the poll command is when more than 64 interrupt levels are needed (64 is the limit when cascading 8259's). The only limit of interrupts using the poll command is the number of 8259's that can be addressed in a particular system. Still another application of the poll command might be when the INT or $\overline{INTA}$ signals are not available. This might be the case in a large system where a processor on one card needs to use an 8259A on a different card. In this instance, the poll command is the only way to monitor the interrupt devices and still take advantage of the 8259A's prioritizing features. For those cases when the 8259A is using the poll command only and not the interrupt method, each 8259A must receive an initialization sequence (interrupt vector). This must be done even though the interrupt vector features of the 8259A are not used. In this case, the interrupt vector specified in the initialization sequence could be a "fake".

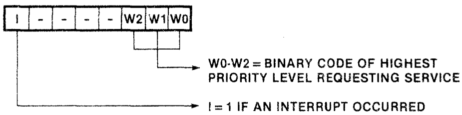


Figure 17. Poll Word

3.5 INTERRUPT CASCADING

As mentioned earlier, more than one 8259A can be used to expand the priority interrupt scheme to up to 64 levels without additional hardware. This method for expanded interrupt capability is called "cascading". The 8259A supports cascading operations with the cascade mode. Additionally, the special fully nested mode and the buffered mode are available for increased flexibility when cascading 8259A's in certain applications.

Cascade Mode

When programmed in the cascade mode, basic operation consists of one 8259A acting as a master to the others which are serving as slaves. Figure 18 shows a system containing a master and two slaves, providing a total of 22 interrupt levels.

A specific hardware set-up is required to establish operation in the cascade mode. With Figure 18 as a reference, note that the master is designated by a high on the $\overline{SP}/\overline{EN}$ pin, while the $\overline{SP}/\overline{EN}$ pins of the slaves are grounded (this can also be done by software, see buffered mode). Additionally, the INT output pin of each slave is connected to an IR input pin of the master. The CAS0-2 pins for all 8259A's are paralleled. These pins act as outputs when the 8259A is a master and as inputs for the slaves. Serving as a private 8259A bus, they control which slave has control of the system bus for interrupt vectoring operation with the processor. All other pins are connected as in normal operation (each 8259A receives an INTA pulse).

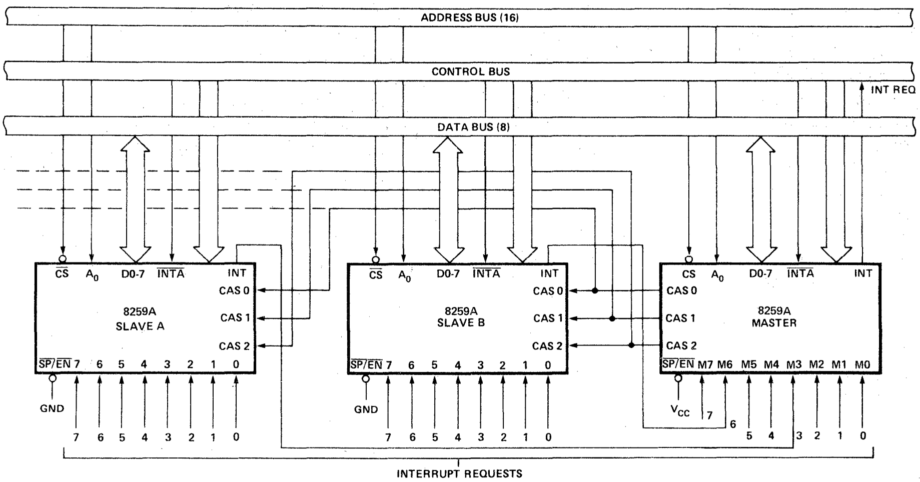


Figure 18. Cascaded 8259A'S 22 Interrupt Levels

Besides hardware set-up requirements, all 8259A's must be software programmed to work in the cascade mode. Programming the cascade mode is done during the initialization of each 8259A. The 8259A that is selected as master must receive specification during its initialization as to which of its IR inputs are connected to a slave's INT pin. Each slave 8259A, on the other hand, must be designated during its initialization with an ID (0 through 7) corresponding to which of the master's IR inputs its INT pin is connected to. This is all necessary so the CAS0-2 pins of the masters will be able to address each individual slave. Note that as in normal operation, each 8259A must also be initialized to give its IR inputs a unique interrupt vector. More detail on the necessary programming of the cascade mode is explained in "Programming the 8259A".

Now, with background information on both hardware and software for the cascade mode, let's go over the sequence of events that occur during a valid interrupt request from a slave. Suppose a slave IR input has received an interrupt request. Assuming this request is higher priority than other requests and in-service levels on the slave, the slave's INT pin is driven high. This signals the master of the request by causing an interrupt request on a designated IR pin of the master. Again, assuming that this request to the master is higher priority than other master requests and in-service levels (possibly from other slaves), the master's INT pin is pulled high, interrupting the processor.

The interrupt acknowledge sequence appears to the processor the same as the non-cascading interrupt acknowledge sequence; however, it's different among the 8259A's. The first $\overline{INTA}$ pulse is used by all the 8259A's for internal set-up purposes and, if in the 8080/8085 mode, the master will place the CALL opcode on the data bus. The first $\overline{INTA}$ pulse also signals the master to place the requesting slave's ID code on the CAS lines. This turns control over to the slave for the rest of the interrupt acknowledge sequence, placing the appropriate pre-programmed interrupt vector on the data bus, completing the interrupt request.

During the interrupt acknowledge sequence, the corresponding ISR bit of both the master and the slave get set. This means two EOI commands must be issued (if not in the automatic EOI mode), one for the master and one for the slave.

Special consideration should be taken when mixed interrupt requests are assigned to a master 8259A; that is, when some of the master's IR inputs are used for slave interrupt requests and some are used for individual interrupt requests. In this type of structure, the master's IR0 must not be used for a slave. This is because when an IR input that isn't initialized as a slave receives an interrupt request, the CAS0-2 lines won't be activated, thus staying in the default condition addressing for IR0 (slave IR0). If a slave is connected to the master's IR0 when a non-slave interrupt occurs on another master IR input, erroneous conditions may result. Thus IR0 should be the last choice when assigning slaves to IR inputs.

Special Fully Nested Mode

Depending on the application, changes in the nested structure of the cascade mode may be desired. This is because the nested structure of a slave 8259A differs from that of the normal fully nested mode. In the cascade mode, if a slave receives a higher priority interrupt request than one which is in service (through the same slave), it won't be recognized by the master. This is because the master's ISR bit is set, ignoring all requests of equal or lower priority. Thus, in this case, the higher priority slave interrupt won't be serviced until after the master's ISR bit is reset by an EOI command. This is most likely after the completion of the lower priority routine.

If the user wishes to have a truly fully nested structure within a slave 8259A, the special fully nested mode should be used. The special fully nested mode is programmed in the master only. This is done during the master's initialization. In this mode the master will ignore only those interrupt requests of lower priority than the set ISR bit and will respond to all requests of equal or higher priority. Thus if a slave receives a higher priority request than one in service, it will be recognized. To insure proper interrupt operation when using the special fully nested mode, the software must determine if any other slave interrupts are still in service before issuing an EOI command to the master. This is done by resetting the appropriate slave ISR bit with an EOI and then reading its ISR. If the ISR contains all zeros, there aren't any other interrupts from the slave in service and an EOI command can be sent to the master. If the ISR isn't all zeros, an EOI command shouldn't be sent to the master. Clearing the master's ISR bit with an EOI command while there are still slave interrupts in service would allow lower priority interrupts to be recognized at the master. An example of this process is shown in the second application in the "Applications Examples" section.

Buffered Mode

The buffered mode is useful in large systems where buffering is required on the data bus. Although not limited to only 8259A cascading, it's most pertinent in this use. In the buffered mode, whenever the 8259A's data bus output is enabled, its $\overline{SP}/\overline{EN}$ pin will go low. This signal can be used to enable data transfer through a buffer transceiver in the required direction.

Figure 19 shows a conceptual diagram of three 8259A's in cascade, each slave is controlling an individual 8286 8-bit bidirectional bus driver by means of the buffered mode. Note the pull-up on the $\overline{SP}/\overline{EN}$. It is used to enable data transfer to the 8259A for its initial programming. When data transfer is to go from the 8259A to the processor, $\overline{SP}/\overline{EN}$ will go low; otherwise, it will be high.

A question should arise, however, from the fact that the $\overline{SP}/\overline{EN}$ pin is used to designate a master from a slave; how can it be used for both master-slave selection and buffer control? The answer to this is the provision for software programmable master-slave selection when in the buffer mode. The buffered mode is selected during each 8259A's initialization. At the same time, the user can assign each individual 8259A as a master or slave (see "Programming the 8259A").

4. PROGRAMMING THE 8259A

Programming the 8259A is accomplished by using two types of command words: Initialization Command Words (ICWs) and Operational Command Words (OCWs). All the modes and commands explained in the previous section, "Operation of the 8259A", are programmable using the ICWs and OCWs (see Appendix A for cross reference). The ICWs are issued from the processor in a sequential format and are used to set-up the 8259A in an initial state of operation. The OCWs are issued as needed to vary and control 8259A operation.

Both ICWs and OCWs are sent by the processor to the 8259A via the data bus (8259A $\overline{CS}=0$, $\overline{WR}=0$). The 8259A distinguishes between the different ICWs and OCWs by the state of its A0 pin (controlled by processor addressing), the sequence they're issued in (ICWs only), and some dedicated bits among the ICWs and OCWs. Those bits which are dedicated are indicated so by fixed values (0 or 1) in the corresponding ICW or OCW programming formats which are covered shortly. Note, when issuing either ICWs or OCWs, the interrupt request pin of the processor should be disabled.

4.1 INITIALIZATION COMMAND WORDS (ICWs)

Before normal operation can begin, each 8259A in a system must be initialized by a sequence of two to four programming bytes called ICWs (Initialization Command Words). The ICWs are used to set-up the necessary conditions and modes for proper 8259A operation.

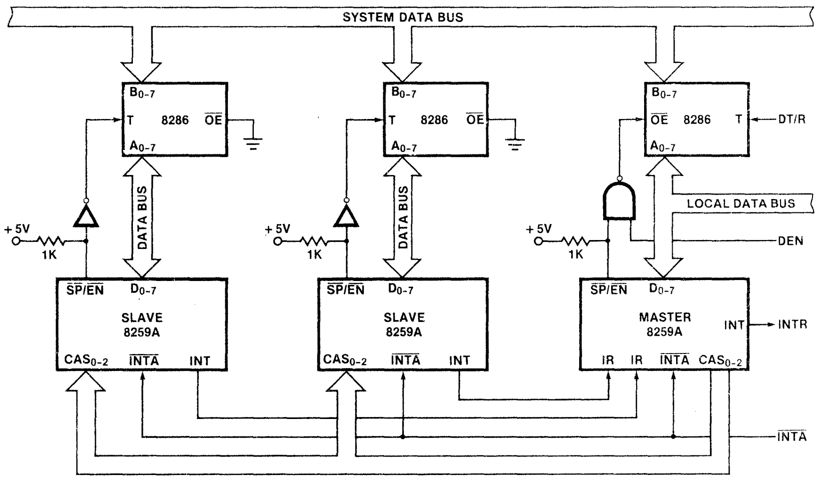


Figure 19. Cascade-Buffered Mode Example

Figure 20 shows the initialization flow of the 8259A. Both ICW1 and ICW2 must be issued for any form of 8259A operation. However, ICW3 and ICW4 are used only if designated so in ICW1. Determining the necessity and use of each ICW is covered shortly in individual groupings. Note that, once intialized, if any programming changes within the ICWs are to be made, the entire ICW sequence must be reprogrammed, not just an individual ICW.

Certain internal set-up conditions occur automatically within the 8259A after the first ICW has been issued. These are:

A. Sequencer logic is set to accept the remaining ICWs as designated in ICW1.

B. The ISR (In-Service Register) and IMR (Interrupt Mask Register) are both cleared.

C. The special mask mode is reset.

D. The rotate in automatic EOI mode flip-flop is cleared.

E. The IRR (Interrupt Request Register) is selected for the read register command.

F. If the IC4 bit equals 0 in ICW1, all functions in ICW4 are cleared; 8080/8085 mode is selected by default.

G. The fully nested mode is entered with an initial priority assignment of IR0 highest through IR7 lowest.

H. The edge sense latch of each IR priority cell is cleared, thus requiring a low to high transition to generate an interrupt (edge triggered mode effected only).

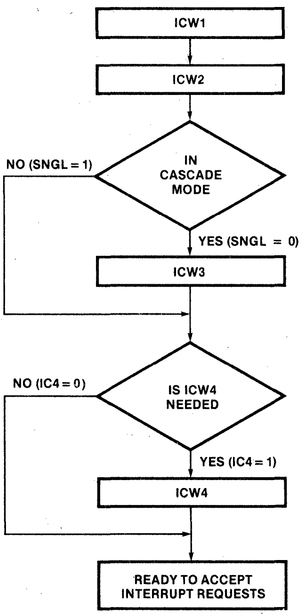
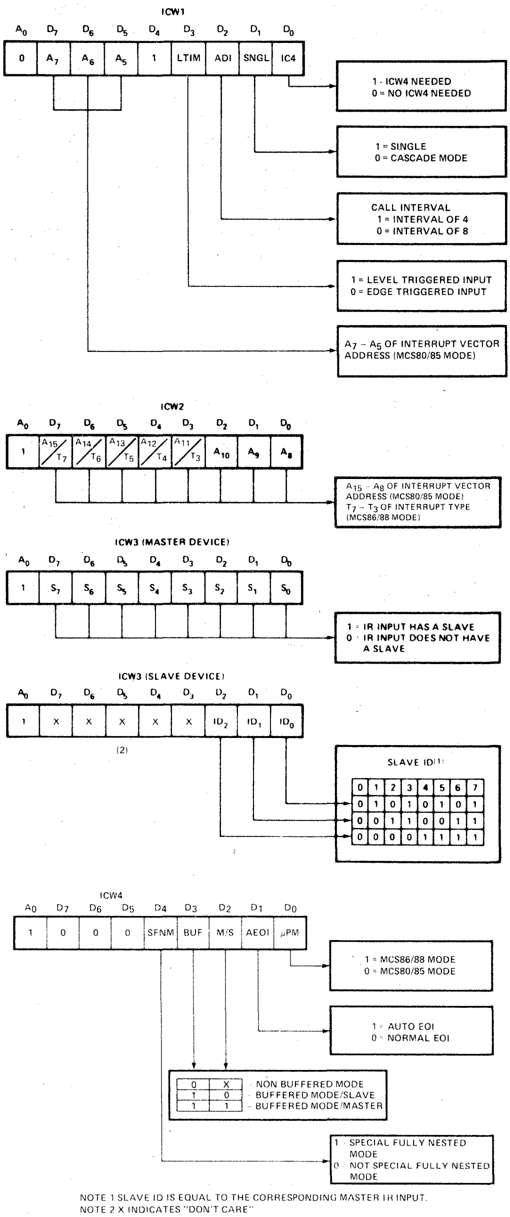


Figure 20. Initialization Flow

The ICW programming format, Figure 21, shows bit designation and a short definition of each ICW. With the ICW format as reference, the functions of each ICW will now be explained individually.



SOME OF THE TERMINOLOGY USED MAY DIFFER SLIGHTLY FROM EXISTING 8259A DATA SHEETS. THIS IS DONE TO BETTER CLARIFY AND EXPLAIN THE PROGRAMMING OF THE 8259A, THE OPERATIONAL RESULTS REMAIN THE SAME.

Figure 21. Initialization Command Words (ICWS) Programming Format

ICW1 and ICW2

Issuing ICW1 and ICW2 is the minimum amount of programming needed for any type of 8259A operation. The majority of bits within these two ICWs are used to designate the interrupt vector starting address. The remaining bits serve various purposes. Description of the ICW1 and ICW2 bits is as follows:

IC4: The IC4 bit is used to designate to the 8259A whether or not ICW4 will be issued. If any of the ICW4 operations are to be used, ICW4 must equal 1. If they aren't used, then ICW4 needn't be issued and IC4 can equal 0. Note that if IC4 = 0, the 8259A will assume operation in the MCS-80/85 mode.

SNGL: The SNGL bit is used to designate whether or not the 8259A is to be used alone or in the cascade mode. If the cascade mode is desired, SNGL must equal 0. In doing this, the 8259A will accept ICW3 for further cascade mode programming. If the 8259A is to be used as the single 8259A within a system, the SNGL bit must equal 1; ICW3 won't be accepted.

ADI: The ADI bit is used to specify the address interval for the MCS-80/85 mode. If a 4-byte address interval is to be used, ADI must equal 1. For an 8-byte address interval, ADI must equal 0. The state of ADI is ignored when the 8259A is in the MCS-86/88 mode.

LTIM: The LTIM bit is used to select between the two IR input triggering modes. If LTIM = 1, the level triggered mode is selected. If LTIM = 0, the edge triggered mode is selected.

A5-A15: The A5-A15 bits are used to select the interrupt vector address when in the MCS-80/85 mode. There are two programming formats that can be used to do this. Which one is implemented depends upon the selected address interval (ADI). If ADI is set for the 4-byte interval, then the 8259A will automatically insert A0-A4 (A0, A1 = 0 and A2, A3, A4 = IR0-7). Thus A5-A15 must be user selected by programming the A5-A15 bits with the desired address. If ADI is set for the 8-byte interval, then A0-A5 are automatically inserted (A0, A1, A2 = 0 and A3, A4, A5 = IR0-7). This leaves A6-A15 to be selected by programming the A6-A15 bits with the desired address. The state of bit 5 is ignored in the latter format.

T3-T7: The T3-T7 bits are used to select the interrupt type when the MCS-86/88 mode is used. The programming of T3-T7 selects the upper 5 bits. The lower 3 bits are automatically inserted, corresponding to the IR level causing the interrupt. The state of bits A5-A10 will be ignored when in the MCS-86/88 mode. Establishing the actual memory address of the interrupt is shown in Figure 22.

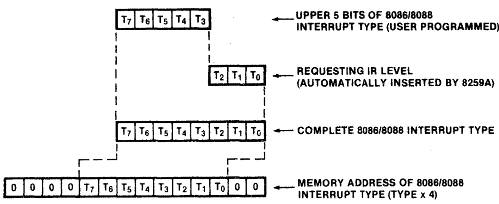


Figure 22. Establishing Memory Address of 8086/8088 Interrupt Type

ICW3

The 8259A will only accept ICW3 if programmed in the cascade mode (ICW1, SNGL = 0). ICW3 is used for specific programming within the cascade mode. Bit definition of ICW3 differs depending on whether the 8259A is a master or a slave. Definition of the ICW3 bits is as follows:

S0-7 (Master): If the 8259A is a master (either when the $\overline{SP}/\overline{EN}$ pin is tied high or in the buffered mode when M/S = 1 in ICW4), ICW3 bit definition is S0-7, corresponding to "slave 0-7". These bits are used to establish which IR inputs have slaves connected to them. A 1 designates a slave, a 0 no slave. For example, if a slave was connected to IR3, the S3 bit should be set to a 1. (S0) should be last choice for slave designation.

ID0-ID2 (Slave): If the 8259A is a slave (either when the $\overline{SP}/\overline{EN}$ pin is low or in the buffered mode when M/S = 0 in ICW4), ICW3 bit definition is used to establish its individual identity. The ID code of a particular slave must correspond to the number of the masters IR input it is connected to. For example, if a slave was connected to IR6 of the master, the slaves ID0-2 bits should be set to ID0 = 0, ID1 = 1, and ID2 = 1.

ICW4

The 8259A will only accept ICW4 if it was selected in ICW1 (bit IC4 = 1). Various modes are offered by using ICW4. Bit definition of ICW4 is as follows:

μPM: The μPM bit allows for selection of either the MCS-80/85 or MCS-86/88 mode. If set as a 1 the MCS-86/88 mode is selected, if a 0, the MCS-80/85 mode is selected.

AEOI: The AEOI bit is used to select the automatic end of interrupt mode. If AEOI = 1, the automatic end of interrupt mode is selected. If AEOI = 0, it isn't selected; thus an EOI command must be used during a service routine.

M/S: The M/S bit is used in conjunction with the buffered mode. If in the buffered mode, M/S defines whether the 8259A is a master or a slave. When M/S is set to a 1, the 8259A operates as the master; when M/S is 0, it operates as a slave. If not programmed in the buffered mode, the state of the M/S bit is ignored.

BUF: The BUF bit is used to designate operation in the buffered mode, thus controlling the use of the SP/EN pin. If BUF is set to a 1, the buffered mode is programmed and SP/EN is used as a transceiver enable output. If BUF is 0, the buffered mode isn't programmed and SP/EN is used for master/slave selection. Note if ICW4 isn't programmed, SP/EN is used for master/slave selection.

SFNM: The SFNM bit designates selection of the special fully nested mode which is used in conjunction with the cascade mode. Only the master should be programmed in the special fully nested mode to assure a truly fully nested structure among the slave IR inputs. If SFNM is set to a 1, the special fully nested mode is selected; if SFNM is 0, it is not selected.

4.2 OPERATIONAL COMMAND WORD (OCWs)

Once initialized by the ICWs, the 8259A will most likely be operating in the fully nested mode. At this point, operation can be further controlled or modified by the use of OCWs (Operation Command Words). Three OCWs are available for programming various modes and commands. Unlike the ICWs, the OCWs needn't be in any type of sequential order. Rather, they are issued by the processor as needed within a program.

Figure 23, the OCW programming format, shows the bit designation and short definition of each OCW. With the OCW format as reference, the functions of each OCW will be explained individually.

OCW1

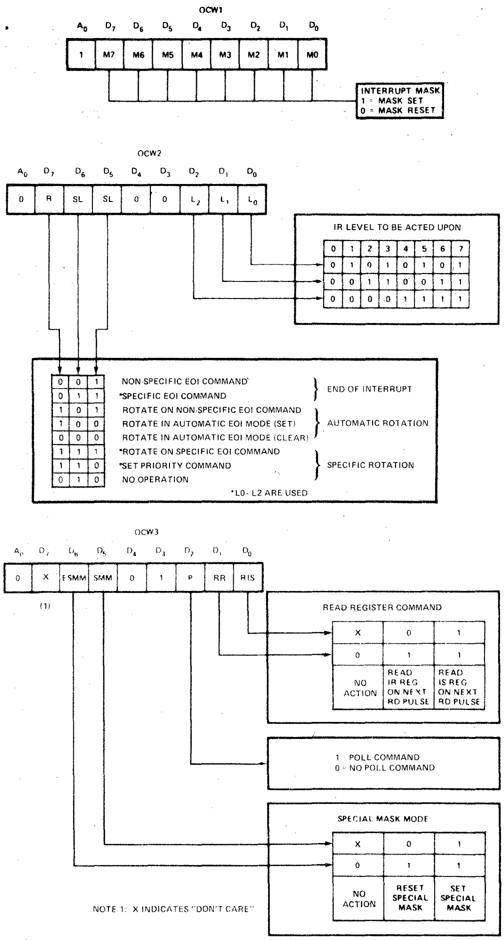
OCW1 is used solely for 8259A masking operations. It provides a direct link to the IMR (Interrupt Mask Register). The processor can write to or read from the IMR via OCW1. The OCW1 bit definition is as follows:

M0–M7: The M0–M7 bits are used to control the masking of IR inputs. If an M bit is set to a 1, it will mask the corresponding IR input. A 0 clears the mask, thus enabling the IR input. These bits convey the same meaning when being read by the processor for status update.

OCW2

OCW2 is used for end of interrupt, automatic rotation, and specific rotation operations. Associated commands and modes of these operations (with the exception of AEOI initialization), are selected using the bits of OCW2 in a combined fashion. Selection of a command or mode should be made with the corresponding table for OCW2 in the OCW programming format (Figure 20), rather than on a bit by bit basis. However, for completeness of explanation, bit definition of OCW2 is as follows:

L0–L2: The L0–L2 bits are used to designate an interrupt level (0–7) to be acted upon for the operation selected by the EOI, SL, and R bits of OCW2. The level designated will either be used to reset a specific ISR bit or to set a specific priority. The L0–L2 bits are enabled or disabled by the SL bit.



SOME OF THE TERMINOLOGY USED MAY DIFFER SLIGHTLY FROM EXISTING 8259A DATA SHEETS. THIS IS DONE TO BETTER CLARIFY AND EXPLAIN THE PROGRAMMING OF THE 8259A, THE OPERATIONAL RESULTS REMAIN THE SAME.

Figure 23. Operational Command Words (OCWs) Programming Format

EOI: The EOI bit is used for all end of interrupt commands (not automatic end of interrupt mode). If set to a 1, a form of an end of interrupt command will be executed depending on the state of the SL and R bits. If EOI is 0, an end of interrupt command won't be executed.

SL: The SL bit is used to select a specific level for a given operation. If SL is set to a 1, the L0–L2 bits are enabled. The operation selected by the EOI and R bits will be executed on the specified interrupt level. If SL is 0, the L0–L2 bits are disabled.

R: The R bit is used to control all 8259A rotation operations. If the R bit is set to a 1, a form of priority rotation will be executed depending on the state of SL and EOI bits. If R is 0, rotation won't be executed.

OCW3

OCW3 is used to issue various modes and commands to the 8259A. There are two main categories of operation associated with OCW3: interrupt status and interrupt masking. Bit definition of OCW3 is as follows:

RIS: The RIS bit is used to select the ISR or IRR for the read register command. If RIS is set to 1, ISR is selected. If RIS is 0, IRR is selected. The state of the RIS is only honored if the RR bit is a 1.

RR: The RR bit is used to execute the read register command. If RR is set to a 1, the read register command is issued and the state of RIS determines the register to be read. If RR is 0, the read register command isn't issued.

P: The P bit is used to issue the poll command. If P is set to a 1, the poll command is issued. If it is 0, the poll command isn't issued. The poll command will override a read register command if set simultaneously.

SMM: The SMM bit is used to set the special mask mode. If SMM is set to a 1, the special mask mode is selected. If it is 0, it is not selected. The state of the SMM bit is only honored if it is enabled by the ESMM bit.

ESMM: The ESMM bit is used to enable or disable the effect of the SMM bit. If ESMM is set to a 1, SMM is enabled. If ESMM is 0, SMM is disabled. This bit is useful to prevent interference of mode and command selections in OCW3.

5. APPLICATION EXAMPLES

In this section, the 8259A is shown in three different application examples. The first is an actual design implementation supporting an 8080A microprocessor system, "Power Fail/Auto Start with Battery Back-Up RAM". The second is a conceptual example of incorporating more than 64 interrupt levels in an 8080A or 8085A system, "78 Level Interrupt System". The third application is a conceptual design using an 8086 system, "Timer Controlled Interrupts". Although specific microprocessor systems are used in each example, these applications can be applied to either MCS-80, MCS-85, MCS-86, or MCS-88 systems, providing the necessary hardware and software changes are made. Overall, these applications should serve as a useful guide, illustrating the various procedures in using the 8259A.

5.1 POWER FAIL/AUTO-START WITH BATTERY BACK-UP RAM

The first application illustrates the 8259A used in an 8080A system, supporting a battery back-up scheme for the RAM (Random Access Memory) in a microcomputer system. Such a scheme is important in numerical and process control applications. The entire microcomputer system could be supported by a battery back-up scheme, however, due to the large amount of current usually required and the fact that most machinery is not supported by an auxiliary power source, only the state of calculations and variables usually need to be saved. In the event of a loss of power, if these items are not already stored in RAM, they can be transferred there and saved using a simple battery back-up system.

The vehicle used in this application is the Intel® SBC-80/20 Single Board Computer. An 8259A is used in the SBC-80/20 along with control lines helpful in implementing the power-down and automatic restart sequence used in a battery back-up system. The SBC-80/20 also contains user-selectable jumpers which allow the on-board RAM to be powered by a supply separate from the supply used for the non-RAM components. Also, the output of an undedicated latch is available to be connected to the IR inputs of the 8259A (the latch is cleared via an output port). In addition, an undedicated, buffered input line is provided, along with an input to the RAM decoder that will protect memory when asserted.

The additional circuitry to be described was constructed on an SBC-905 prototyping board. An SBC-635 power supply was used to power the non-RAM section of the SBC-80/20 while an external DC supply was used to simulate the back-up battery supplying power to the RAM. The SBC-635 was used since it provides an open collector ACLO output which indicates that the AC input line voltage is below 103/206 VAC (RMS).

The following is an example of a power-down and restart sequence that introduces the various power fail signals.

1. An AC power failure occurs and the ACLO goes high (ACLO is pulled up by the battery supply). This indicates that DC power will be reliable for at most 7.5 ms. The power fail circutry generates a Power Fail Interrupt ($\overline{\text{PFI}}$) signal. This signal sets the $\overline{\text{PFI}}$ latch, which is connected to the IR0 input of the 8259A, and sets the Power Fail Sense (PFS) latch. The state of this latch will indicate to the processor, upon reset, whether it is coming up from a power failure (warm start) or if it is coming up initially (cold start).
2. The processor is interrupted by the 8259A when the PFI latch is set. This pushes the pre-power-down program counter onto the stack and calls the service routine for the IR0 input. The IR0 service routine saves the processor status and any other needed variables. The routine should end with a HALT instruction to minimize bus transitions.
3. After a predetermined length of time (5 ms in this example) the power fail circuitry generates a Memory Protect ($\overline{\text{MPRO}}$) signal. All processing for the power failure (including the interrupt response delays) must be completed within this 5 ms window. The $\overline{\text{MPRO}}$ signal ensures that spurious transitions on the system control bus caused by power going down do not alter the contents of the RAM.
4. DC power goes down.
5. AC power returns. The power-on reset circuitry on the SBC-80/20 generates a system RESET.
6. The processor reads the state of the $\overline{\text{PFS}}$ line to determine the appropriate start-up sequence. The PFS latch is cleared, the MPRO signal is removed, and the PFI latch driving IR0 is cleared by the Power Fail Sense Reset ($\overline{\text{PFSR}}$) signal. The system then continues from the pre-power-down location for a warm start by restoring the processor status and popping the pre-power-down program counter off the stack.

Figure 24 illustrates this timing.

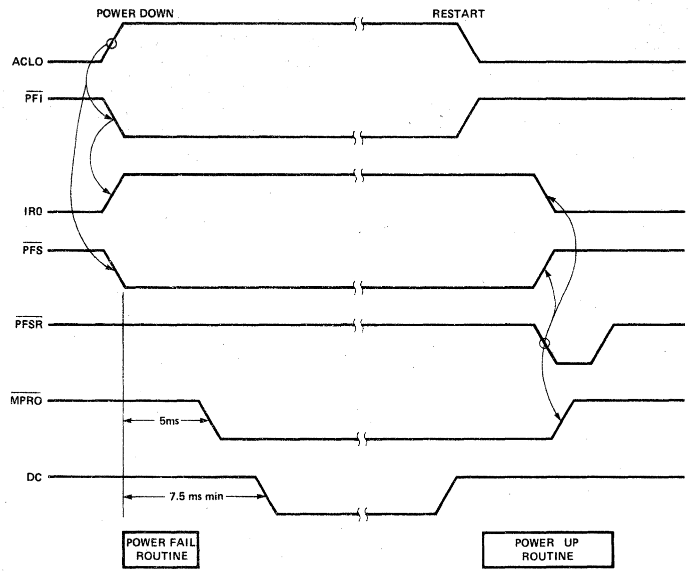


Figure 24. Power Down Restart Timing

Figure 25 shows the block diagram for the system. Notice that the RAM, the RAM decoder, and the power-down circuitry are powered by the battery supply.

The schematic of the power-down circuitry and the SBC-80/20 interface is shown in Figure 26. The design is very straightforward and uses CMOS logic to minimize the battery current requirements. The cold start switch is necessary to ensure that during a cold start, the $\overline{PFS}$ line is indicating "cold start" sense ($\overline{PFS}$ high). Thus, for a cold start, the cold start switch is depressed during power on. After that, no further action is needed. Notice that the $\overline{PFI}$ signal sets the on-board PFI latch. The output of this latch drives the 8259A IR0 input. This latch is cleared during the restart routine by executing an OUTput D4H instruction. The state of the $\overline{PFS}$ line may be read on the least significant data bus line (DB0) by executing an INput D4H instruction. An 8255 port (8255 #1, port C, bit 0) is used to control the $\overline{PFSR}$ line.

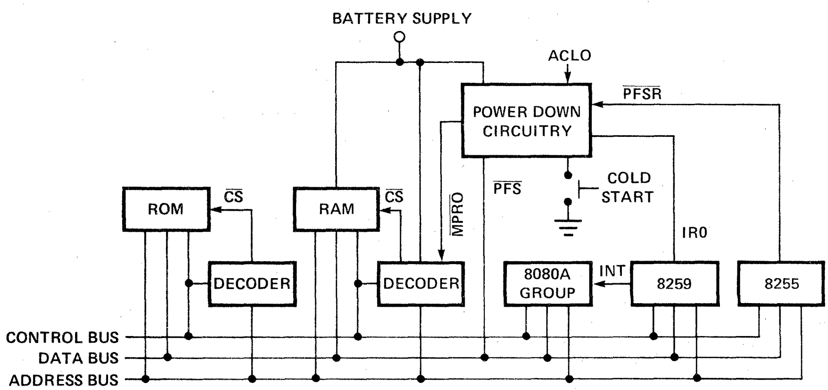


Figure 25. Block Diagram of SBC 80/20 with Power Down Circuit

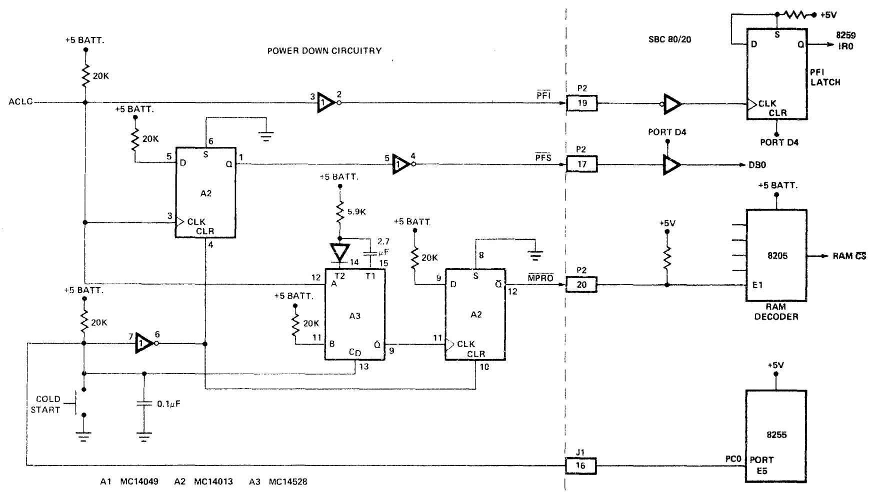


Figure 26. Power Down Circuit - SBC 80/20 Interface

The fully nested mode for the 8259A is used in its initial state to ensure the IR0 always has the highest priority. The remaining IR inputs can be used for any other purpose in the system. The only constraint is that the service routines must enable interrupts as early as possible. Obviously, this is to ensure that the power-down interrupt does not have to wait for service. If a rotating priority scheme is desired, another 8259A could be added as a slave and be programmed to operate in a rotating mode. The master would remain in the initial state of the fully nested mode so that the IR0 still remains the highest priority input.

The software to support the power-down circuitry is shown in Figure 27. The flow for each label will be discussed.

After any system reset, the processor starts execution at location 0000H (START). The PFS status is read and execution is transferred to CSTART if PFS indicates a cold start (i.e., someone is depressing the cold start switch) or WSTART if a warm start is indicated (PFS LOW). CSTART is the start of the user's program. The Stack Pointers (SP) and device initialization were included just to remind the reader that these must occur. The first EI instruction must appear after the 8259A has received its initialization sequence. The 8259A (and other devices) are initialized in the INIT subroutine.

When a power failure occurs, execution is vectored by the 8259A to REGSAV by way of the jump table at JSTART. The pre-power-down program counter is placed on the stack. REGSAV saves the processor registers and flags in the usual manner by pushing them onto the stack. Other items, such as output port status, programmable peripheral states, etc., are pushed onto the stack at this time. The Stack Pointer (SP) could be pushed onto the stack by way of the register pair HL but the top of the stack can exist anywhere in memory and there is no way then of knowing where that is when in the power-up routine. Thus, the SP is saved at a dedicated location in RAM. It isn't really necessary to send an EOI command to the 8259A in REGSAV since power will be removed from the 8259A, but one is included for completeness. The final instruction before actually losing power is a HALT. This minimizes somewhat spurious transitions on the various busses and lets the processor die gracefully.

On reset, when a warm start is detected, execution is transferred to WSTART. WSTART activates PFSR by way of the 8255 (all outputs go low then the 8255 is initialized). In the power-down circuitry, PFSR clears the PFS latch and removes the MPRO signal which then allows access to the RAM. WSTART also clears the PFI latch which arms the 8259A IR0 input. Then the 8259A is re-initialized along with any other devices. The SP is retrieved from RAM and the processor registers and flags are restored by popping them off the stack. Interrupts are then enabled. Now the power-down program counter is on top of the stack, so executing a RETurn instruction transfers the processor to exactly where it left off before the power failure.

Aside from illustrating the usefulness of the 8259A (and the SBC-80/20) in implementing a power failure protected microcomputer system, this application should also point out a way of preserving the processor status when using interrupts.

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                  0 ;
                  1 ;
                  2 ;POWER DOWN AND RESTART FOR THE SBC 80/20
                  3 ;
                  4 ;
                  5 ;SYSTEM EQUATES:
00DA              6 PT59A   EQU    0DAH           ;8259 PORT WITH A0=0
00DB              7 PT59B   EQU    0DBH           ;8259 PORT WITH A0=1
00E7              8 PPI1CT  EQU    0E7H           ;8255 #1 CONTROL PORT
00E6              9 PPI1C   EQU    0E6H           ;8255 #1 PORT C
3800             10 SPSAVE  EQU    3800H          ;SP STORAGE IN RAM
0001             11 JPT     EQU    01H            ;MSB OF 8259 JUMP TABLE
                 12 ;
                 13 ;
                 14 ;STARTIING POINT AFTER SYSTEM RESET
                 15 ;
                 16 ;
0000             17         ORG    0H
0000 DBD4        18 START:  IN     0D4H           ;READ PFS/ STATUS
0002 1F          19         RAR                   ;PFS/ ON DB0, PUT IN CARRY
0003 DA2001      20         JC     CSTART         ;PFS/=1, THEN COLD START
                 21 ;
                 22 ;
                 23 ;WSTART LOCATION, PFS/=0, THEN WARM START
                 24 ;
                 25 ;
0006 3E80        26 WSTART: MVI    A,80H          ;SET 8255 #1 TO OUTPUT MODE
0008 D3E7        27         OUT    PPI1CT         ;8255 CONTROL PORT, PFSR/ GOES LOW
                 28         ;
                 29         ;OUTPUT COMMAND MAKES PFSR/ GO LOW WHICH REMOVES MPRO/ AND
                 30         ;CLEARS PFS LATCH
                 31         ;
000A 3E01        32         MVI    A,01H          ;RETURN PFSR/ HIGH
000C D3E6        33         OUT    PPI1C          ;8255 #1 PORT C
000E D3D4        34         OUT    0D4H           ;RESET PFI LATCH
0010 CD1D00      35         CALL   INIT           ;GO INITIALIZE EVERYTHING
0013 2A0038      36         LHLD   SPSAVE         ;RETRIVE SP FROM RAM
0016 F9          37         SPHL                  ;PUT BACK INTO SP
0017 C1          38         POP    B              ;RESTORE BC
0018 D1          39         POP    D              ;RESTORE DE
0019 E1          40         POP    H              ;RESTORE HL
001A F1          41         POP    PSW            ;RESTORE A PLUS FLAGS
001B FB          42         EI                    ;ENABLE INTERRUPTS
001C C9          43         RET                   ;PRE-POWER-DOWN PC ON TOP OF STACK
                 44                               ;RETURN TO IT
                 45 ;
                 46 ;
                 47 ;INITIALIZATION ROUTINE.  AT LEAST DO 8259 BUT OTHERS CAN BE ADDED
                 48 ;
                 49 ;
001D 3E16        50 INIT:   MVI    A,16H          ;F=1,S=1,A7-A5=0  ICW1
001F D3DA        51         OUT    PT59A          ;8259 PORT WITH A0=0
0021 3E01        52         MVI    A,JPT          ;MSB OF JUMP TABLE ICW2
0023 D3DB        53         OUT    PT59B          ;8259 PORT WITH A0=1
                 54
                 55         ;ADD ANY OTHER INITIALIZATIONS HERE
                 56         ;
0025 C9          57         RET                   ;RETURN
                 58 ;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                 59 ;
                 60 ;POWER DOWN ROUTINE TO SAVE REGISTERS AND STATUS
                 61 ;
                 62 ;
0026 F5          63 REGSAV: PUSH   PSW            ;SAVE A PLUS FLAGS
0027 E5          64         PUSH   H              ;SAVE HL
0028 D5          65         PUSH   D              ;SAVE DE
0029 C5          66         PUSH   B              ;SAVE BC
002A 210000      67         LXI    H,0000H        ;GET SET TO GET SP
002D 39          68         DAD    SP             ;SP NOW IN HL
002E 220038      69         SHLD   SPSAVE         ;SAVE SP IN RAM
                 70         ;
                 71         ;EOI NOT REALLY NEEDED BUT INCLUDED FOR COMPLETENESS
                 72         ;
0031 3E20        73         MVI    A,20H          ;NON-SPECIFIC EOI
0033 D3DA        74         OUT    PT59A          ;8259 PORT WITH A0=0
0035 76          75         HLT                   ;HALT - GO DOWN GRACEFULLY
                 76 ;
                 77 ;
                 78 ;8259 JUMP TABLE.  ONLY IR0 IS USED, OTHERS DIRECTED TO RAM
                 79 ;
                 80
0100             81         ORG    100H
0100 C32600      82 JSTART  JMP    REGSAV         ;IR0
0103 00          83         NOP
0104 C31038      84         JMP    3810H          ;IR1
0107 00          85         NOP
0108 C32038      86         JMP    3820H          ;IR2
010B 00          87         NOP
010C C33038      88         JMP    3830H          ;IR3
010F 00          89         NOP
0110 C34038      90         JMP    3840H          ;IR4
0113 00          91         NOP
0114 C35038      92         JMP    3850H          ;IR5
0117 00          93         NOP
0118 C36038      94         JMP    3860H          ;IR6
011B 00          95         NOP
011C C37038      96         JMP    3870H          ;IR7
011F 00          97         NOP
                 98 ;
                 99 ;
                100 ;COLD START LOCATION, USER'S PROGRAM ENTERS HERE
                101 ;
                102 ;
0120 31803F     103 CSTART: LXI    SP,3F80H       ;INITIALIZE SP
0123 CD1D00     104         CALL   INIT           ;INITIALIZE EVERYTHING ELSE
0126 D3D4       105         OUT    0D4H           ;RESET PFI LATCH
0128 FB         106         EI                    ;ENABLE INTERRUPTS
                107         ;
                108         ;USER PROGRAM STARTS HERE
                109         ;
                110         END                   ;DONE
```

Figure 27. Power Down and Restart Software

5.2 78 LEVEL INTERRUPT SYSTEM

The second application illustrates an interrupt structure with greater than 64 levels for an 8080A or 8085A system. In the cascade mode, the 8259A supports up to 64 levels with direct vectoring to the service routine. Extending the structure to greater than 64 levels requires polling, using the poll command. A 78 level interrupt structure is used as an illustration; however, the principles apply to systems with up to 512 levels.

To implement the 78 level structure, 3 tiers of 8259A's are used. Nine 8259A's are cascaded in the master-slave scheme, giving 64 levels at tier 2. Two additional 8259A's are connected, by way of the INT outputs, to two of the 64 inputs. The 16 inputs at tier 3, combined with the 62 remaining tier 2 inputs, give 78 total levels. The fully nested structure is preserved over all levels, although direct vectoring is supplied for only the tier 2 inputs. Software is required to vector any tier 3 requests. Figure 28 shows the tiered structure used in this example. Notice that the tier 3 8259A's are connected to the bottom level slave (SA7). The master-slaves are interconnected as shown in "Interrupt Cascading", while the tier 3 8259A's are connected as "masters"; that is, the SP/EN pins are pulled high and the CAS pins are left unconnected. Since these 8259A's are only going to be used with the poll command, no INTA is required, therefore the INTA pins are pulled high.

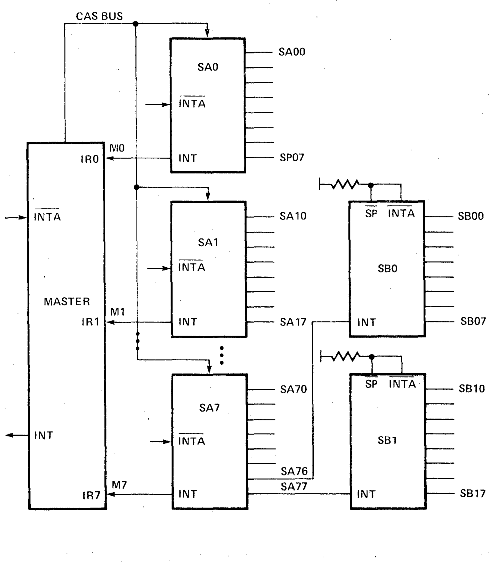


Figure 28. 78 Level Interrupt Structure

The concept used to implement the 78 levels is to directly vector to all tier 2 input service routines. If a tier 2 input contains a tier 3 8259A, the service routine for that input will poll the tier 3 8259A and branch to the tier 3 input service routine based on the poll word read after the poll command. Figure 29 shows how the jump table is organized assuming a starting location of 1000H and contiguous tables for all the tier 2 8259A's. Note that "SA35" denotes the IR5 input of the slave connected to the master IR3 input. Also note that for the normal tier 2 inputs, the jump table vectors the processor directly to the service routine for that input, while for the tier 2 inputs with 8259A's connected to their IR inputs, the processor is vectored to a service routine (i.e., SB0) which will poll to determine the actual tier 3 input requesting service. The polling routine utilizes the jump table starting at 1200H to vector the processor to the correct tier 3 service routine.

Each 8259A must receive an initialization sequence regardless of the mode. Since the tier 1 and 2 8259A's are in cascade and the special fully nested mode is used (covered shortly), all ICWs are required. The tier 3 8259A's don't require ICW3 or ICW4 since only polling will be used on them and they are connected as masters not in the cascade mode. The initialization sequence for each tier is shown in Figure 30. Notice that the master is initialized with a "dummy" jump table starting at 00H since all vectoring is done by the slaves. The tier 3 devices also receive "dummy" tables since only polling is used on tier 3.

As explained in "Interrupt Cascading", to preserve a truly fully nested mode within a slave, the master 8259A should be programmed in the special fully nested mode. This allows the master to acknowledge all interrupts at and above the level in service disregarding only those of lower priority. The special fully nested mode is programmed in the master only, so it only affects the immediate slaves (tier 2 not tier 3). To implement a fully nested structure among tier 3 slaves some special housekeeping software is required in all the tier-2-with-tier-3-slave routines. The software should simply save the state of the tier 2 IMR, mask all the lower tier 2 interrupts, then issue a specific EOI, resetting the ISR of the tier 2 interrupt level. On completion of the routine the IMR is restored.

Figure 31 shows an example flow and program for any tier 2 service routine without a tier 3 8259A. Figure 32 shows an example flow and program for any tier 2 service routine with a tier 3 8259A. Notice the reading of the ISR in both examples; this is done to determine whether or not to issue an EOI command to the master (refer to the section on "Special Fully Nested Mode" for further details).

```
LOCATION    8259   CODE               COMMENTS
1000 H      SA0    JMP   SA00         ; SA00 SERVICE ROUTINE
   .
   .
101C H             JMP   SA07         ; SA07 SERVICE ROUTINE
1020 H      SA1    JMP   SA10         ; SA10 SERVICE ROUTINE
   .
103C H             JMP   SA17         ; SA17 SERVICE ROUTINE
   .         .            .
   .         .            .           ; SA20-SA67 SERVICE ROUTINES
   .         .            .
10E0 H      SA7    JMP   SA70         ; SA70 SERVICE ROUTINE
   .
10F8 H             JMP   SB0          ; SB0 POLL ROUTINE
10FC H             JMP   SB1          ; SB1 POLL ROUTINE
1200 H      SB0    JMP   SB00         ; SB00 SERVICE ROUTINE
   .
121C H             JMP   SB07         ; SB07 SERVICE ROUTINE
1220 H      SB1    JMP   SB10         ; SB10 SERVICE ROUTINE
   .
123C H             JMP   SB17         ; SB17 SERVICE ROUTINE
```

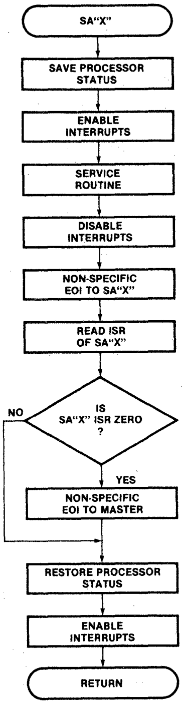
Figure 29. Jump Table Organization

```
; INITIALIZATION SEQUENCE FOR 78 LEVEL INTERRUPT STRUCTURE
;
; INITIALIZE MASTER
;
MINT:    MVI    A,15H      ; ICWI, LTM = 0, ADI = 1, S = 0, IC4 = 1
         OUT    MPTA       ; MASTER PORT A0 = 0
         MVI    A,00H      ; ICW2, DUMMY ADDRESS
         OUT    MPTB       ; MASTER PORT A0 = 1
         MVI    A,0FFH     ; ICW3, S7-S0 = 1
         OUT    MPTB       ; MASTER PORT A0 = 1
         MVI    A,I0H      ; ICW4, SFNM = 1
         OUT    MPTB       ; MASTER PORT A0 = 1
;
; INITIALIZE SA SLAVES - X DENOTES SLAVE ID (SEE KEY)
;
SAXINT:  MVI    A,α        ; SEE KEY FOR ICW1. LTM = 0, ADI = 1, S = 0, IC4 = 1
         OUT    SAXPTA     ; SA"X" PORT A0 = 0
         MVI    A, 10H     ; ICW2, ADDRESS MSB
         OUT    SAXPTB     ; SA"X" PORT A0 = 1
         MVI    A0XH       ; ICW3, SA ID
         OUT    SAXPTB     ; SA"X" PORT A0 = 1
         MVI    A10H       ; ICW4, SFNM = 1
         OUT    SAXPTB     ; SA"X" PORT A0 = 1
;
; REPEAT ABOVE FOR EACH SA SLAVE
;
; INITIALIZE SB SLAVES - X DENOTES 0 or 1 (DO SB0, REPEAT FOR SB1)
;
SBXINT   MVI    A,I6H      ; ICW1, LTM = 0, ADI = 1, S = 1. IC4 = 0
         OUT    SBXPTA     ; SB"X" PORT A0 = 0
         MVI    A,00H      ; ICW2, DUMMY ADDRESS
         OUT    SBXPTB     ; SB"X" PORT A0 = 1
```

SA INITIALIZATION KEY

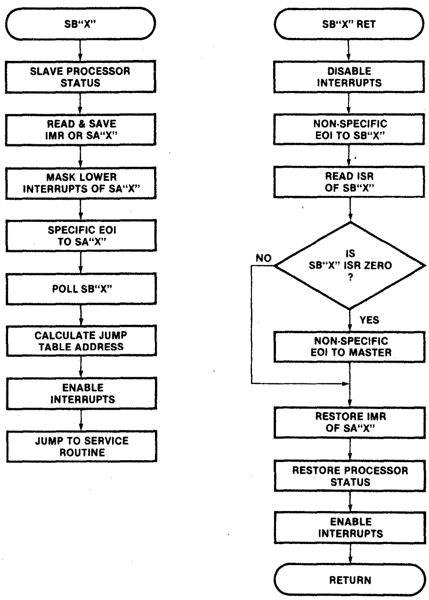
SA"X"	α (ICW1)	JUMP TABLE START (H)
0	15	1000
1	35	1020
2	55	1040
3	75	1060
4	95	1080
5	B5	10A0
5	D5	10C0
7	F5	10E0

Figure 30. Initialization Sequence for 78 Level Interrupt Structure




```
; SA"X" ROUTINE - GENERAL INTERRUPT SERVICE ROUTINE
; FOR TIER 2 INTERRUPTS WITHOUT TIER 3 8259A
;
SAX:     PUSH D              ; SAVE DE
         PUSH B              ; SAVE BC
         PUSH H              ; SAVE HL
         PUSH PSW            ; SAVE A, FLAGS
         EI                  ; ENABLE INTERRUPTS
;
; SERVICE ROUTINE GOES HERE
;
         DI                  ; DISABLE INTERRUPTS
         MVI    20H          ; OCW2, NON-SPECIFIC EOI
         OUT    SAXPTA       ; SA"X" PORT A0=0
         MUI    A,0BH        ; OCW3, READ REGISTER, ISR
         OUT    SAXPTA       ; SA"X" PORT A0=0
         IN     SAXPTA       ; SA"X" PORT A0=0, SA"X" ISR
         ANI    0FFH         ; TEST FOR ZERO
         JZN    SAXRSR       ; IF NOT ZERO, RESTORE STATUS
         MVI    A,0BH        ; OCW2, NON-SPECIFIC EOI
         OUT    MASPTA       ; MASTER PORT A0=0
SAXRSR:  POP    PSW          ; RESTORE A, FLAGS
         POP    H            ; RESTORE HL
         POP    B            ; RESTORE BC
         POP    D            ; RESTORE DE
         EI                  ; ENABLE INTERRUPTS
         RET                 ; RETURN
```

Figure 31. Example Service Routine for Tier 2 Interrupt (SA"X") without Tier 3 8259A (SB"X")




```
; SB"X" ROUTINE - SERVICE ROUTINE FOR TIER 2
; INTERRUPTS WITH TIER 3 8259AS
SBX:     PUSH D              ; SAVE DE
         PUSH B              ; SAVE BC
         PUSH H              ; SAVE HL
         PUSH PSW            ; SAVE A, FLAGS
         IN     SAXPTB       ; READ SA"X" IMR
         MOV    D,A          ; SAVE
         MVI    A,XXH        ; MASK SA"X" LOWER IR
         OUT    SAXPTB       ; SA"X" PORT A0=1
         MVI    A,6XH        ; OCW2 SPECIFIC EOI SA"X"
         OUT    SAXPTA       ; SA"X" PORT A0=1
         LXI    H,1200H      ; JUMP TABLE START
         MVI    B,00H        ; CLEAR B
         MVI    A,0CH        ; OCW3, POLL COMMAND
         OUT    SBXPTA       ; SB"X" PORT A0=0
         IN     SBXPTA       ; GET POLL WORD
         ANI    07H          ; LIMIT TO 3 BITS
         ADD    A            ; GET TABLE OFFSET
         ADD    A            ;
         MOV    C,A          ; OFFSET TO C
         DAD    B            ; HL HAS TABLE ADDRESS
         EI                  ; ENABLE INTERRUPTS
;
; SB"X"RET ROUTINE - FOR EOI AND MASK RESTORE
; AFTER SB"X" ROUTINE
;
SBXRET   DI                  ; DISABLE INTERRUPTS
         MVI    A,20H        ; OCW2, NON SPECIFIC EOI
         OUT    SBXPTA       ; SA"X" PORT A0=0
         MVI    A,0BH        ; OCW3, READ REGISTER ISR
         OUT    SAXPTA       ; SA"X" PORT A0=0
         IN     SBXPTA       ; SA"X" PORT A0=0, ISR
         ANI    0FFH         ; TEST FOR ZERO
         JNZ    SBXRSR       ; IF ≠0 RESTORE IMR
         MVI    A,20H        ; OCW2, NON-SPECIFIC EOI
         OUT    MASPTA       ; MASTER PORT A0=0
SBXRSR:  MOV    A,D          ; RESTORE SA"X" IMR
         OUT    SAXPTB       ; SA"X" PORT A0=1
         POP    PSW          ; RESTORE A, FLAGS
         POP    H            ; RESTORE HL
         POP    B            ; RESTORE BC
         POP    D            ; RESTORE BC
         EI                  ; RESTORE DE
         RET                 ; RETURN
```

Figure 32. Example Service Routine for Tier 2 Interrupt (SA"X") with Tier 3 8259A (SB"X")

5.3 TIMER CONTROLLED INTERRUPTS

In a large number of controller type microprocessor designs, certain timing requirements must be implemented throughout program execution. Such time dependent applications include control of keyboards, displays, CRTs, printers, and various facets of industrial control. These examples, however, are just a few of many designs which require device servicing at specific rates or generation of time delays. Trying to maintain these timing requirements by processor control alone can be costly in throughput and software complexity. So, what can be done to alleviate this problem? The answer, use the 8259A Programmable Interrupt Controller and external timing to interrupt the processor for time dependent device servicing.

This application example uses the 8259A for timer controlled interrupts in an 8086 system. External timing is done by two 8253 Programmable Interval Timers. Figure 33 shows a block diagram of the timer controlled interrupt circuitry which was built on the breadboard area of an SDK-86 (system design kit). Besides the 8259A and the 8253's, the necessary I/O decoding is also shown. The timer controlled interrupt circuitry interfaces with the SDK-86 which serves as the vehicle of operation for this design.

A short overview of how this application operates is as follows. The 8253's are programmed to generate interrupt requests at specific rates to a number of the 8259A IR inputs. The 8259A processes these requests by interrupting the 8086 and vectoring program execution to the appropriate service routine. In this example, the routines use the SDK-86 display panel to display the number of the interrupt level being serviced. These routines are merely for demonstration purposes to show the necessary procedures to establish the user's own routines in a timer controlled interrupt scheme.

Let's go over the operation starting with the actual interrupt timing generation which is done by two 8253 Programmable Interval Timers (8253 #1 and 8253 #2). Each 8253 provides three individual 16-bit counters (counters 0-2) which are software programmable by the processor. Each counter has a clock input (CLK), gate input (GATE), and an output (OUT). The output signal is based on divisions of the clock input signal. Just how or when the output occurs is determined by one of the 8253's six programmable modes, a programmable 16-bit count, and the state of the gate input.

Figure 34 shows the 8253 timing configuration used for generating interrupts to the 8259A. The SDK-86's PCLK (peripheral clock) signal provides a 400 ns period clock to CLK0 of 8253 #1. Counter 0 is used in mode 3 (square wave rate generator), and acts as a prescaler to provide the clock inputs of the other counters with a 10 ms period square wave. This 10 ms clock period made it easy to calculate exact timings for the other counters. Counter 2 of the 8253 #1 is used in mode 2 (rate generator), it is programmed to output a 10 ms pulse for every 200 pulses it receives (every 2 sec). The output of counter 2 causes an interrupt on IR1 of the 8259A. All the 8253 #2 counters are used in mode 5 (hardware triggered strobe) in which the gate input initiates counter operations. In this case the output of 8253 #1 counter 2 controls the gate of each 8253 #2 counter. When one of the 8253 #2 counters receive the 8253 #1 counter 2 output pulse on its gate, it will output a pulse (10 ms in duration) after a certain preprogrammed number of clock pulses have occurred. The programmed number of clock pulses for the 8253 #2 counters is as follows: 50 pulses (0.5 sec) for counter 0, 100 pulses (1 sec) for counter 1, and 150 pulses (1.5 sec) for counter 2. The outputs of these counters cause interrupt requests on IR2 through IR4 of the 8259A. Counter 1 of 8253 #1 is used in mode 0 (interrupt on terminal count). Unlike the other modes used which initialize operation automatically or by gate triggering, mode 0 allows software controlled counter initialization. When counter 1 of 8253 #1 is set during program execution, it will count 25 clocks (250 ms) and then pull its output high, causing an interrupt request on IR0 of the 8259A. Figure 35 shows the timing generated by the 8253's which cause interrupt request on the 8259A IR inputs.

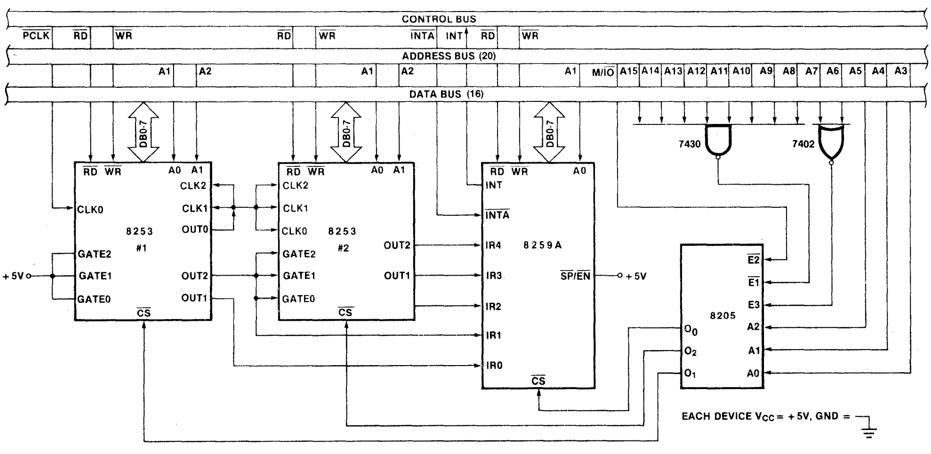


Figure 33. Timer Controlled Interrupt Circuit on SDK 86 Breadboard Area

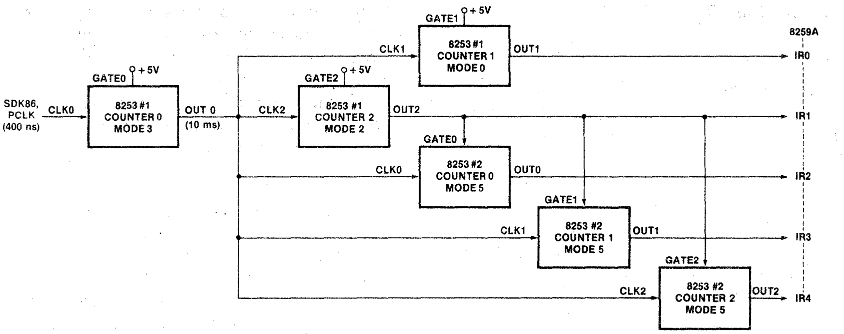


Figure 34. 8253 Timing Configuration for Timer Controlled Interrupts

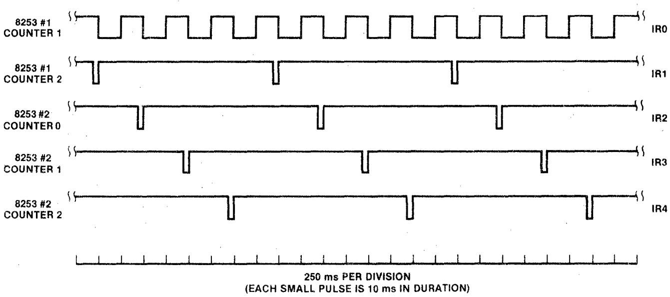


Figure 35. 8259A IR Input Signal From 8253S

There are basically two methods of timing generation that can be used in a timer controlled interrupt structure: dependent timing and independent timing. Dependent timing uses a single timing occurrence as a reference to base other timing occurrences on. On the other hand, independent timing has no mutual reference between occurrences. Industrial controller type applications are more apt to use dependent timing, whereas independent timing is prone to individual device control.

Although this application uses primarily dependent timing, independent timing is also incorporated as an example. The use of dependent timing can be seen back in Figure 34, where timing for IR2 through IR4 uses the IR1 pulse as reference. Each one of the 8253 #2 counters will generate an interrupt request a specific amount of times after the IR1 interrupt request occurs. When using the dependent method, as in this case, the IR2 through IR4 requests must occur before the next IR1 request. Independent timing is used to control the IR0 interrupt request. Note that its timing isn't controlled by any of the other IR requests. In this timer controlled interrupt configuration the dependent timing is initially set to be self running and the independent timing is software initialized. However, both methods can work either way by using the various 8253 modes to generate the same interrupt timing.

The 8259A processes the interrupts generated by the 8253's according to how it is programmed. In this application it is programmed to operate in the edge triggered mode, MCS-86/88 mode, and automatic EOI mode. In the edge triggered mode an interrupt request on an 8259A IR input becomes active on the rising edge. With this in mind, Figure 35 shows that IR0 will generate an interrupt every half second and IR1 through IR4 will each generate an interrupt every 2 seconds spaced apart at half second intervals. Interrupt vectoring in the MCS-86/88 mode is programmed so IR0, when activated, will select interrupt type 72. This means IR1 will select interrupt type 73, IR2 interrupt type 74, and so on through IR4. Since IR5 through IR7 aren't used, they are masked off. This prevents the possibility of any accidental interrupts and rids the necessity to tie the unused IR inputs to a steady level. Figure 36 shows the 8259A IR levels (IR0–IR4) with their corresponding interrupt type in the 8086 interrupt-vector table. Type 77 in the table is selected by a software "INT" instruction during program execution. Each type is programmed with the necessary code segment and instruction pointer values for vectoring to the appropriate service routine. Since the 8259A is programmed in the automatic EOI Mode, it doesn't require an EOI command to designate the completion of the service routine.

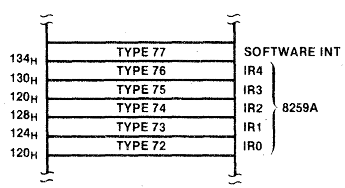


Figure 36. Interrupt "Type" Designation

As mentioned earlier, the interrupt service routines in this application are used merely to demonstrate the timer controlled interrupt scheme, not to implement a particular design. Thus a service routine simply displays the number of its interrupting level on the SDK-86 display panel. The display panel is controlled by the 8279 Keyboard and Display Controller. It is initialized to display "Ir" in its two left-most digits during the entire display sequence. When an interrupt from IR1 through IR4 occurs the corresponding routine will display its IR number via the 8279. During each IR1 through IR4 service routine a software "INT77" instruction is executed. This instruction vectors program execution to the service routine designated by type 77, which sets the 8253 counter controlling IR0 so it will cause an interrupt in 250 ms. When the IR0 interrupt occurs its routine will turn off the digit displayed by the IR1 through IR4 routines. Thus each IR level (IR1-IR4) will be displayed for 250 ms followed by a 250 ms off time caused by IR0. Figure 37 shows the entire display sequence of the timer controlled interrupt application.

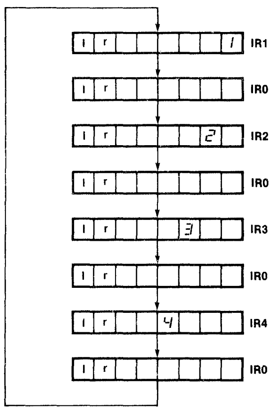


Figure 37. SDK Display Sequence for Timer Controlled Interrupts Program (Each Display Block Shown is 250 msec in Duration)

Now that we've covered the operation, let's move on to the program flow and structure of the timer controlled interrupt program. The program flow is made up of an initialization section and six interrupt service routines. The initialization program flow is shown in Figure 38. It starts by initializing some of the 8086's registers for program operation; this includes the extra segment, data segment, stack segment, and stack pointer. Next, by using the extra segement as reference, interrupt types 72 through 77 are set to vector interrupts to the appropriate routines. This is done by moving the code segment and instruction pointer values of each service routine into the corresponding type location. The 8253 counters are then programmed with the proper mode and count to provide the interrupt timing mentioned earlier. All counters with the exception of the 8253 #1, counter 1 are fully initialized at this point and will start counting. Counter 1 of 8253 #1 starts counting when its counter is loaded during the "INTR77" service routine, which will be covered shortly. Next, the 8259A is issued ICW1, ICW2, ICW4, and OCW1. The ICWs program the 8259A for the edge triggered mode, automatic EOI mode, and the proper interrupt vectoring (IR0, type 72). OCW1 is used to mask off the unused IR inputs (IR5-IR7). The 8279 is then set to display "IR" on its two left-most digits. After that the 8086 enables interrupts and a "dummy" main program is executed to wait for interrupt requests.

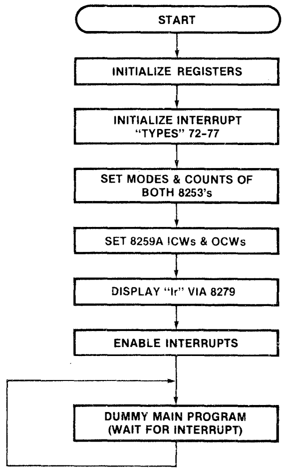


Figure 38. Initialization Program Flow for Timer Controlled Interrupts

There are six different interrupt service routines used in the program. Five of these routines, "INTR72" through "INTR76", are vectored to via the 8259A. Figure 39A-C shows the program flow for all six service routines. Note that "INTR73" through "INTR76" (IR1-IR4) basically use the same flow. These four similar routines display the number of its interrupting IR level on the SDK-86 display panel. The "INTR77" routine is vectored to by software during each of the previously mentioned routines and sets up interrupt timing to cause the "INTR72" (IR0) routine to be executed. The "INTR72" routine turns off the number on the SDK-86 display panel.

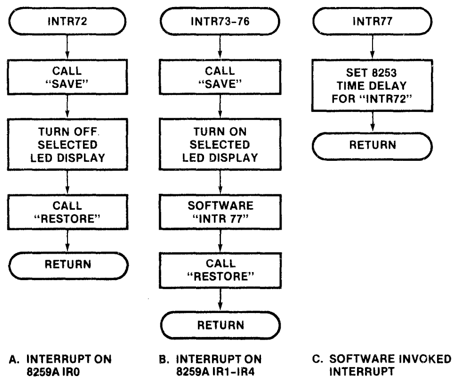


Figure 39. A-C. Interrupts Service Routine Flow for Timer Controlled Interrupts.

To best explain how these service routines work, let's assume an interrupt occurred on IR1 of the 8259A. The associated service routine for IR1 is "INTR73". Entering "INTR73", the first thing done is saving the pre-interrupt program status. This isn't really necessary in this program since a "dummy" main program is being executed; however, it is done as an example to show the operation. Rather than having code for saving the registers in each separate routine, a mutual call routine, "SAVE", is used. This routine will save the register status by pushing it on the stack. The next portion of "INTR73" will display the number of its IR level, "1", in the first digit of the SDK-86 display panel. After that, a software INT instruction is executed to vector program execution to the "INTR77" service routine. The "INTR77" service routine simply sets the 8253 #1 counter 1 to cause an interrupt on IR0 in 250 ms and then returns to "INTR73". Once back in "INTR73", the pre-interrupt status is restored by a call routine, "RESTORE". It does the opposite of "SAVE", returning the register status by popping it off the stack. The "INTR73" routine then returns to the "dummy " main program. The flow for the "INTR74" through "INTR76" routines are the same except for the digit location and the IR level displayed.

After 250 ms have elapsed, counter 1 of 8253 #1 makes an interrupt request on IR0 of the 8259A. This causes the "INTR72" service routine to be executed. Since this routine interrupts the main program, it also uses the "SAVE" routine to save pre-interrupt program status. It then turns off the digit displaying the IR level. In the case of the "INTR73" routine, the "1" is blanked out. The pre-interrupt status is then restored using the "RESTORE" routine and program execution returns to the "dummy" main program.

The complete program for the timer controlled interrupts application is shown in Appendix B. The program was executed in SDK-86 RAM starting at location 0500H (code segment = 0050, instruction pointer = 0).

CONCLUSION

This application note has explained the 8259A in detail and gives three applications illustrating the use of some of the numerous programmable features available. It should be evident from these discussions that the 8259A is an extremely flexible and easily programmable member of the Intel® MCS-80, MCS-85, MCS-86, and MCS-88 families.

APPENDIX A

This table is provided merely for reference information between the "Operation of the 8259A" and "Programming the 8259A" sections of this application note. It shouldn't be used as a programming reference guide (see "Programming the 8259A").

Operational Description	Command Words	Bits
MCS-80/85 Mode	ICW1, ICW4*	IC4, μPM*
Address Interval for MCS-80/85 Mode	ICW1	ADI
Interrupt Vector Address for MCS-80/85 Mode	ICW1, ICW2	A5–A15
MCS-86/88 Mode	ICW1, ICW4	IC4, μPM
Interrupt Vector Byte for MCS-86/88 Mode	ICW2	T3–T7
Fully Nested Mode	OCW-Default	—
Non-Specific EOI Command	OCW2	EOI
Specific EOI Command	OCW2	SEOI, EOI, LO–L2
Automatic EOI Mode	ICW1, ICW4	IC4, AEOI
Rotate On Non-Specific EOI Command	OCW2	EOI
Rotate In Automatic EOI Mode	OCW2	R, SEOI, EOI
Set Priority Command	OCW2	L0–L2
Rotate on Specific EOI Command	OCW2	R, SEOI, EOI
Interrupt Mask Register	OCW1	M0–M7
Special Mask Mode	OCW3	ESMM-SMM
Level Triggered Mode	ICW1	LTIM
Edge Triggered Mode	ICW1	LTIM
Read Register Command, IRR	OCW3	ERIS, RIS
Read Register Command, ISR	OCW3	ERIS, RIS
Read IMR	OCW1	M0–M7
Poll Command	OCW3	P
Cascade Mode	ICW1, ICW3	SNGL, S0–7, ID0–2
Special Fully Nested Mode	ICW1, ICW4	IC4, SFNM
Buffered Mode	ICW1, ICW4	IC4, BUF, M/S

*Only needed if ICW4 is used for purposes other than μP mode set.

```
MCS-86 ASSEMBLER    TCI59A                                                              PAGE    1


ISIS-II MCS-86 ASSEMBLER V1.0 ASSEMBLY OF MODULE TCI59A
OBJECT MODULE PLACED IN :F1:TCI59A.OBJ
ASSEMBLER INVOKED BY:  :F1:ASM86 :F1:TCI59A.SRC

LOC  OBJ                 LINE    SOURCE

                                 ;******************* TIMER CONTROLLED INTERRUPTS ******************
                                 ;
                                 ;
                                 ;
                                 ;                   EXTRA SEGMENT DECLARATIONS
                                 ;
----                             EXTRA SEGMENT
                                 ;
0120                                     ORG     120H
0120 0401                        TP72IP  DW      INTR72          ;TYPE 72 INSTRUCTION POINTER
0122 ????                        TP72CS  DW      ?               ;TYPE 72 CODE SEGMENT
0124 1801                        TP73IP  DW      INTR73          ;TYPE 73 INSTRUCTION POINTER
0126 ????                        TP73CS  DW      ?               ;TYPE 73 CODE SEGMENT
0128 3001                        TP74IP  DW      INTR74          ;TYPE 74 INSTRUCTION POINTER
012A ????                        TP74CS  DW      ?               ;TYPE 74 CODE SEGMENT
012C 4801                        TP75IP  DW      INTR75          ;TYPE 75 INSTRUCTION POINTER
012E ????                        TP75CS  DW      ?               ;TYPE 75 CODE SEGMENT
0130 6001                        TP76IP  DW      INTR76          ;TYPE 76 INSTRUCTION POINTER
0132 ????                        TP76CS  DW      ?               ;TYPE 76 CODE SEGMENT
0134 7801                        TP77IP  DW      INTR77          ;TYPE 77 INSTRUCTION POINTER
0136 ????                        TP77CS  DW      ?               ;TYPE 77 CODE SEGMENT
                                 ;
----                             EXTRA ENDS
                                 ;
                                 ;                   DATA SEGMENT DECLARATIONS
                                 ;
----                             DATA    SEGMENT
                                 ;
0000 ????                        STACK1  DW      ?                         ;VARIABLE TO SAVE CALL ADDRESS
0002 ????                        AXTEMP  DW      ?                         ;VARIABLE TO SAVE AX REGISTER
0004 ??                          DIGIT   DB      ?                         ;VARIABLE TO SAVE SELECTED DIGIT
                                 ;
----                             DATA    ENDS
                                 ;
                                 ;                   CODE SEGMENT DECLARATION
                                 ;
----                             CODE    SEGMENT
                                 ;
                                 ASSUME  ES:EXTRA,DS:DATA,CS:CODE
                                 ;
                                 ;                   INITIALIZE REGISTERS
                                 ;
0000 B80000                      START:  MOV     AX,0H                     ;EXTRA SEGMENT AT 0H
0003 8EC0                                MOV     ES,AX
0005 B87000                              MOV     AX,70H                    ;DATA SEGMENT AT 700H
0008 8ED8                                MOV     DS,AX
000A B87800                              MOV     AX,78H                    ;STACK SEGMENT AT 780H
000D 8ED0                                MOV     SS,AX
000F BC8000                              MOV     SP,80H                    ;STACK POINTER AT 80H (STACK=800H)
```

```
MCS-86 ASSEMBLER    TCI59A                                                        PAGE    2

LOC  OBJ              LINE  SOURCE

                        50  ;
                        51  ;               LOAD INTERRUPT VECTOR TABLE
                        52  ;
0012 B80401             53  TYPES:  MOV     AX,OFFSET (INTR72)      ;LOAD TYPE 72
0015 26A32001           54          MOV     TP72IP,AX
0019 268C0E2201         55          MOV     TP72CS,CS
001E B81801             56          MOV     AX,OFFSET (INTR73)      ;LOAD TYPE 73
0021 26A32401           57          MOV     TP73IP,AX
0025 268C0E2601         58          MOV     TP73CS,CS
002A B83001             59          MOV     AX,OFFSET (INTR74)      ;LOAD TYPE 74
002D 26A32801           60          MOV     TP74IP,AX
0031 268C0E2A01         61          MOV     TP74CS,CS
0036 B84801             62          MOV     AX,OFFSET (INTR75)      ;LOAD TYPE 75
0039 26A32C01           63          MOV     TP75IP,AX
003D 268C0E2E01         64          MOV     TP75CS,CS
0042 B86001             65          MOV     AX,OFFSET (INTR76)      ;LOAD TYPE 76
0045 26A33001           66          MOV     TP76IP,AX
0049 268C0E3201         67          MOV     TP76CS,CS
004E B87801             68          MOV     AX,OFFSET (INTR77)      ;LOAD TYPE 77
0051 26A33401           69          MOV     TP77IP,AX
0055 268C0E3601         70          MOV     TP77CS,CS
                        71  ;
                        72  ;               8253 INITIALIZATION
                        73  ;
005A BA0EFF             74  SET531: MOV     DX,0FF0EH               ;8253 #1 CONTROL WORD
005D B036               75          MOV     AL,36H                  ;COUNTER 0, MODE 3, BINARY
005F EE                 76          OUT     DX,AL
0060 B071               77          MOV     AL,71H                  ;COUNTER 1, MODE 0, BCD
0062 EE                 78          OUT     DX,AL
0063 B0B5               79          MOV     AL,0B5H                 ;COUNTER 2, MODE 2, BCD
0065 EE                 80          OUT     DX,AL
0066 BA08FF             81          MOV     DX,0FF08H               ;LOAD COUNTER 0 (10MS)
0069 B0A8               82          MOV     AL,0A8H                 ;LSB
006B EE                 83          OUT     DX,AL
006C B061               84          MOV     AL,61H                  ;MSB
006E EE                 85          OUT     DX,AL
006F BA0CFF             86          MOV     DX,0FF0CH               ;LOAD COUNTER 2 (2SEC)
0072 B000               87          MOV     AL,00H                  ;LSB
0074 EE                 88          OUT     DX,AL
0075 B002               89          MOV     AL,02H                  ;MSB
0077 EE                 90          OUT     DX,AL
0078 BA16FF             91  SET532: MOV     DX,0FF16H               ;8253 #2 CONTROL WORD
007B B03B               92          MOV     AL,3BH                  ;COUNTER 0, MODE 5,BCD
007D EE                 93          OUT     DX,AL
007E B07B               94          MOV     AL,7BH                  ;COUNTER 1, MODE 5, BCD
0080 EE                 95          OUT     DX,AL
0081 B0BB               96          MOV     AL,0BBH                 ;COUNTER 2, MODE 5, BCD
0083 EE                 97          OUT     DX,AL
0084 BA10FF             98          MOV     DX,0FF10H               ;LOAD COUNTER 0 (.5SEC)
0087 B050               99          MOV     AL,50H                  ;LSB
0089 EE                100          OUT     DX,AL
008A B000              101          MOV     AL,00H                  ;MSB
008C EE                102          OUT     DX,AL
008D BA12FF            103          MOV     DX,0FF12H               ;LOAD COUNTER 1 (1SEC)
0090 B000              104          MOV     AL,00H                  ;LSB
```

```
MCS-86 ASSEMBLER    TCI59A                                                        PAGE  3

LOC  OBJ            SOURCE

0092 EE                      OUT     DX,AL
0093 B001                    MOV     AL,01H                  ;MSB
0095 EE                      OUT     DX,AL
0096 BA14FF                  MOV     DX,0FF14H               ;LOAD COUNTER 2 (1.5SEC)
0099 B050                    MOV     AL,50H                  ;LSB
009B EE                      OUT     DX,AL
009C B001                    MOV     AL,01H                  ;MSB
009E EE                      OUT     DX,AL
                    ;
                    ;                8259A INITIALIZATION
                    ;
009F BA00FF         SET59A:  MOV     DX,0FF00H               ;8259A A0=0
00A2 B013                    MOV     AL,13H                  ;ICW1-LTIM=0,S=1,IC4=1
00A4 EE                      OUT     DX,AL
00A5 BA02FF                  MOV     DX,0FF02H               ;8259A A0=1
00A8 B048                    MOV     AL, 48H                 ;ICW2-INTERRUPT TYPE 72 (120H)
00AA EE                      OUT     DX,AL
00AB B003                    MOV     AL, 03H                 ;ICW4-SFNM=0,BUF=0,AEOI=1,MPM=1
00AD EE                      OUT     DX,AL
00AE B0E0                    MOV     AL,0E0H                 ;OCW1-MASK IR5,6,7 (NOT USED)
00B0 EE                      OUT     DX,AL
                    ;
                    ;                8279 INITIALIZATION
                    ;
00B1 BAEAFF         SET79:   MOV     DX,0FFEAH               ;8279 COMMAND WORDS AND STATUS
00B4 B0D0                    MOV     AL,0D0H                 ;CLEAR DISPLAY
00B6 EE                      OUT     DX,AL
00B7 EC             WAIT79:  IN      AL,DX                   ;READ STATUS
00B8 D0C0                    ROL     AL,1                    ;"DU" BIT TO CARRY
00BA 72FB                    JB      WAIT79                  ;JUMP IF DISPLAY IS UNAVAILABLE
00BC B087                    MOV     AL, 87H                 ;DIGIT 8
00BE EE                      OUT     DX,AL
00BF BAE8FF                  MOV     DX,0FFE8H               ;8279 DATA WORD
00C2 B006                    MOV     AL,06H                  ;CHARACTER "I"
00C4 EE                      OUT     DX,AL
00C5 BAEAFF                  MOV     DX,0FFEAH               ;8279 COMMAND WORD
00C8 B086                    MOV     AL,86H                  ;DIGIT 7
00CA EE                      OUT     DX,AL
00CB BAE8FF                  MOV     DX,0FFE8H               ;8279 DATA WORD
00CE B050                    MOV     AL,50H                  ;CHARACTER "R"
00D0 EE                      OUT     DX,AL
00D1 FB                      STI                             ;ENABLE INTERRUPTS
                    ;
                    ;
                    ;                DUMMY PROGRAM
                    ;
00D2 EBFE           DUMMY:   JMP     DUMMY                   ;WAIT FOR INTERRUPT
                    ;
                    ;
00D4 A30200         SAVE:    MOV     AXTEMP,AX               ;SAVE AX
00D7 58                      POP     AX                      ;POP CALL RETURN ADDRESS
00D8 A30000                  MOV     STACK1,AX               ;SAVE CALL RETURN ADDRESS
00DB A10200                  MOV     AX,AXTEMP               ;RESTORE AX
00DE 50                      PUSH    AX                      ;SAVE PROCESSOR STATUS
00DF 53                      PUSH    BX
```

```
MCS-86 ASSEMBLER    TCI59A                                                       PAGE    4

LOC  OBJ            LINE   SOURCE

00E0 51                           PUSH   CX
00E1 52                           PUSH   DX
00E2 55                           PUSH   BP
00E3 56                           PUSH   SI
00E4 57                           PUSH   DI
00E5 1E                           PUSH   DS
00E6 06                           PUSH   ES
00E7 A10000                       MOV    AX,STACK1             ;RESTORE CALL RETURN ADDRESS
00EA 50                           PUSH   AX                    ;PUSH CALL RETURN ADDRESS
00EB C3                           RET

00EC 58                    RESTOR: POP   AX                    ;POP CALL RETURN ADDRESS
00ED A30000                       MOV    STACK1,AX             ;SAVE CALL RETURN ADDRESS
00F0 07                           POP    ES                    ;RESTORE PROCESSOR STATUS
00F1 1F                           POP    DS
00F2 5F                           POP    DI
00F3 5E                           POP    SI
00F4 5D                           POP    BP
00F5 5A                           POP    DX
00F6 59                           POP    CX
00F7 5B                           POP    BX
00F8 58                           POP    AX
00F9 A30200                       MOV    AXTEMP,AX             ;SAVE AX
00FC A10000                       MOV    AX,STACK1             ;RESTORE CALL RETURN ADDRESS
00FF 50                           PUSH   AX                    ;PUSH CALL RETURN ADDRESS
0100 A10200                       MOV    AX,AXTEMP             ;RESTORE AX
0103 C3                           RET
                           ;
                           ;
                           ;             INTERRUPT 72, CLEAR DISPLAY,IR0 8259A
                           ;
0104 E8CDFF                INTR72: CALL  SAVE                  ;ROUTINE TO SAVE PROCESSOR STATUS
0107 BAEAFF                       MOV    DX,0FFEAH             ;8279 COMMAND WORD
010A A00400                       MOV    AL,DIGIT              ;SELECTED LED DIGIT
010D EE                           OUT    DX,AL
010E BAE8FF                       MOV    DX,0FFE8H             ;8279 DATA
0111 B000                         MOV    AL,00H                ;BLANK OUT DIGIT
0113 EE                           OUT    DX,AL
0114 E8D5FF                       CALL   RESTOR                ;ROUTINE TO RESTORE PROCESSOR STATUS
0117 CF                           IRET                         ;RETURN FROM INTERRUPT
                           ;
                           ;
                           ;             INTERRUPT 73, IR1 8259A
                           ;
0118 E8B9FF                INTR73: CALL  SAVE                  ;ROUTINE TO SAVE PROCESSOR STATUS
011B BAEAFF                       MOV    DX,0FFEAH             ;8279 COMMAND WORD
011E B080                         MOV    AL,80H                ;LED DISPLAY DIGIT 1
0120 A20400                       MOV    DIGIT,AL
0123 EE                           OUT    DX,AL
0124 BAE8FF                       MOV    DX,0FFE8H             ;8279 DATA
0127 B006                         MOV    AL,06H                ;CHARACTER "1"
0129 EE                           OUT    DX,AL
012A CD4D                         INT    77                    ;TIMER DELAY FOR LED ON TIME
012C E8BDFF                       CALL   RESTOR                ;ROUTINE TO RESTORE PROCESSOR STATUS
012F CF                           IRET                         ;RETURN FROM INTERRUPT
```

```
MCS-86 ASSEMBLER    TCI59A                                              PAGE    5

LOC  OBJ          LINE  SOURCE

                        ;
                        ;
                        ;                  INTERRUPT 74, IR2 8259A
                        ;
0130 E8A1FF             INTR74: CALL   SAVE               ;ROUTINE TO SAVE PROCESSOR STATUS
0133 BAEAFF                     MOV    DX,0FFEAH          ;8279 COMMAND WORD
0136 B081                       MOV    AL,81H             ;LED DISPLAY DIGIT 2
0138 A20400                     MOV    DIGIT,AL
013B EE                         OUT    DX,AL
013C BAE8FF                     MOV    DX,0FFE8H          ;8279 DATA
013F B05B                       MOV    AL,5BH             ;CHARACTER "2"
0141 EE                         OUT    DX,AL
0142 CD4D                       INT    77                 ;TIMER DELAY FOR LED ON TIME
0144 E8A5FF                     CALL   RESTOR             ;ROUTINE TO RESTORE PROCESSOR STATUS
0147 CF                         IRET                      ;RETURN FROM INTERRUPT
                        ;
                        ;
                        ;                  INTERRUPT 75, IR3 8259A
                        ;
0148 E889FF             INTR75: CALL   SAVE               ;ROUTINE TO SAVE PROCESSOR STATUS
014B BAEAFF                     MOV    DX,0FFEAH          ;8279 COMMAND WORD
014E B082                       MOV    AL,82H             ;LED DISPLAY DIGIT 3
0150 A20400                     MOV    DIGIT,AL
0153 EE                         OUT    DX,AL
0154 BAE8FF                     MOV    DX,0FFE8H          ;8279 DATA
0157 B04F                       MOV    AL,4FH             ;CHARACTER "3"
0159 EE                         OUT    DX,AL
015A CD4D                       INT    77                 ;TIMER DELAY FOR LED ON TIME
015C E88DFF                     CALL   RESTOR             ;ROUTINE TO RESTORE PROCESSOR STATUS
015F CF                         IRET                      ;RETURN FROM INTERRUPT
                        ;
                        ;
                        ;                  INTERRUPT 76, IR4 8259A
                        ;
0160 E871FF             INTR76: CALL   SAVE               ;ROUTINE TO SAVE PROCESSOR STATUS
0163 BAEAFF                     MOV    DX,0FFEAH          ;8279 COMMAND WORD
0166 B083                       MOV    AL,83H             ;LED DISPLAY DIGIT 4
0168 A20400                     MOV    DIGIT,AL
016B EE                         OUT    DX,AL
016C BAE8FF                     MOV    DX,0FFE8H          ;8279 DATA
016F B066                       MOV    AL,66H             ;CHARACTER "4"
0171 EE                         OUT    DX,AL
0172 CD4D                       INT    77                 ;TIMER DELAY FOR LED ON TIME
0174 E875FF                     CALL   RESTOR             ;ROUTINE TO RESTORE PROCESSOR STATUS
0177 CF                         IRET                      ;RETURN FROM INTERRUPT
                        ;
                        ;
                        ;                  INTERRUPT 77, TIMER DELAY, SOFTWARE CONTROLLED
                        ;
0178 BA0AFF             INTR77: MOV    DX,0FF0AH          ;LOAD COUNTER 1 8253 #1 (250 MSEC)
017B B025                       MOV    AL,25H             ;LSB
017D EE                         OUT    DX,AL
017E B000                       MOV    AL,00H             ;MSB
0180 EE                         OUT    DX,AL
0181 CF                         IRET                      ;RETURN FROM INTERRUPT
```

```
MCS-86 ASSEMBLER    TCI59A                                              PAGE    6


LOC  OBJ                  LINE   SOURCE

                           270   ;
                           271   ;
----                       272   CODE    ENDS;
                           273   ;
                           274   ;
 0000                      275           END     START

SYMBOL TABLE LISTING
------ ----- -------


NAME     TYPE     VALUE  ATTRIBUTES

??SEG .  SEGMENT         SIZE=0000H PARA PUBLIC
AXTEMP   V WORD   0002H  DATA
CODE. .  SEGMENT         SIZE=0182H PARA
DATA. .  SEGMENT         SIZE=0005H PARA
DIGIT .  V BYTE   0004H  DATA
DUMMY .  L NEAR   00D2H  CODE
EXTRA .  SEGMENT         SIZE=0138H PARA
INTR72.  L NEAR   0104H  CODE
INTR73.  L NEAR   0118H  CODE
INTR74.  L NEAR   0130H  CODE
INTR75.  L NEAR   0148H  CODE
INTR76.  L NEAR   0160H  CODE
INTR77.  L NEAR   0178H  CODE
RESTOR.  L NEAR   00ECH  CODE
SAVE. .  L NEAR   00D4H  CODE
SET531.  L NEAR   005AH  CODE
SET532.  L NEAR   0078H  CODE
SET59A.  L NEAR   009FH  CODE
SET79 .  L NEAR   00B1H  CODE
STACK1.  V WORD   0000H  DATA
START .  L NEAR   0000H  CODE
TP72CS.  V WORD   0122H  EXTRA
TP72IP.  V WORD   0120H  EXTRA
TP73CS.  V WORD   0126H  EXTRA
TP73IP.  V WORD   0124H  EXTRA
TP74CS.  V WORD   012AH  EXTRA
TP74IP.  V WORD   0128H  EXTRA
TP75CS.  V WORD   012EH  EXTRA
TP75IP.  V WORD   012CH  EXTRA
TP76CS.  V WORD   0132H  EXTRA
TP76IP.  V WORD   0130H  EXTRA
TP77CS.  V WORD   0136H  EXTRA
TP77IP.  V WORD   0134H  EXTRA
TYPES .  L NEAR   0012H  CODE
WAIT79.  L NEAR   00B7H  CODE


ASSEMBLY COMPLETE, NO ERRORS FOUND
```

8086, 8088, 80186, 80188 Microprocessors

3

intel

8086
16-BIT HMOS MICROPROCESSOR

8086/8086-2/8086-1*

- **Direct Addressing Capability 1 MByte of Memory**
- **Architecture Designed for Powerful Assembly Language and Efficient High Level Languages.**
- **14 Word, by 16-Bit Register Set with Symmetrical Operations**
- **24 Operand Addressing Modes**
- **Bit, Byte, Word, and Block Operations**
- **8 and 16-Bit Signed and Unsigned Arithmetic in Binary or Decimal Including Multiply and Divide**
- **Range of Clock Rates:**
  5 MHz for 8086,
  8 MHz for 8086-2,
  10 MHz for 8086-1
- **MULTIBUS® System Compatible Interface**
- **Available in EXPRESS**
  - Standard Temperature Range
  - Extended Temperature Range
- **Available in 40-Lead Cerdip and Plastic Package**
  (See Packaging Spec, Order #231369)

The Intel 8086 high performance 16-bit CPU is available in three clock rates: 5, 8 and 10 MHz. The CPU is implemented in N-Channel, depletion load, silicon gate technology (HMOS), and packaged in a 40-pin CERDIP or plastic package. The 8086 operates in both single processor and multiple processor configurations to achieve high performance levels.

*Changes from the 1985 handbook specification have been made for the 8086-1. See A.C. Characteristics TGVCH and TCLGL.

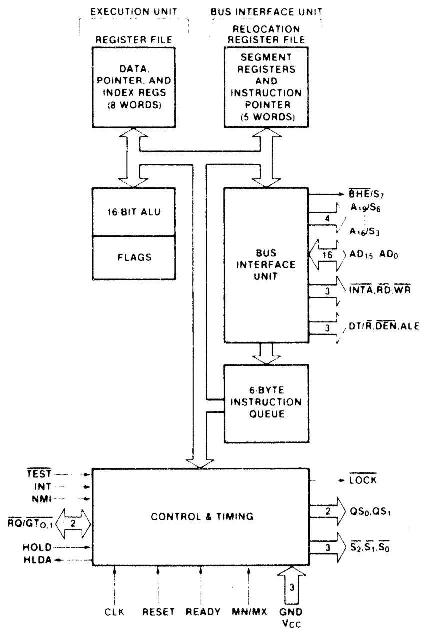


Figure 1. 8086 CPU Block Diagram

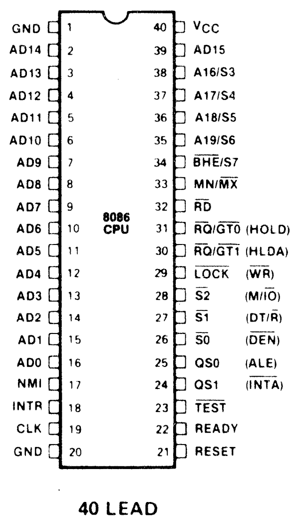


Figure 2. 8086 Pin Configuration

Table 1. Pin Description

The following pin function descriptions are for 8086 systems in either minimum or maximum mode. The "Local Bus" in these descriptions is the direct multiplexed bus interface connection to the 8086 (without regard to additional bus buffers).

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>AD_{15}-AD_0</td><td>2-16, 39</td><td>I/O</td><td>Address Data Bus: These lines constitute the time multiplexed memory/IO address (T_1) and data (T_2, T_3, T_W, T_4) bus. A_0 is analogous to $\overline{BHE}$ for the lower byte of the data bus, pins D_7-D_0. It is LOW during T_1 when a byte is to be transferred on the lower portion of the bus in memory or I/O operations. Eight-bit oriented devices tied to the lower half would normally use A_0 to condition chip select functions. (See $\overline{BHE}$.) These lines are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge."</td></tr>
<tr><td>A_{19}/S_6,
A_{18}/S_5,
A_{17}/S_4,
A_{16}/S_3</td><td>35-38</td><td>O</td><td>Address/Status: During T_1 these are the four most significant address lines for memory operations. During I/O operations these lines are LOW. During memory and I/O operations, status information is available on these lines during T_2, T_3, T_W, and T_4. The status of the interrupt enable FLAG bit (S_5) is updated at the beginning of each CLK cycle. A_{17}/S_4 and A_{16}/S_3 are encoded as shown.

This information indicates which relocation register is presently being used for data accessing.

These lines float to 3-state OFF during local bus "hold acknowledge."
<table>
<tr><th>A_{17}/S_4</th><th>A_{16}/S_3</th><th>Characteristics</th></tr>
<tr><td>0 (LOW)</td><td>0</td><td>Alternate Data</td></tr>
<tr><td>0</td><td>1</td><td>Stack</td></tr>
<tr><td>1 (HIGH)</td><td>0</td><td>Code or None</td></tr>
<tr><td>1</td><td>1</td><td>Data</td></tr>
<tr><td>S_6 is 0 (LOW)</td><td></td><td></td></tr>
</table></td></tr>
<tr><td>$\overline{BHE}/S_7$</td><td>34</td><td>O</td><td>Bus High Enable/Status: During T_1 the bus high enable signal ($\overline{BHE}$) should be used to enable data onto the most significant half of the data bus, pins D_{15}-D_8. Eight-bit oriented devices tied to the upper half of the bus would normally use $\overline{BHE}$ to condition chip select functions. $\overline{BHE}$ is LOW during T_1 for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the high portion of the bus. The S_7 status information is available during T_2, T_3, and T_4. The signal is active LOW, and floats to 3-state OFF in "hold." It is LOW during T_1 for the first interrupt acknowledge cycle.
<table>
<tr><th>$\overline{BHE}$</th><th>A_0</th><th>Characteristics</th></tr>
<tr><td>0</td><td>0</td><td>Whole word</td></tr>
<tr><td>0</td><td>1</td><td>Upper byte from/ to odd address</td></tr>
<tr><td>1</td><td>0</td><td>Lower byte from/ to even address</td></tr>
<tr><td>1</td><td>1</td><td>None</td></tr>
</table></td></tr>
<tr><td>$\overline{RD}$</td><td>32</td><td>O</td><td>Read: Read strobe indicates that the processor is performing a memory of I/O read cycle, depending on the state of the S_2 pin. This signal is used to read devices which reside on the 8086 local bus. $\overline{RD}$ is active LOW during T_2, T_3 and T_W of any read cycle, and is guaranteed to remain HIGH in T_2 until the 8086 local bus has floated.

This signal floats to 3-state OFF in "hold acknowledge."</td></tr>
<tr><td>READY</td><td>22</td><td>I</td><td>READY: is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The READY signal from memory/IO is synchronized by the 8284A Clock Generator to form READY. This signal is active HIGH. The 8086 READY input is not synchronized. Correct operation is not guaranteed if the setup and hold times are not met.</td></tr>
<tr><td>INTR</td><td>18</td><td>I</td><td>Interrupt Request: is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.</td></tr>
<tr><td>$\overline{TEST}$</td><td>23</td><td>I</td><td>$\overline{TEST}$: input is examined by the "Wait" instruction. If the $\overline{TEST}$ input is LOW execution continues, otherwise the processor waits in an "Idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.</td></tr>
</table>

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
NMI	17	I	**Non-maskable interrupt:** an edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software. A transition from a LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.
RESET	21	I	**Reset:** causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. It restarts execution, as described in the Instruction Set description, when RESET returns LOW. RESET is internally synchronized.
CLK	19	I	**Clock:** provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.
V_{CC}	40		**V_{CC}:** + 5V power supply pin.
GND	1, 20		**Ground**
MN/$\overline{MX}$	33	I	**Minimum/Maximum:** indicates what mode the processor is to operate in. The two modes are discussed in the following sections.

The following pin function descriptions are for the 8086/8288 system in maximum mode (i.e., MN/$\overline{MX}$ = V_{SS}). Only the pin functions which are unique to maximum mode are described; all other pin functions are as described above.

$\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$	26-28	O	**Status:** active during T_4, T_1, and T_2 and is returned to the passive state (1,1,1) during T_3 or during T_W when READY is HIGH. This status is used by the 8288 Bus Controller to generate all memory and I/O access control signals. Any change by $\overline{S_2}$, $\overline{S_1}$, or $\overline{S_0}$ during T_4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T_3 or T_W is used to indicate the end of a bus cycle. These signals float to 3-state OFF in "hold acknowledge." These status lines are encoded as shown.
$\overline{RQ}/\overline{GT}_0$, $\overline{RQ}/\overline{GT}_1$	30, 31	I/O	**Request/Grant:** pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with $\overline{RQ}/\overline{GT}_0$ having higher priority than $\overline{RQ}/\overline{GT}_1$. $\overline{RQ}/\overline{GT}$ has an internal pull-up resistor so may be left unconnected. The request/grant sequence is as follows (see Figure 9): 1. A pulse of 1 CLK wide from another local bus master indicates a local bus request ("hold") to the 8086 (pulse 1). 2. During a T_4 or T_I clock cycle, a pulse 1 CLK wide from the 8086 to the requesting master (pulse 2), indicates that the 8086 has allowed the local bus to float and that it will enter the "hold acknowledge" state at the next CLK. The CPU's bus interface unit is disconnected logically from the local bus during "hold acknowledge." 3. A pulse 1 CLK wide from the requesting master indicates to the 8086 (pulse 3) that the "hold" request is about to end and that the 8086 can reclaim the local bus at the next CLK. Each master-master exchange of the local bus is a sequence of 3 pulses. There must be one dead CLK cycle after each bus exchange. Pulses are active LOW. If the request is made while the CPU is performing a memory cycle, it will release the local bus during T_4 of the cycle when all the following conditions are met: 1. Request occurs on or before T_2. 2. Current cycle is not the low byte of a word (on an odd address). 3. Current cycle is not the first acknowledge of an interrupt acknowledge sequence. 4. A locked instruction is not currently executing.

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Characteristics
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	Halt
1 (HIGH)	0	0	Code Access
1	0	1	Read Memory
1	1	0	Write Memory
1	1	1	Passive

Table 1. Pin Description (Continued)

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td></td><td></td><td></td><td>If the local bus is idle when the request is made the two possible events will follow:
1. Local bus will be released during the next clock.
2. A memory cycle will start within 3 clocks. Now the four rules for a currently active memory cycle apply with condition number 1 already satisfied.</td></tr>
<tr><td>$\overline{\text{LOCK}}$</td><td>29</td><td>O</td><td>$\overline{\text{LOCK}}$: output indicates that other system bus masters are not to gain control of the system bus while $\overline{\text{LOCK}}$ is active LOW. The $\overline{\text{LOCK}}$ signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW, and floats to 3-state OFF in "hold acknowledge."</td></tr>
<tr><td>QS_1, QS_0</td><td>24, 25</td><td>O</td><td>Queue Status: The queue status is valid during the CLK cycle after which the queue operation is performed.
QS_1 and QS_0 provide status to allow external tracking of the internal 8086 instruction queue.
<table>
<tr><th>QS_1</th><th>QS_0</th><th>CHARACTERISTICS</th></tr>
<tr><td>0 (LOW)</td><td>0</td><td>No Operation</td></tr>
<tr><td>0</td><td>1</td><td>First Byte of Op Code from Queue</td></tr>
<tr><td>1 (HIGH)</td><td>0</td><td>Empty the Queue</td></tr>
<tr><td>1</td><td>1</td><td>Subsequent Byte from Queue</td></tr>
</table></td></tr>
</table>

The following pin function descriptions are for the 8086 in minimum mode (i.e., MN/$\overline{\text{MX}}$ = V_{CC}). Only the pin functions which are unique to minimum mode are described; all other pin functions are as described above.

Symbol	Pin No.	Type	Name and Function
M/$\overline{\text{IO}}$	28	O	**Status line:** logically equivalent to S_2 in the maximum mode. It is used to distinguish a memory access from an I/O access. M/$\overline{\text{IO}}$ becomes valid in the T_4 preceding a bus cycle and remains valid until the final T_4 of the cycle (M = HIGH, IO = LOW). M/$\overline{\text{IO}}$ floats to 3-state OFF in local bus "hold acknowledge."
$\overline{\text{WR}}$	29	O	**Write:** indicates that the processor is performing a write memory or write I/O cycle, depending on the state of the M/$\overline{\text{IO}}$ signal. $\overline{\text{WR}}$ is active for T_2, T_3 and T_W of any write cycle. It is active LOW, and floats to 3-state OFF in local bus "hold acknowledge."
$\overline{\text{INTA}}$	24	O	**$\overline{\text{INTA}}$** is used as a read strobe for interrupt acknowledge cycles. It is active LOW during T_2, T_3 and T_W of each interrupt acknowledge cycle.
ALE	25	O	**Address Latch Enable:** provided by the processor to latch the address into the 8282/8283 address latch. It is a HIGH pulse active during T_1 of any bus cycle. Note that ALE is never floated.
DT/$\overline{\text{R}}$	27	O	**Data Transmit/Receive:** needed in minimum system that desires to use an 8286/8287 data bus transceiver. It is used to control the direction of data flow through the transceiver. Logically DT/$\overline{\text{R}}$ is equivalent to $\overline{S_1}$ in the maximum mode, and its timing is the same as for M/$\overline{\text{IO}}$. (T = HIGH, R = LOW.) This signal floats to 3-state OFF in local bus "hold acknowledge."
$\overline{\text{DEN}}$	26	O	**Data Enable:** provided as an output enable for the 8286/8287 in a minimum system which uses the transceiver. $\overline{\text{DEN}}$ is active LOW during each memory and I/O access and for INTA cycles. For a read or $\overline{\text{INTA}}$ cycle it is active from the middle of T_2 until the middle of T_4, while for a write cycle it is active from the beginning of T_2 until the middle of T_4. $\overline{\text{DEN}}$ floats to 3-state OFF in local bus "hold acknowledge."
HOLD, HLDA	31, 30	I/O	**HOLD:** indicates that another master is requesting a local bus "hold." To be acknowledged, HOLD must be active HIGH. The processor receiving the "hold" request will issue HLDA (HIGH) as an acknowledgement in the middle of a T_1 clock cycle. Simultaneous with the issuance of HLDA the processor will float the local bus and control lines. After HOLD is detected as being LOW, the processor will LOWer the HLDA, and when the processor needs to run another cycle, it will again drive the local bus and control lines. The same rules as for $\overline{\text{RQ}}$/$\overline{\text{GT}}$ apply regarding when the local bus will be released. HOLD is not an asynchronous input. External synchronization should be provided if the system cannot otherwise guarantee the setup time.

FUNCTIONAL DESCRIPTION

GENERAL OPERATION

The internal functions of the 8086 processor are partitioned logically into two processing units. The first is the Bus Interface Unit (BIU) and the second is the Execution Unit (EU) as shown in the block diagram of Figure 1.

These units can interact directly but for the most part perform as separate asynchronous operational processors. The bus interface unit provides the functions related to instruction fetching and queuing, operand fetch and store, and address relocation. This unit also provides the basic bus control. The overlap of instruction pre-fetching provided by this unit serves to increase processor performance through improved bus bandwidth utilization. Up to 6 bytes of the instruction stream can be queued while waiting for decoding and execution.

The instruction stream queuing mechanism allows the BIU to keep the memory utilized very efficiently. Whenever there is space for at least 2 bytes in the queue, the BIU will attempt a word fetch memory cycle. This greatly reduces "dead time" on the memory bus. The queue acts as a First-In-First-Out (FIFO) buffer, from which the EU extracts instruction bytes as required. If the queue is empty (following a branch instruction, for example), the first byte into the queue immediately becomes available to the EU.

The execution unit receives pre-fetched instructions from the BIU queue and provides un-relocated operand addresses to the BIU. Memory operands are passed through the BIU for processing by the EU, which passes results to the BIU for storage. See the Instruction Set description for further register set and architectural descriptions.

MEMORY ORGANIZATION

The processor provides a 20-bit address to memory which locates the byte being referenced. The memory is organized as a linear array of up to 1 million bytes, addressed as 00000(H) to FFFFF(H). The memory is logically divided into code, data, extra data, and stack segments of up to 64K bytes each, with each segment falling on 16-byte boundaries. (See Figure 3a.)

All memory references are made relative to base addresses contained in high speed segment registers. The segment types were chosen based on the addressing needs of programs. The segment register to be selected is automatically chosen according to the rules of the following table. All information in one segment type share the same logical attributes (e.g. code or data). By structuring memory into relocatable areas of similar characteristics and by automatically selecting segment registers, programs are shorter, faster, and more structured.

Word (16-bit) operands can be located on even or odd address boundaries and are thus not constrained to even boundaries as is the case in many 16-bit computers. For address and data operands, the least significant byte of the word is stored in the lower valued address location and the most significant byte in the next higher address location. The BIU automatically performs the proper number of memory accesses, one if the word operand is on an even byte boundary and two if it is on an odd byte boundary. Except for the performance penalty, this double access is transparent to the software. This performance penalty does not occur for instruction fetches, only word operands.

Physically, the memory is organized as a high bank (D_{15}-D_8) and a low bank (D_7-D_0) of 512K 8-bit bytes addressed in parallel by the processor's address lines

A_{19} - A_1. Byte data with even addresses is transferred on the D_7-D_0 bus lines while odd addressed byte data (A_0 HIGH) is transferred on the D_{15}-D_8 bus lines. The processor provides two enable signals, $\overline{BHE}$ and A_0, to selectively allow reading from or writing into either an odd byte location, even byte location, or both. The instruction stream is fetched from memory as words and is addressed internally by the processor to the byte level as necessary.

Memory Reference Need	Segment Register Used	Segment Selection Rule
Instructions	CODE (CS)	Automatic with all instruction prefetch.
Stack	STACK (SS)	All stack pushes and pops. Memory references relative to BP base register except data references.
Local Data	DATA (DS)	Data references when: relative to stack, destination of string operation, or explicitly overridden.
External (Global) Data	EXTRA (ES)	Destination of string operations: Explicitly selected using a segment override.

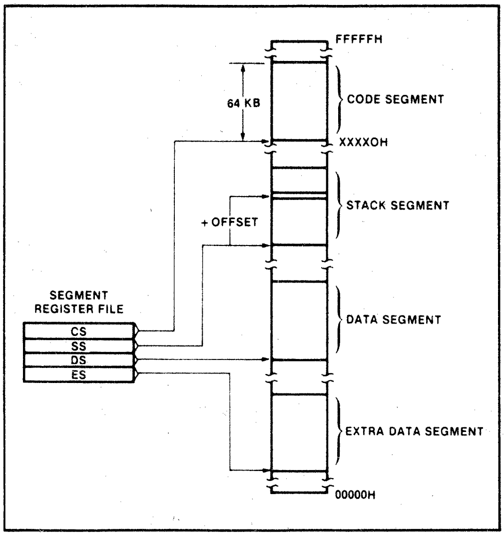


Figure 3a. Memory Organization

In referencing word data the BIU requires one or two memory cycles depending on whether or not the starting byte of the word is on an even or odd address, respectively. Consequently, in referencing word operands performance can be optimized by locating data on even address boundaries. This is an especially useful technique for using the stack, since odd address references to the stack may adversely affect the context switching time for interrupt processing or task multiplexing.

Certain locations in memory are reserved for specific CPU operations (see Figure 3b.) Locations from address FFFF0H through FFFFFH are reserved for operations including a jump to the initial program loading routine. Following RESET, the CPU will always begin execution at location FFFF0H where the jump must be. Locations 00000H through 003FFH are reserved for interrupt operations. Each of the 256 possible interrupt types has its service routine pointed to by a 4-byte pointer element consisting of a 16-bit segment address and a 16-bit offset address. The pointer elements are assumed to have been stored at the respective places in reserved memory prior to occurrence of interrupts.

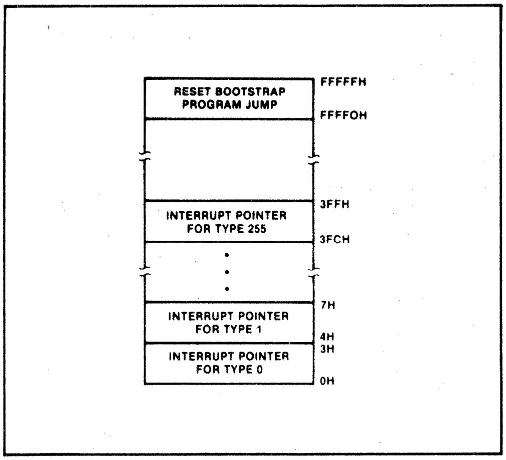


Figure 3b. Reserved Memory Locations

MINIMUM AND MAXIMUM MODES

The requirements for supporting minimum and maximum 8086 systems are sufficiently different that they cannot be done efficiently with 40 uniquely defined pins. Consequently, the 8086 is equipped with a strap pin (MN/$\overline{MX}$) which defines the system configuration. The definition of a certain subset of the pins changes dependent on the condition of the strap pin. When MN/$\overline{MX}$ pin is strapped to GND, the 8086 treats pins 24 through 31 in maximum mode. An 8288 bus controller interprets status information coded into $\overline{S_0}$, $\overline{S_2}$, $\overline{S_2}$ to generate bus timing and control signals compatible with the MULTIBUS® architecture. When the MN/$\overline{MX}$ pin is strapped to V_{CC}, the 8086 generates bus control signals itself on pins 24 through 31, as shown in parentheses in Figure 2. Examples of minimum mode and maximum mode systems are shown in Figure 4.

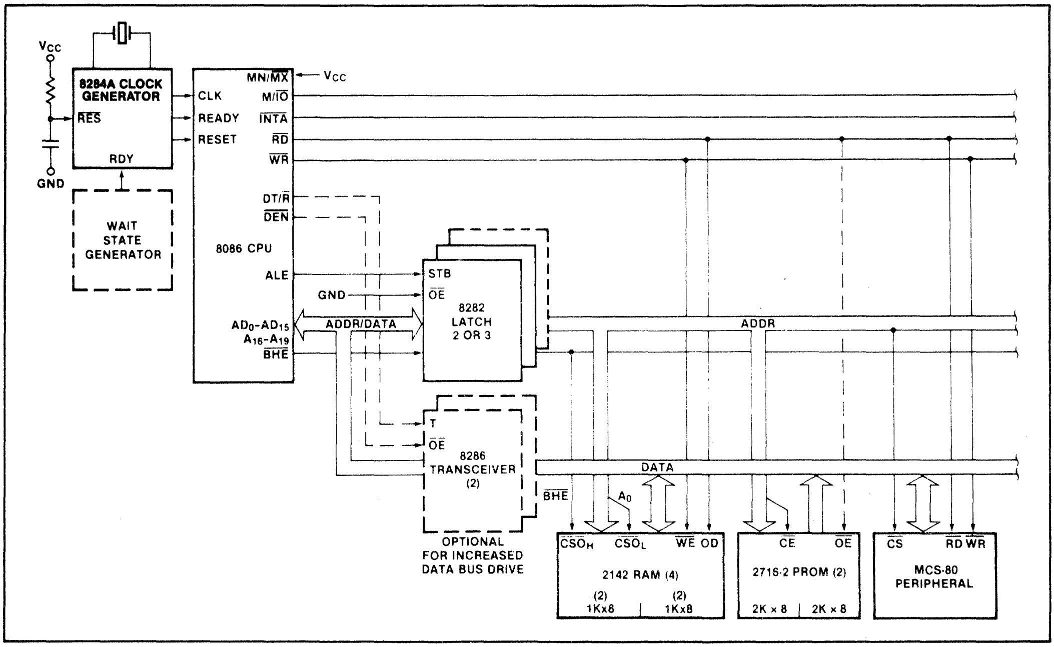


Figure 4a. Minimum Mode 8086 Typical Configuration

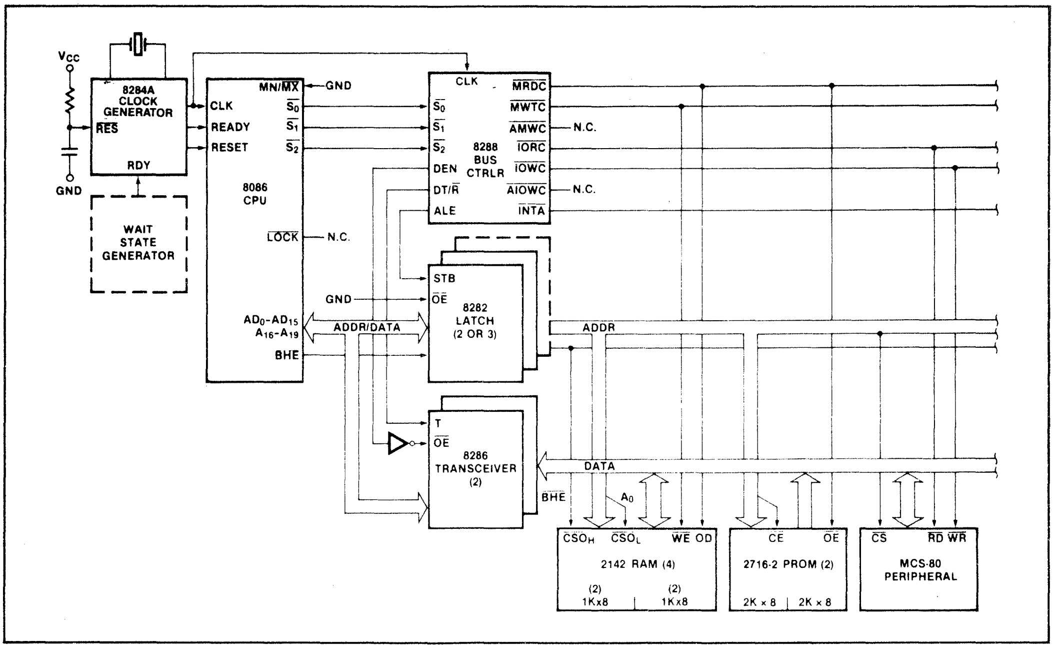


Figure 4b. Maximum Mode 8086 Typical Configuration

BUS OPERATION

The 8086 has a combined address and data bus commonly referred to as a time multiplexed bus. This technique provides the most efficient use of pins on the processor while permitting the use of a standard 40-lead package. This "local bus" can be buffered directly and used throughout the system with address latching provided on memory and I/O modules. In addition, the bus can also be demultiplexed at the processor with a single set of address latches if a standard non-multiplexed bus is desired for the system.

Each processor bus cycle consists of at least four CLK cycles. These are referred to as T_1, T_2, T_3 and T_4 (see Figure 5). The address is emitted from the processor during T_1 and data transfer occurs on the bus during T_3 and T_4. T_2 is used primarily for changing the direction of the bus during read operations. In the event that a "NOT READY" indication is given by the addressed device, "Wait" states (T_W) are inserted between T_3 and T_4. Each inserted "Wait" state is of the same duration as a CLK cycle. Periods can occur between 8086 bus cycles. These are referred to as "Idle" states (T_I) or inactive CLK cycles. The processor uses these cycles for internal housekeeping.

During T_1 of any bus cycle the ALE (Address Latch Enable) signal is emitted (by either the processor or the 8288 bus controller, depending on the MN/$\overline{MX}$ strap). At the trailing edge of this pulse, a valid address and certain status information for the cycle may be latched.

Status bits $\overline{S_0}$, $\overline{S_1}$, and $\overline{S_2}$ are used, in maximum mode, by the bus controller to identify the type of bus transaction according to the following table:

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	CHARACTERISTICS
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1 (HIGH)	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

Status bits S_3 through S_7 are multiplexed with high-order address bits and the $\overline{BHE}$ signal, and are therefore valid during T_2 through T_4. S_3 and S_4 indicate which segment register (see Instruction Set description) was used for this bus cycle in forming the address, according to the following table:

S_4	S_3	CHARACTERISTICS
0 (LOW)	0	Alternate Data (extra segment)
0	1	Stack
1 (HIGH)	0	Code or None
1	1	Data

S_5 is a reflection of the PSW interrupt enable bit. S_6=0 and S_7 is a spare status bit.

I/O ADDRESSING

In the 8086, I/O operations can address up to a maximum of 64K I/O byte registers or 32K I/O word registers. The I/O address appears in the same format as the memory address on bus lines A_{15}–A_0. The address lines A_{19}–A_{16} are zero in I/O operations. The variable I/O instructions which use register DX as a pointer have full address capability while the direct I/O instructions directly address one or two of the 256 I/O byte locations in page 0 of the I/O address space.

I/O ports are addressed in the same manner as memory locations. Even addressed bytes are transferred on the D_7–D_0 bus lines and odd addressed bytes on D_{15}–D_8. Care must be taken to assure that each register within an 8-bit peripheral located on the lower portion of the bus be addressed as even.

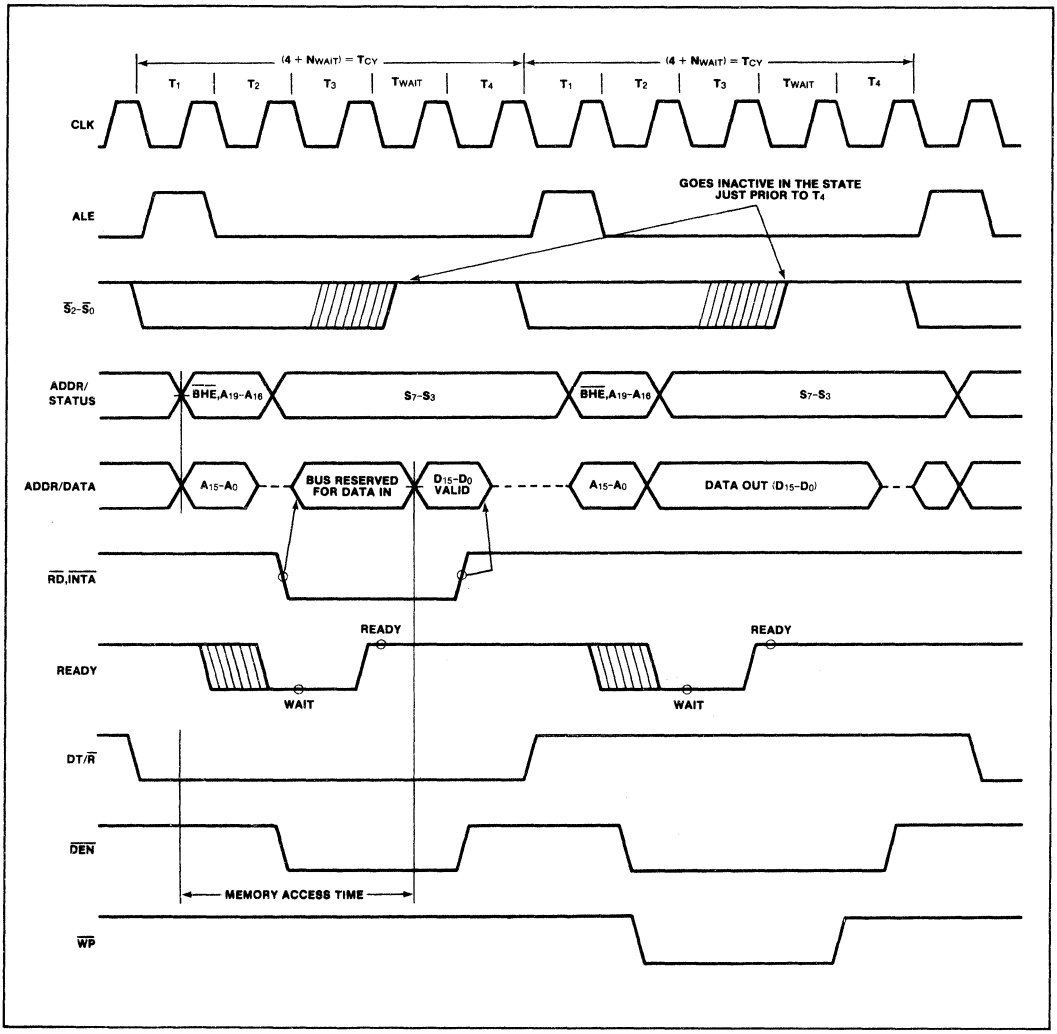


Figure 5. Basic System Timing

EXTERNAL INTERFACE

PROCESSOR RESET AND INITIALIZATION

Processor initialization or start up is accomplished with activation (HIGH) of the RESET pin. The 8086 RESET is required to be HIGH for greater than 4 CLK cycles. The 8086 will terminate operations on the high-going edge of RESET and will remain dormant as long as RESET is HIGH. The low-going transition of RESET triggers an internal reset sequence for approximately 10 CLK cycles. After this interval the 8086 operates normally beginning with the instruction in absolute location FFFF0H (see Figure 3B). The details of this operation are specified in the Instruction Set description of the MCS-86 Family User's Manual. The RESET input is internally synchronized to the processor clock. At initialization the HIGH-to-LOW transition of RESET must occur no sooner than 50 μs after power-up, to allow complete initialization of the 8086.

NMI may not be asserted prior to the 2nd CLK cycle following the end of RESET.

INTERRUPT OPERATIONS

Interrupt operations fall into two classes; software or hardware initiated. The software initiated interrupts and software aspects of hardware interrupts are specified in the Instruction Set description. Hardware interrupts can be classified as non-maskable or maskable.

Interrupts result in a transfer of control to a new program location. A 256-element table containing address pointers to the interrupt service program locations resides in absolute locations 0 through 3FFH (see Figure 3b), which are reserved for this purpose. Each element in the table is 4 bytes in size and corresponds to an interrupt "type". An interrupting device supplies an 8-bit type number, during the interrupt acknowledge sequence, which is used to "vector" through the appropriate element to the new interrupt service program location.

NON-MASKABLE INTERRUPT (NMI)

The processor provides a single non-maskable interrupt pin (NMI) which has higher priority than the maskable interrupt request pin (INTR). A typical use would be to activate a power failure routine. The NMI is edge-triggered on a LOW-to-HIGH transition. The activation of this pin causes a type 2 interrupt. (See Instruction Set description.)

NMI is required to have a duration in the HIGH state of greater than two CLK cycles, but is not required to be synchronized to the clock. Any high-going transition of NMI is latched on-chip and will be serviced at the end of the current instruction or between whole moves of a block-type instruction. Worst case response to NMI would be for multiply, divide, and variable shift instructions. There is no specification on the occurrence of the low-going edge; it may occur before, during, or after the servicing of NMI. Another high-going edge triggers another response if it occurs after the start of the NMI procedure. The signal must be free of logical spikes in general and be free of bounces on the low-going edge to avoid triggering extraneous responses.

MASKABLE INTERRUPT (INTR)

The 8086 provides a single interrupt request input (INTR) which can be masked internally by software with the resetting of the interrupt enable FLAG status bit. The interrupt request signal is level triggered. It is internally synchronized during each clock cycle on the high-going edge of CLK. To be responded to, INTR must be present (HIGH) during the clock period preceding the end of the current instruction or the end of a whole move for a block-type instruction. During the interrupt response sequence further interrupts are disabled. The enable bit is reset as part of the response to any interrupt (INTR, NMI, software interrupt or single-step), although the

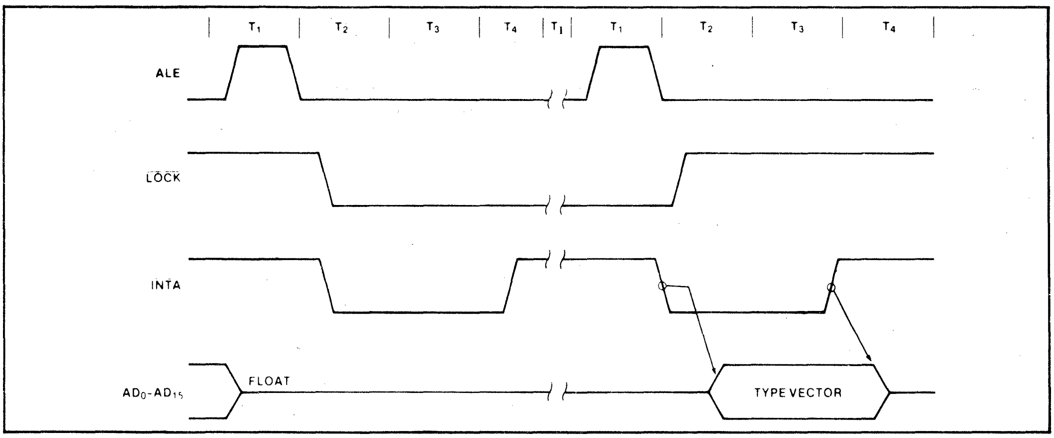


Figure 6. Interrupt Acknowledge Sequence

FLAGS register which is automatically pushed onto the stack reflects the state of the processor prior to the interrupt. Until the old FLAGS register is restored the enable bit will be zero unless specifically set by an instruction.

During the response sequence (figure 6) the processor executes two successive (back-to-back) interrupt acknowledge cycles. The 8086 emits the LOCK signal from T_2 of the first bus cycle until T_2 of the second. A local bus "hold" request will not be honored until the end of the second bus cycle. In the second bus cycle a byte is fetched from the external interrupt system (e.g., 8259A PIC) which identifies the source (type) of the interrupt. This byte is multiplied by four and used as a pointer into the interrupt vector lookup table. An INTR signal left HIGH will be continually responded to within the limitations of the enable bit and sample period. The INTERRUPT RETURN instruction includes a FLAGS pop which returns the status of the original interrupt enable bit when it restores the FLAGS.

HALT

When a software "HALT" instruction is executed the processor indicates that it is entering the "HALT" state in one of two ways depending upon which mode is strapped. In minimum mode, the processor issues one ALE with no qualifying bus control signals. In Maximum Mode, the processor issues appropriate HALT status on $\overline{S_2}\overline{S_1}\overline{S_0}$ and the 8288 bus controller issues one ALE. The 8086 will not leave the "HALT" state when a local bus "hold" is entered while in "HALT". In this case, the processor reissues the HALT indicator. An interrupt request or RESET will force the 8086 out of the "HALT" state.

READ/MODIFY/WRITE (SEMAPHORE) OPERATIONS VIA LOCK

The $\overline{\text{LOCK}}$ status information is provided by the processor when directly consecutive bus cycles are required during the execution of an instruction. This provides the processor with the capability of performing read/modify/write operations on memory (via the Exchange Register With Memory instruction, for example) without the possibility of another system bus master receiving intervening memory cycles. This is useful in multiprocessor system configurations to accomplish "test and set lock" operations. The $\overline{\text{LOCK}}$ signal is activated (forced LOW) in the clock cycle following the one in which the software "LOCK" prefix instruction is decoded by the EU. It is deactivated at the end of the last bus cycle of the instruction following the "LOCK" prefix instruction. While $\overline{\text{LOCK}}$ is active a request on a RQ/GT pin will be recorded and then honored at the end of the LOCK.

EXTERNAL SYNCHRONIZATION VIA TEST

As an alternative to the interrupts and general I/O capabilities, the 8086 provides a single software-testable input known as the $\overline{\text{TEST}}$ signal. At any time the program may execute a WAIT instruction. If at that time the $\overline{\text{TEST}}$ signal is inactive (HIGH), program execution becomes suspended while the processor waits for $\overline{\text{TEST}}$ to become active. It must remain active for at least 5 CLK cycles. The WAIT instruction is re-executed repeatedly until that time. This activity does not consume bus cycles. The processor remains in an idle state while waiting. All 8086 drivers go to 3-state OFF if bus "Hold"is entered. If interrupts are enabled, they may occur while the processor is waiting. When this occurs the processor fetches the WAIT instruction one extra time, processes the interrupt, and then re-fetches and re-executes the WAIT instruction upon returning from the interrupt.

BASIC SYSTEM TIMING

Typical system configurations for the processor operating in minimum mode and in maximum mode are shown in Figures 4a and 4b, respectively. In minimum mode, the MN/$\overline{\text{MX}}$ pin is strapped to V_{CC} and the processor emits bus control signals in a manner similar to the 8085. In maximum mode, the MN/$\overline{\text{MX}}$ pin is strapped to V_{SS} and the processor emits coded status information which the 8288 bus controller uses to generate MULTIBUS compatible bus control signals. Figure 5 illustrates the signal timing relationships.

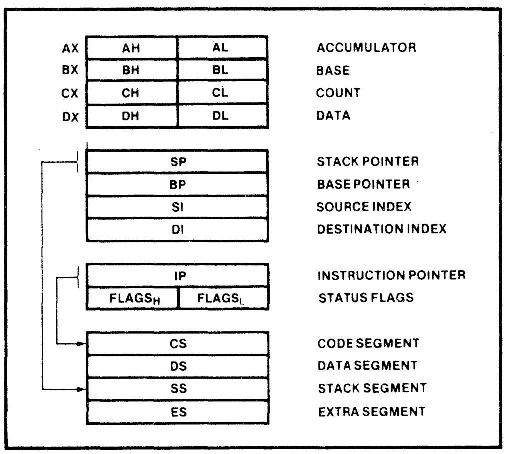


Figure 7. 8086 Register Model

SYSTEM TIMING — MINIMUM SYSTEM

The read cycle begins in T_1 with the assertion of the Address Latch Enable (ALE) signal. The trailing (low-going) edge of this signal is used to latch the address information, which is valid on the local bus at this time, into the 8282/8283 latch. The $\overline{\text{BHE}}$ and A_0 signals address the low, high, or both bytes. From T_1 to T_4 the M/$\overline{\text{IO}}$ signal indicates a memory or I/O operation. At T_2 the address is removed from the local bus and the bus goes to a high impedance state. The read control signal is also asserted at T_2. The read ($\overline{\text{RD}}$) signal causes the addressed device to enable its data bus drivers to the local bus. Some time later valid data will be available on the bus and the addressed device will drive the READY line HIGH. When the processor returns the read signal to a HIGH level, the addressed device will again 3-state its bus drivers. If a transceiver (8286/8287) is required to buffer the 8086 local bus, signals DT/$\overline{R}$ and $\overline{DEN}$ are provided by the 8086.

A write cycle also begins with the assertion of ALE and the emission of the address. The M/$\overline{IO}$ signal is again asserted to indicate a memory or I/O write operation. In the T_2 immediately following the address emission the processor emits the data to be written into the addressed location. This data remains valid until the middle of T_4. During T_2, T_3, and T_W the processor asserts the write control signal. The write ($\overline{WR}$) signal becomes active at the beginning of T_2 as opposed to the read which is delayed somewhat into T_2 to provide time for the bus to float.

The $\overline{BHE}$ and A_0 signals are used to select the proper byte(s) of the memory/IO word to be read or written according to the following table:

$\overline{BHE}$	A0	CHARACTERISTICS
0	0	Whole word
0	1	Upper byte from/ to odd address
1	0	Lower byte from/ to even address
1	1	None

I/O ports are addressed in the same manner as memory location. Even addressed bytes are transferred on the D_7–D_0 bus lines and odd addressed bytes on D_{15}–D_8.

The basic difference between the interrupt acknowledge cycle and a read cycle is that the interrupt acknowledge signal ($\overline{INTA}$) is asserted in place of the read ($\overline{RD}$) signal and the address bus is floated. (See Figure 6.) In the second of two successive INTA cycles, a byte of information is read from bus lines D_7–D_0 as supplied by the interrupt system logic (i.e., 8259A Priority Interrupt Controller). This byte identifies the source (type) of the interrupt. It is multiplied by four and used as a pointer into an interrupt vector lookup table, as described earlier.

BUS TIMING—MEDIUM SIZE SYSTEMS

For medium size systems the MN/$\overline{MX}$ pin is connected to V_{SS} and the 8288 Bus Controller is added to the system as well as an 8282/8283 latch for latching the system address, and a 8286/8287 transceiver to allow for bus loading greater than the 8086 is capable of handling. Signals ALE, DEN, and DT/$\overline{R}$ are generated by the 8288 instead of the processor in this configuration although their timing remains relatively the same. The 8086 status outputs ($\overline{S}_2$, $\overline{S}_1$, and $\overline{S}_0$) provide type-of-cycle information and become 8288 inputs. This bus cycle information specifies read (code, data, or I/O), write (data or I/O), interrupt acknowledge, or software halt. The 8288 thus issues control signals specifying memory read or write, I/O read or write, or interrupt acknowledge. The 8288 provides two types of write strobes, normal and advanced, to be applied as required. The normal write strobes have data valid at the leading edge of write. The advanced write strobes have the same timing as read strobes, and hence data isn't valid at the leading edge of write. The 8286/8287 transceiver receives the usual T and OE inputs from the 8288's DT/$\overline{R}$ and DEN.

The pointer into the interrupt vector table, which is passed during the second INTA cycle, can derive from an 8259A located on either the local bus or the system bus. If the master 8259A Priority Interrupt Controller is positioned on the local bus, a TTL gate is required to disable the 8286/8287 transceiver when reading from the master 8259A during the interrupt acknowledge sequence and software "poll".

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . 0 °C to 70 °C
Storage Temperature . . . . – 65 °C to + 150 °C
Voltage on Any Pin with
Respect to Ground . . . . – 1.0 to + 7V
Power Dissipation . . . . 2.5 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

(8086: T_A = 0°C to 70°C, V_{CC} = 5V ± 10%)
(8086-1: T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)
(8086-2: T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	– 0.5	+ 0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL}=2.5 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = – 400 μA
I_{CC}	Power Supply Current: 8086 8086-1 8086-2		340 360 350	mA	T_A = 25 °C
I_{LI}	Input Leakage Current		± 10	μA	0V ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current		± 10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}
V_{CL}	Clock Input Low Voltage	– 0.5	+ 0.6	V	
V_{CH}	Clock Input High Voltage	3.9	V_{CC} + 1.0	V	
C_{IN}	Capacitance of Input Buffer (All input except $AD_0 - AD_{15}$, $\overline{RQ}/\overline{GT}$)		15	pF	fc = 1 MHz
C_{IO}	Capacitance of I/O Buffer ($AD_0 - AD_{15}$, $\overline{RQ}/\overline{GT}$)		15	pF	fc = 1 MHz

Note: 1. V_{IL} tested with MN/$\overline{MX}$ Pin = 0V.
2. V_{IH} tested with MN/$\overline{MX}$ Pin = 5V.
MN/$\overline{MX}$ Pin is a Strap Pin.

A.C. CHARACTERISTICS (8086: T_A = 0°C to 70°C V_{CC} = 5V ± 10%)
(8086-1: T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)
(8086-2: T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)

MINIMUM COMPLEXITY SYSTEM TIMING REQUIREMENTS

Symbol	Parameter	8086		8086-1		8086-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TCLCL	CLK Cycle Period	200	500	100	500	125	500	ns	
TCLCH	CLK Low Time	118		53		68		ns	
TCHCL	CLK High Time	69		39		44		ns	
TCH1CH2	CLK Rise Time		10		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		5		20		ns	
TCLDX	Data in Hold Time	10		10		10		ns	
TR1VCL	RDY Setup Time into 8284A (See Notes 1, 2)	35		35		35		ns	
TCLR1X	RDY Hold Time into 8284A (See Notes 1, 2)	0		0		0		ns	
TRYHCH	READY Setup Time into 8086	118		53		68		ns	
TCHRYX	READY Hold Time into 8086	30		20		20		ns	
TRYLCL	READY Inactive to CLK (See Note 3)	−8		−10		−8		ns	
THVCH	HOLD Setup Time	35		20		20		ns	
TINVCH	INTR, NMI, $\overline{\text{TEST}}$ Setup Time (See Note 2)	30		15		15		ns	
TILIH	Input Rise Time (Except CLK)		20		20		20	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		12		12		12	ns	From 2.0V to 0.8V

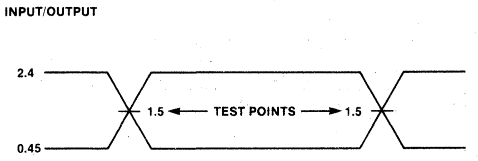
A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES

Symbol	Parameter	8086		8086-1		8086-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TCLAV	Address Valid Delay	10	110	10	50	10	60	ns	
TCLAX	Address Hold Time	10		10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	10	40	TCLAX	50	ns	
TLHLL	ALE Width	TCLCH−20		TCLCH−10		TCLCH-10		ns	
TCLLH	ALE Active Delay		80		40		50	ns	
TCHLL	ALE Inactive Delay		85		45		55	ns	
TLLAX	Address Hold Time to ALE Inactive	TCHCL−10		TCHCL−10		TCHCL−10		ns	
TCLDV	Data Valid Delay	10	110	10	50	10	60	ns	*C_L = 20-100 pF for all 8086 Outputs (In addition to 8086 self-load)
TCHDX	Data Hold Time	10		10		10		ns	
TWHDX	Data Hold Time After WR	TCLCH−30		TCLCH−25		TCLCH−30		ns	
TCVCTV	Control Active Delay 1	10	110	10	50	10	70	ns	
TCHCTV	Control Active Delay 2	10	110	10	45	10	60	ns	
TCVCTX	Control Inactive Delay	10	110	10	50	10	70	ns	
TAZRL	Address Float to READ Active	0		0		0		ns	
TCLRL	$\overline{RD}$ Active Delay	10	165	10	70	10	100	ns	
TCLRH	$\overline{RD}$ Inactive Delay	10	150	10	60	10	80	ns	
TRHAV	$\overline{RD}$ Inactive to Next Address Active	TCLCL−45		TCLCL−35		TCLCL−40		ns	
TCLHAV	HLDA Valid Delay	10	160	10	60	10	100	ns	
TRLRH	$\overline{RD}$ Width	2TCLCL−75		2TCLCL−40		2TCLCL−50		ns	
TWLWH	$\overline{WR}$ Width	2TCLCL−60		2TCLCL−35		2TCLCL−40		ns	
TAVAL	Address Valid to ALE Low	TCLCH−60		TCLCH−35		TCLCH−40		ns	
TOLOH	Output Rise Time		20		20		20	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		12		12		12	ns	From 2.0V to 0.8V

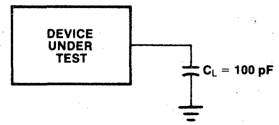
NOTES:
1. Signal at 8284A shown for reference only.
2. Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
3. Applies only to T2 state. (8 ns into T3).

A.C. TESTING INPUT, OUTPUT WAVEFORM



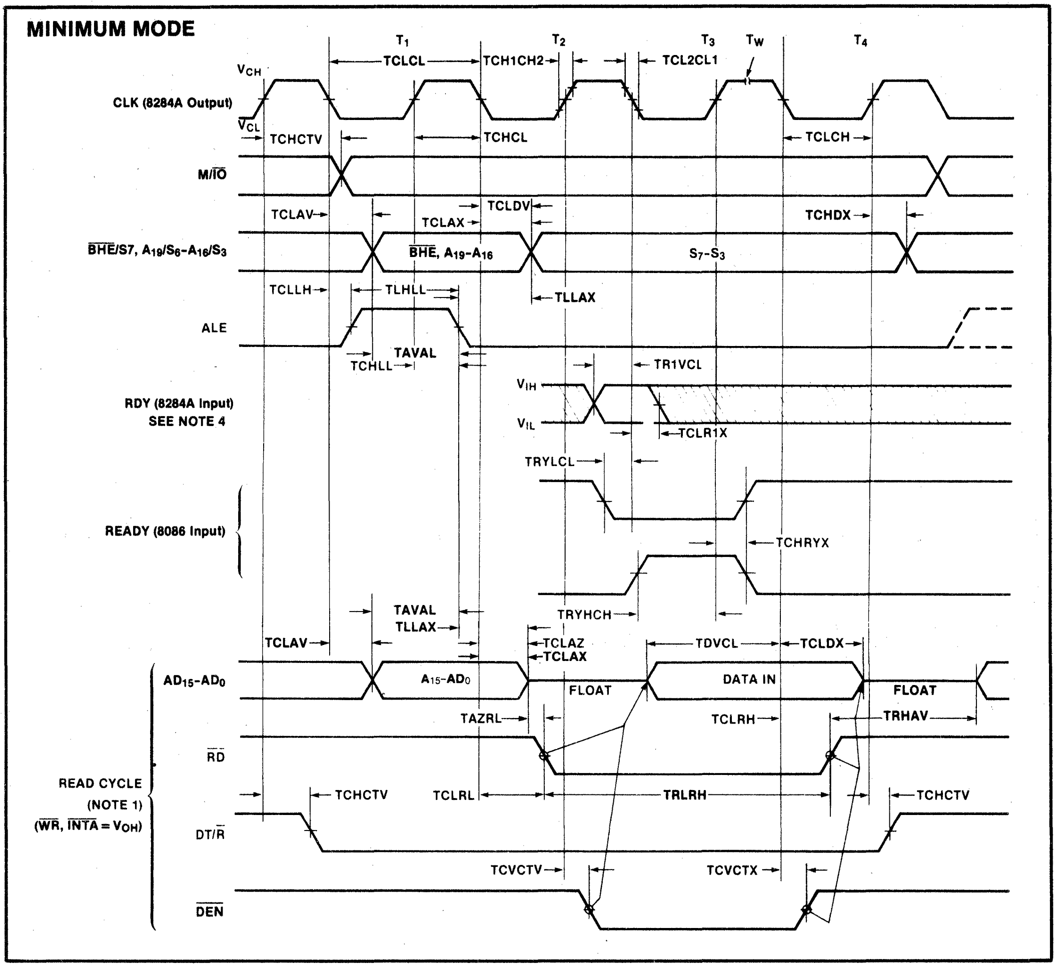
A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 1.5V FOR BOTH A LOGIC "1" AND "0."

A.C. TESTING LOAD CIRCUIT

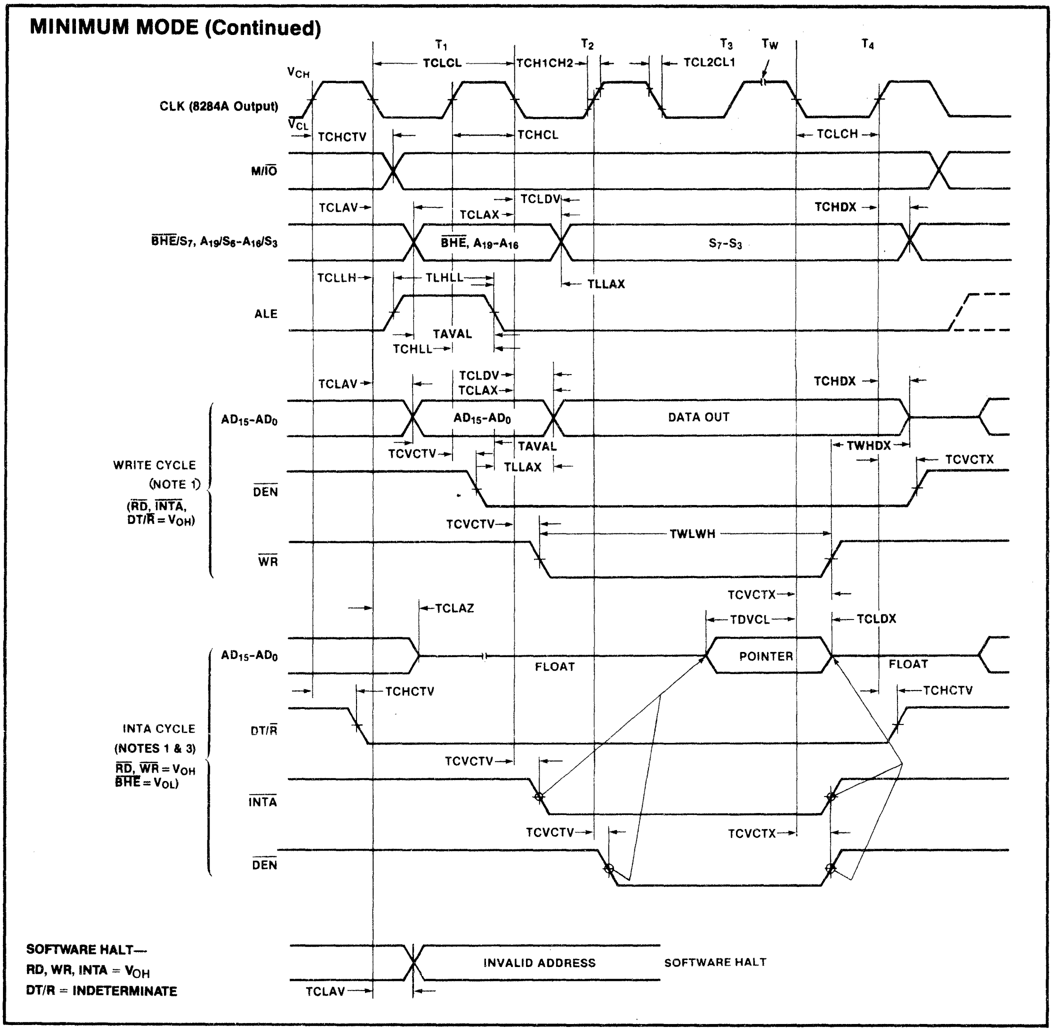


C_L INCLUDES JIG CAPACITANCE

WAVEFORMS



WAVEFORMS (Continued)



NOTES:

1. All signals switch between V_{OH} and V_{OL} unless otherwise specified.
2. RDY is sampled near the end of T_2, T_3, T_W to determine if T_W machines states are to be inserted.
3. Two INTA cycles run back-to-back. The 8086 LOCAL ADDR/DATA BUS is floating during both INTA cycles. Control signals shown for second INTA cycle.
4. Signals at 8284A are shown for reference only.
5. All timing measurements are made at 1.5V unless otherwise noted.

A.C. CHARACTERISTICS

MAX MODE SYSTEM (USING 8288 BUS CONTROLLER)
TIMING REQUIREMENTS

Symbol	Parameter	8086		8086-1		8086-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TCLCL	CLK Cycle Period	200	500	100	500	125	500	ns	
TCLCH	CLK Low Time	118		53		68		ns	
TCHCL	CLK High Time	69		39		44		ns	
TCH1CH2	CLK Rise Time		10		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		5		20		ns	
TCLDX	Data In Hold Time	10		10		10		ns	
TR1VCL	RDY Setup Time into 8284A (See Notes 1, 2)	35		35		35		ns	
TCLR1X	RDY Hold Time into 8284A (See Notes 1, 2)	0		0		0		ns	
TRYHCH	READY Setup Time into 8086	118		53		68		ns	
TCHRYX	READY Hold Time into 8086	30		20		20		ns	
TRYLCL	READY Inactive to CLK (See Note 4)	−8		−10		−8		ns	
TINVCH	Setup Time for Recognition (INTR, NMI, $\overline{TEST}$) (See Note 2)	30		15		15		ns	
TGVCH	$\overline{RQ}/\overline{GT}$ Setup Time (See Note 5)	30		15		15		ns	
TCHGX	$\overline{RQ}$ Hold Time into 8086	40		20		30		ns	
TILIH	Input Rise Time (Except CLK)		20		20		20	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		12		12		12	ns	From 2.0V to 0.8V

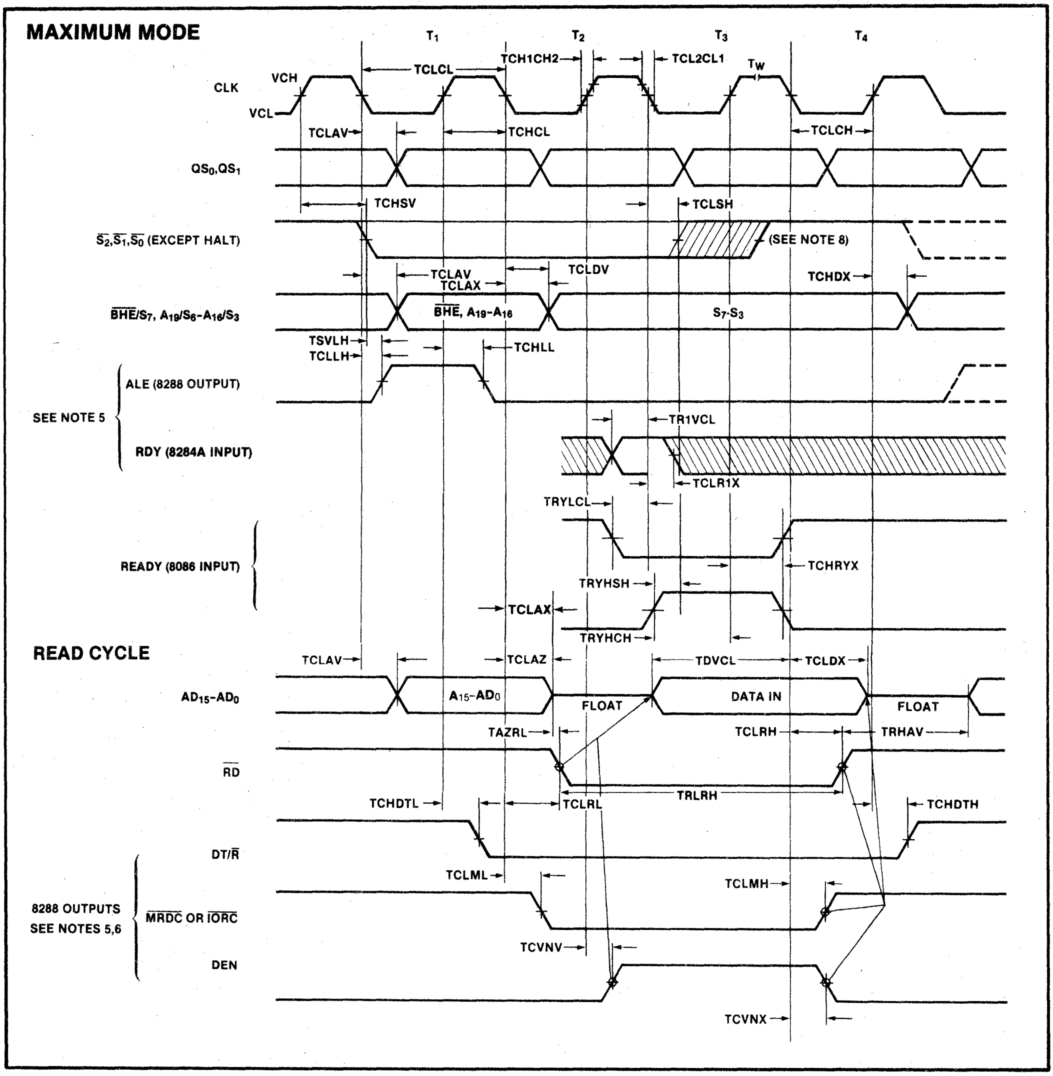
NOTES:
1. Signal at 8284A or 8288 shown for reference only.
2. Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
3. Applies only to T3 and wait states.
4. Applies only to T2 state (8 ns into T3).
5. Change from 1985 Handbook.

A.C. CHARACTERISTICS (Continued)

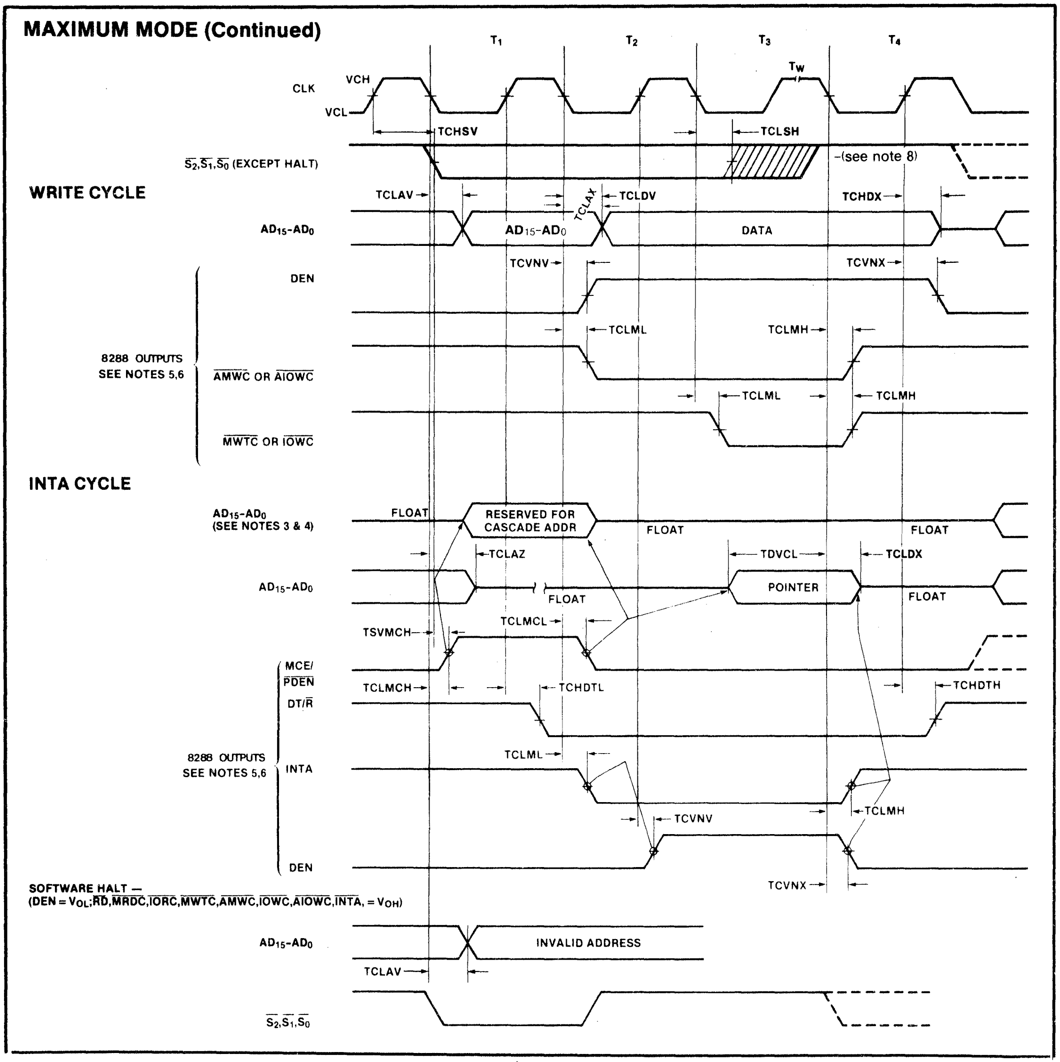
TIMING RESPONSES

Symbol	Parameter	8086		8086-1		8086-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TCLML	Command Active Delay (See Note 1)	10	35	10	35	10	35	ns	
TCLMH	Command Inactive Delay (See Note 1)	10	35	10	35	10	35	ns	
TRYHSH	READY Active to Status Passive (See Note 3)		110		45		65	ns	
TCHSV	Status Active Delay	10	110	10	45	10	60	ns	
TCLSH	Status Inactive Delay	10	130	10	55	10	70	ns	
TCLAV	Address Valid Delay	10	110	10	50	10	60	ns	
TCLAX	Address Hold Time	10		10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	10	40	TCLAX	50	ns	
TSVLH	Status Valid to ALE High (See Note 1)		15		15		15	ns	
TSVMCH	Status Valid to MCE High (See Note 1)		15		15		15	ns	
TCLLH	CLK Low to ALE Valid (See Note 1)		15		15		15	ns	
TCLMCH	CLK Low to MCE High (See Note 1)		15		15		15	ns	
TCHLL	ALE Inactive Delay (See Note 1)		15		15		15	ns	C_L = 20-100 pF for all 8086 Outputs (In addition to 8086 self-load)
TCLMCL	MCE Inactive Delay (See Note 1)		15		15		15	ns	
TCLDV	Data Valid Delay	10	110	10	50	10	60	ns	
TCHDX	Data Hold Time	10		10		10		ns	
TCVNV	Control Active Delay (See Note 1)	5	45	5	45	5	45	ns	
TCVNX	Control Inactive Delay (See Note 1)	10	45	10	45	10	45	ns	
TAZRL	Address Float to Read Active	0		0		0		ns	
TCLRL	RD Active Delay	10	165	10	70	10	100	ns	
TCLRH	RD Inactive Delay	10	150	10	60	10	80	ns	
TRHAV	RD Inactive to Next Address Active	TCLCL−45		TCLCL−35		TCLCL−40		ns	
TCHDTL	Direction Control Active Delay (See Note 1)		50		50		50	ns	
TCHDTH	Direction Control Inactive Delay (See Note 1)		30		30		30	ns	
TCLGL	GT Active Delay (See Note 5)	0	85	0	38	0	50	ns	
TCLGH	GT Inactive Delay	0	85	0	45	0	50	ns	
TRLRH	RD Width	2TCLCL−75		2TCLCL−40		2TCLCL−50		ns	
TOLOH	Output Rise Time		20		20		20	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		12		12		12	ns	From 2.0V to 0.8V

WAVEFORMS


MAXIMUM MODE
T1
T2
T3
T4
TCLCL
TCH1CH2
TCL2CL1
TW
VCH
VCL
CLK
TCLAV
TCHCL
TCLCH
QS0,QS1
TCHSV
TCLSH
S2,S1,S0 (EXCEPT HALT)
(SEE NOTE 8)
TCLAV
TCLAX
TCLDV
TCHDX
BHE/S7, A19/S6-A16/S3
BHE, A19-A16
S7-S3
TSVLH
TCLLH
TCHLL
ALE (8288 OUTPUT)
SEE NOTE 5
RDY (8284A INPUT)
TR1VCL
TCLR1X
TRYLCL
READY (8086 INPUT)
TCHRYX
TRYHSH
TCLAX
TRYHCH
READ CYCLE
TCLAV
TCLAZ
TDVCL
TCLDX
AD15-AD0
A15-AD0
FLOAT
DATA IN
FLOAT
TAZRL
TCLRH
TRHAV
RD
TCHDTL
TCLRL
TRLRH
TCHDTH
DT/R
TCLML
TCLMH
8288 OUTPUTS
SEE NOTES 5,6
MRDC OR IORC
TCVNV
DEN
TCVNX

WAVEFORMS (Continued)

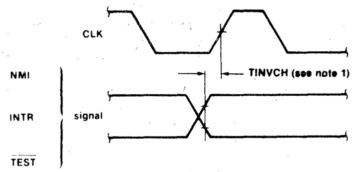


NOTES:

1. All signals switch between V_{OH} and V_{OL} unless otherwise specified.
2. RDY is sampled near the end of T_2, T_3, T_W to determine if T_W machines states are to be inserted.
3. Cascade address is valid between first and second INTA cycle.
4. Two INTA cycles run back-to-back. The 8086 LOCAL ADDR/DATA BUS is floating during both INTA cycles. Control for pointer address is shown for second INTA cycle.
5. Signals at 8284A or 8288 are shown for reference only.
6. The issuance of the 8288 command and control signals (MRDC, MWTC, AMWC, IORC, IOWC, AIOWC, INTA and DEN) lags the active high 8288 CEN.
7. All timing measurements are made at 1.5V unless otherwise noted.
8. Status inactive in state just prior to T_4.

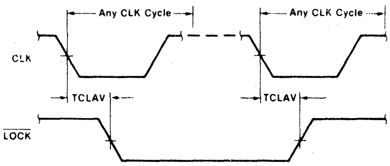
WAVEFORMS (Continued)

ASYNCHRONOUS SIGNAL RECOGNITION

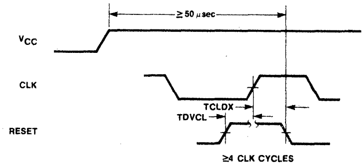


NOTE: 1. SETUP REQUIREMENTS FOR ASYNCHRONOUS SIGNALS ONLY TO GUARANTEE RECOGNITION AT NEXT CLK

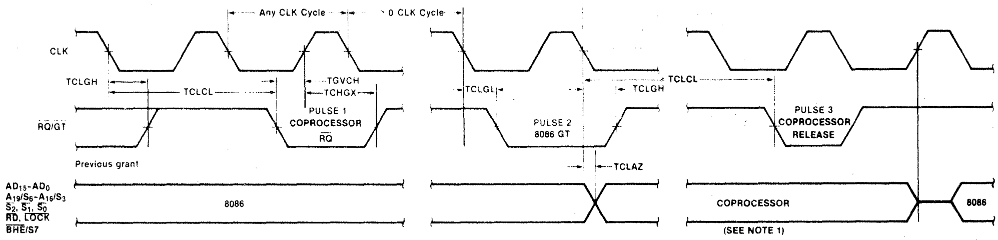
BUS LOCK SIGNAL TIMING (MAXIMUM MODE ONLY)



RESET TIMING



REQUEST/GRANT SEQUENCE TIMING (MAXIMUM MODE ONLY)



NOTES: 1. THE COPROCESSOR MAY NOT DRIVE THE BUSES OUTSIDE THE REGION SHOWN WITHOUT RISKING CONTENTION.

HOLD/HOLD ACKNOWLEDGE TIMING (MINIMUM MODE ONLY)

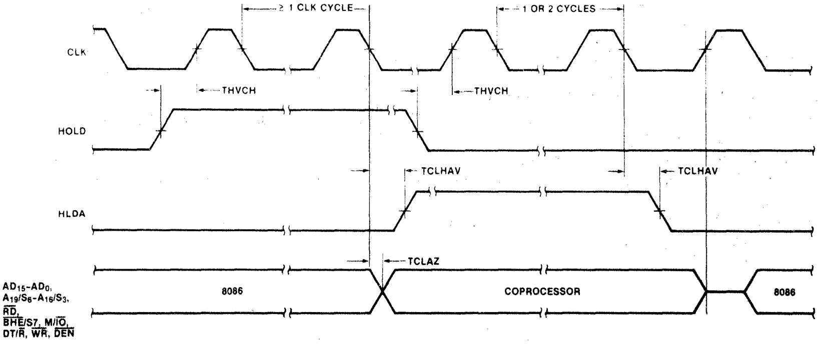


Table 2. Instruction Set Summary

DATA TRANSFER

MOV = **Move:**

	76543210	76543210	76543210	76543210
Register/memory to/from register	1 0 0 0 1 0 d w	mod reg r/m		
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w 1
Immediate to register	1 0 1 1 w reg	data	data if w 1	
Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high	
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high	
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 reg r/m		
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 reg r/m		

PUSH = **Push:**

Register/memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m
Register	0 1 0 1 0 reg	
Segment register	0 0 0 reg 1 1 0	

POP = **Pop:**

Register/memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m
Register	0 1 0 1 1 reg	
Segment register	0 0 0 reg 1 1 1	

XCHG = **Exchange:**

Register/memory with register	1 0 0 0 0 1 1 w	mod reg r/m
Register with accumulator	1 0 0 1 0 reg	

IN = Input from:

Fixed port	1 1 1 0 0 1 0 w	port
Variable port	1 1 1 0 1 1 0 w	

OUT = Output to:

Fixed port	1 1 1 0 0 1 1 w	port
Variable port	1 1 1 0 1 1 1 w	
XLAT = Translate byte to AL	1 1 0 1 0 1 1 1	
LEA = Load EA to register	1 0 0 0 1 1 0 1	mod reg r/m
LDS = Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m
LES = Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m
LAHF = Load AH with flags	1 0 0 1 1 1 1 1	
SAHF = Store AH into flags	1 0 0 1 1 1 1 0	
PUSHF = Push flags	1 0 0 1 1 1 0 0	
POPF = Pop flags	1 0 0 1 1 1 0 1	

ARITHMETIC

ADD = **Add:**

Reg./memory with register to either	0 0 0 0 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s w 01
Immediate to accumulator	0 0 0 0 0 1 0 w	data	data if w 1	

ADC = **Add with carry:**

Reg./memory with register to either	0 0 0 1 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s w 01
Immediate to accumulator	0 0 0 1 0 1 0 w	data	data if w 1	

INC = **Increment:**

Register/memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m
Register	0 1 0 0 0 reg	
AAA = ASCII adjust for add	0 0 1 1 0 1 1 1	
DAA = Decimal adjust for add	0 0 1 0 0 1 1 1	

SUB = **Subtract:**

Reg./memory and register to either	0 0 1 0 1 0 d w	mod reg r/m		
Immediate from register/memory	1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s w 01
Immediate from accumulator	0 0 1 0 1 1 0 w	data	data if w 1	

SBB = **Subtract with borrow**

Reg./memory and register to either	0 0 0 1 1 0 d w	mod reg r/m		
Immediate from register/memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s w 01
Immediate from accumulator	0 0 0 1 1 1 0 w	data	data if w 1	

DEC **Decrement:**

	76543210	76543210	76543210	76543210
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m		
Register	0 1 0 0 1 reg			
NEG Change sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m		

CMP **Compare:**

Register/memory and register	0 0 1 1 1 0 d w	mod reg r/m		
Immediate with register/memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s w 01
Immediate with accumulator	0 0 1 1 1 1 0 w	data	data if w 1	
AAS ASCII adjust for subtract	0 0 1 1 1 1 1 1			
DAS Decimal adjust for subtract	0 0 1 0 1 1 1 1			
MUL Multiply (unsigned)	1 1 1 1 0 1 1 w	mod 1 0 0 r/m		
IMUL Integer multiply (signed)	1 1 1 1 0 1 1 w	mod 1 0 1 r/m		
AAM ASCII adjust for multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0		
DIV Divide (unsigned)	1 1 1 1 0 1 1 w	mod 1 1 0 r/m		
IDIV Integer divide (signed)	1 1 1 1 0 1 1 w	mod 1 1 1 r/m		
AAD ASCII adjust for divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0		
CBW Convert byte to word	1 0 0 1 1 0 0 0			
CWD Convert word to double word	1 0 0 1 1 0 0 1			

LOGIC

NOT Invert	1 1 1 1 0 1 1 w	mod 0 1 0 r/m
SHL/SAL Shift logical/arithmetic left	1 1 0 1 0 0 v w	mod 1 0 0 r/m
SHR Shift logical right	1 1 0 1 0 0 v w	mod 1 0 1 r/m
SAR Shift arithmetic right	1 1 0 1 0 0 v w	mod 1 1 1 r/m
ROL Rotate left	1 1 0 1 0 0 v w	mod 0 0 0 r/m
ROR Rotate right	1 1 0 1 0 0 v w	mod 0 0 1 r/m
RCL Rotate through carry flag left	1 1 0 1 0 0 v w	mod 0 1 0 r/m
RCR Rotate through carry right	1 1 0 1 0 0 v w	mod 0 1 1 r/m

AND **And:**

Reg./memory and register to either	0 0 1 0 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w 1
Immediate to accumulator	0 0 1 0 0 1 0 w	data	data if w 1	

TEST **And function to flags, no result:**

Register/memory and register	1 0 0 0 0 1 0 w	mod reg r/m		
Immediate data and register/memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w 1
Immediate data and accumulator	1 0 1 0 1 0 0 w	data	data if w 1	

OR **Or:**

Reg./memory and register to either	0 0 0 0 1 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w 1
Immediate to accumulator	0 0 0 0 1 1 0 w	data	data if w 1	

XOR **Exclusive or:**

Reg./memory and register to either	0 0 1 1 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w 1
Immediate to accumulator	0 0 1 1 0 1 0 w	data	data if w 1	

STRING MANIPULATION

REP = Repeat	1 1 1 1 0 0 1 z
MOVS = Move byte/word	1 0 1 0 0 1 0 w
CMPS = Compare byte/word	1 0 1 0 0 1 1 w
SCAS = Scan byte/word	1 0 1 0 1 1 1 w
LODS = Load byte/wd to AL/AX	1 0 1 0 1 1 0 w
STOS = Stor byte/wd from AL/A	1 0 1 0 1 0 1 w

Table 2. Instruction Set Summary (Continued)

CONTROL TRANSFER

	76543210	76543210	76543210
CALL = **Call:**			
Direct within segment	1 1 1 0 1 0 0 0	disp-low	disp-high
Indirect within segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m	
Direct intersegment	1 0 0 1 1 0 1 0	offset-low	offset-high
		seg-low	seg-high
Indirect intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m	
JMP = **Unconditional Jump:**			
Direct within segment	1 1 1 0 1 0 0 1	disp-low	disp-high
Direct within segment-short	1 1 1 0 1 0 1 1	disp	
Indirect within segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m	
Direct intersegment	1 1 1 0 1 0 1 0	offset-low	offset-high
		seg-low	seg-high
Indirect intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m	
RET = **Return from CALL:**			
Within segment	1 1 0 0 0 0 1 1		
Within seg. adding immed to SP	1 1 0 0 0 0 1 0	data-low	data-high
Intersegment	1 1 0 0 1 0 1 1		
Intersegment, adding immediate to SP	1 1 0 0 1 0 1 0	data-low	data-high
JE/JZ = Jump on equal/zero	0 1 1 1 0 1 0 0	disp	
JL/JNGE = Jump on less/not greater or equal	0 1 1 1 1 1 0 0	disp	
JLE/JNG = Jump on less or equal/not greater	0 1 1 1 1 1 1 0	disp	
JB/JNAE = Jump on below/not above or equal	0 1 1 1 0 0 1 0	disp	
JBE/JNA = Jump on below or equal/not above	0 1 1 1 0 1 1 0	disp	
JP/JPE = Jump on parity/parity even	0 1 1 1 1 0 1 0	disp	
JO = Jump on overflow	0 1 1 1 0 0 0 0	disp	
JS = Jump on sign	0 1 1 1 1 0 0 0	disp	
JNE/JNZ = Jump on not equal/not zero	0 1 1 1 0 1 0 1	disp	
JNL/JGE = Jump on not less/greater or equal	0 1 1 1 1 1 0 1	disp	
JNLE/JG = Jump on not less or equal/greater	0 1 1 1 1 1 1 1	disp	

	76543210	76543210
JNB/JAE = Jump on not below/above or equal	0 1 1 1 0 0 1 1	disp
JNBE/JA = Jump on not below or equal/above	0 1 1 1 0 1 1 1	disp
JNP/JPO = Jump on not par/par odd	0 1 1 1 1 0 1 1	disp
JNO = Jump on not overflow	0 1 1 1 0 0 0 1	disp
JNS Jump on not sign	0 1 1 1 1 0 0 1	disp
LOOP Loop CX times	1 1 1 0 0 0 1 0	disp
LOOPZ/LOOPE Loop while zero/equal	1 1 1 0 0 0 0 1	disp
LOOPNZ/LOOPNE Loop while not zero/equal	1 1 1 0 0 0 0 0	disp
JCXZ Jump on CX zero	1 1 1 0 0 0 1 1	disp
INT Interrupt		
Type specified	1 1 0 0 1 1 0 1	type
Type 3	1 1 0 0 1 1 0 0	
INTO Interrupt on overflow	1 1 0 0 1 1 1 0	
IRET Interrupt return	1 1 0 0 1 1 1 1	

PROCESSOR CONTROL

	76543210	76543210
CLC Clear carry	1 1 1 1 1 0 0 0	
CMC Complement carry	1 1 1 1 0 1 0 1	
STC Set carry	1 1 1 1 1 0 0 1	
CLD Clear direction	1 1 1 1 1 1 0 0	
STD Set direction	1 1 1 1 1 1 0 1	
CLI Clear interrupt	1 1 1 1 1 0 1 0	
STI Set interrupt	1 1 1 1 1 0 1 1	
HLT Halt	1 1 1 1 0 1 0 0	
WAIT Wait	1 0 0 1 1 0 1 1	
ESC Escape (to external device)	1 1 0 1 1 x x x	mod x x x r/m
LOCK Bus lock prefix	1 1 1 1 0 0 0 0	

Footnotes:

AL = 8-bit accumulator
AX = 16-bit accumulator
CX = Count register
DS = Data segment
ES = Extra segment
Above/below refers to unsigned value.
Greater = more positive;
Less = less positive (more negative) signed values
if d = 1 then "to" reg; if d = 0 then "from" reg
if w = 1 then word instruction; if w = 0 then byte instruction

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP
DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

if s:w = 01 then 16 bits of immediate data form the operand.
if s:w = 11 then an immediate data byte is sign extended to form the 16-bit operand.
if v = 0 then "count" = 1; if v = 1 then "count" in (CL)
x = don't care
z is used for string primitives for comparison with Z.F FLAG.

SEGMENT OVERRIDE PREFIX

0	0	1	reg	1	1	0

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:

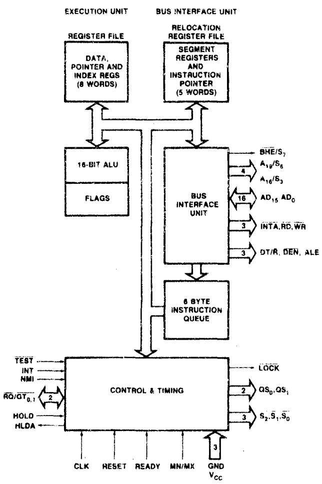
FLAGS = X:X:X:X:(OF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF)

PRELIMINARY

80C86/80C86-2
16-Bit CHMOS Microprocessor

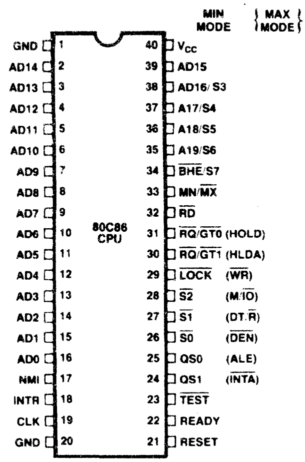
- **Pin-for-Pin and Functionally Compatible to Industry Standard HMOS 8086**
- **Fully Static Design with Frequency Range from D.C. to:**
  - **— 5 MHz for 80C86**
  - **— 8 MHz for 80C86-2**
- **Low Power Operation**
  - **— Operating I_{CC} = 10 mA/MHz**
  - **— Standby I_{CCS} = 500 μA max**
- **Bus-Hold Circuitry Eliminates Pull-Up Resistors**
- **Direct Addressing Capability of 1 MByte of Memory**
- **Architecture Designed for Powerful Assembly Language and Efficient High Level Languages**
- **24 Operand Addressing Modes**
- **Byte, Word and Block Operations**
- **8 and 16-Bit Signed and Unsigned Arithmetic**
  - **— Binary or Decimal**
  - **— Multiply and Divide**
- **Will Be Available in 40-Lead Plastic DIP and 44-Lead PLCC Packages**
  (See Packaging Spec., Order #231369)

The Intel 80C86 is a high performance, CHMOS version of the industry standard HMOS 8086 16-bit CPU. It is available in 5 MHz clock rate and will be available in 8 MHz clock rate in the 1st half of 1986. The 80C86 offers two modes of operation: MINimum for small systems and MAXimum for larger applications such as multi-processing. It is available in 40-pin DIP and will be available in 44-pin plastic leaded chip carrier (PLCC) package in the 1st quarter of 1986.



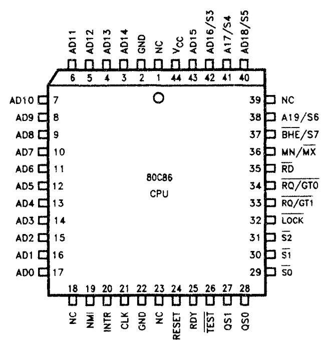
231200-1

Figure 1. 80C86 CPU Block Diagram



231200-2

Figure 2a. 80C86 40-Lead DIP Configuration



231200-3

Figure 2b. 80C86 44-Lead PLCC Configuration

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel. **November 1985**

Order Number: 231200-002

Table 1. Pin Description

The following pin function descriptions are for 80C86 systems in either minimum or maximum mode. The "Local Bus" in these descriptions is the direct multiplexed bus interface connection to the 80C86 (without regard to additional bus buffers).

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>$AD_{15}-AD_0$</td><td>2–16, 39</td><td>I/O</td><td>ADDRESS DATA BUS: These lines constitute the time multiplexed memory/IO address (T_1) and data (T_2, T_3, T_W, T_4) bus. A_0 is analogous to $\overline{BHE}$ for the lower byte of the data bus, pins D_7-D_0. It is LOW during T_1 when a byte is to be transferred on the lower portion of the bus in memory or I/O operations. Eight-bit oriented devices tied to the lower half would normally use A_0 to condition chip select functions. (See $\overline{BHE}$.) These lines are active HIGH and float to 3-state OFF(1) during interrupt acknowledge and local bus "hold acknowledge."</td></tr>
<tr><td>A_{19}/S_6,
A_{18}/S_5,
A_{17}/S_4,
A_{16}/S_3</td><td>35–38</td><td>O</td><td>ADDRESS/STATUS: During T_1 these are the four most significant address lines for memory operations. During I/O operations these lines are LOW. During memory and I/O operations, status information is available on these lines during T_2, T_3, T_W, and T_4. The status of the interrupt enable FLAG bit (S_5) is updated at the beginning of each CLK cycle. A_{17}/S_4 and A_{16}/S_3 are encoded as shown.

This information indicates which relocation register is presently being used for data accessing.

These lines float to 3-state OFF(1) during local bus "hold acknowledge."
<table>
<tr><th>A_{17}/S_4</th><th>A_{16}/S_3</th><th>Characteristics</th></tr>
<tr><td>0 (LOW)</td><td>0</td><td>Alternate Data</td></tr>
<tr><td>0</td><td>1</td><td>Stack</td></tr>
<tr><td>1 (HIGH)</td><td>0</td><td>Code or None</td></tr>
<tr><td>1</td><td>1</td><td>Data</td></tr>
<tr><td>S_6 is 0 (LOW)</td><td></td><td></td></tr>
</table></td></tr>
<tr><td>$\overline{BHE}/S_7$</td><td>34</td><td>O</td><td>BUS HIGH ENABLE/STATUS: During T_1 the bus high enable signal ($\overline{BHE}$) should be used to enable data onto the most significant half of the data bus, pins $D_{15}-D_8$. Eight-bit oriented devices tied to the upper half of the bus would normally use $\overline{BHE}$ to condition chip select functions. $\overline{BHE}$ is LOW during T_1 for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the high portion of the bus. The S_7 status information is available during T_2, T_3, and T_4. The signal is active LOW, and floats to 3-state OFF(1) in "hold." It is LOW during T_1 for the first interrupt acknowledge cycle.
<table>
<tr><th>$\overline{BHE}$</th><th>A_0</th><th>Characteristics</th></tr>
<tr><td>0</td><td>0</td><td>Whole word</td></tr>
<tr><td>0</td><td>1</td><td>Upper byte from/ to odd address</td></tr>
<tr><td>1</td><td>0</td><td>Lower byte from/ to even address</td></tr>
<tr><td>1</td><td>1</td><td>None</td></tr>
</table></td></tr>
</table>

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{RD}$	32	O	**READ:** Read strobe indicates that the processor is performing a memory of I/O read cycle, depending on the state of the S_2 pin. This signal is used to read devices which reside on the 80C86 local bus. $\overline{RD}$ is active LOW during T_2, T_3 and T_W of any read cycle, and is guaranteed to remain HIGH in T_2 until the 80C86 local bus has floated. This floats to 3-state OFF in "hold acknowledge."
READY	22	I	**READY:** is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The READY signal from memory/IO is synchronized by the 82C84A Clock Generator to form READY. This signal is active HIGH. The 80C86 READY input is not synchronized. Correct operation is not guaranteed if the setup and hold times are not met.
INTR	18	I	**INTERRUPT REQUEST:** is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.
$\overline{TEST}$	23	I	**$\overline{TEST}$:** input is examined by the "Wait" instruction. If the $\overline{TEST}$ input is LOW execution continues, otherwise the processor waits in an "Idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.
NMI	17	I	**NON-MASKABLE INTERRUPT:** an edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software. A transition from a LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.
RESET	21	I	**RESET:** causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. It restarts execution, as described in the Instruction Set description, when RESET returns LOW. RESET is internally synchronized.
CLK	19	I	**CLOCK:** provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.
V_{CC}	40		**V_{CC}:** +5V power supply pin.
GND	1, 20		**GROUND:** Both must be connected.
MN/$\overline{MX}$	33	I	**MINIMUM/MAXIMUM:** indicates what mode the processor is to operate in. The two modes are discussed in the following sections.

Table 1. Pin Description (Continued)

The following pin function descriptions are for the 80C86/82C88 system in maximum mode (i.e., $MN/\overline{MX} = V_{SS}$). Only the pin functions which are unique to maximum mode are described; all other pin functions are as described above.

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>$\overline{S_2}, \overline{S_1}, \overline{S_0}$</td><td>26–28</td><td>O</td><td>**STATUS:** active during T_4, T_1, and T_2 and is returned to the passive state (1,1,1) during T_3 or during T_W when READY is HIGH. This status is used by the 82C88 Bus Controller to generate all memory and I/O access control signals. Any change by $\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$ during T_4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T_3 or T_W is used to indicate the end of a bus cycle.

These signals float to 3-state OFF[(1)] in "hold acknowledge." These status lines are encoded as shown.
<table>
<tr><th>$\overline{S_2}$</th><th>$\overline{S_1}$</th><th>$\overline{S_0}$</th><th>Characteristics</th></tr>
<tr><td>0 (LOW)</td><td>0</td><td>0</td><td>Interrupt Acknowledge</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Read I/O Port</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Write I/O Port</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Halt</td></tr>
<tr><td>1 (HIGH)</td><td>0</td><td>0</td><td>Code Access</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Read Memory</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Write Memory</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Passive</td></tr>
</table>
</td></tr>
<tr><td>$\overline{RQ}/\overline{GT}_0$, $\overline{RQ}/\overline{GT}_1$</td><td>30, 31</td><td>I/O</td><td>**REQUEST/GRANT:** pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with $\overline{RQ}/\overline{GT}_0$ having higher priority than $\overline{RQ}/\overline{GT}_1$. $\overline{RQ}/\overline{GT}$ has an internal pull-up resistor so may be left unconnected. The request/grant sequence is as follows (see timing diagram):

1. A pulse of 1 CLK wide from another local bus master indicates a local bus request ("hold") to the 80C86 (pulse 1).
2. During a T_4 or T_1 clock cycle, a pulse 1 CLK wide from the 80C86 to the requesting master (pulse 2), indicates that the 80C86 has allowed the local bus to float and that it will enter the "hold acknowledge" state at the next CLK. The CPU's bus interface unit is disconnected logically from the local bus during "hold acknowledge."
3. A pulse 1 CLK wide from the requesting master indicates to the 80C86 (pulse 3) that the "hold" request is about to end and that 80C86 can reclaim the local bus at the next CLK.

Each master-master exchange of the local bus is a sequence of 3 pulses. There must be one dead CLK cycle after each bus exchange. Pulses are active LOW.

If the request is made while the CPU is performing a memory cycle, it will release the local bus during T_4 of the cycle when all the following conditions are met:

1. Request occurs on or before T_2.
2. Current cycle is not the low byte of a word (on an odd address).
3. Current cycle is not the first acknowledge of an interrupt acknowledge sequence.
4. A locked instruction is not currently executing.</td></tr>
</table>

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
			If the local bus is idle when the request is made the two possible events will follow: 1. Local bus will be released during the next clock. 2. A memory cycle will start within 3 clocks. Now the four rules for a currently active memory cycle apply with condition number 1 already satisfied.
$\overline{LOCK}$	29	O	**$\overline{LOCK}$:** output indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active LOW. The $\overline{LOCK}$ signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW, and floats to 3-state OFF(1) in "hold acknowledge."
QS_1, QS_0	24, 25	O	**QUEUE STATUS:** The queue status is valid during the CLK cycle after which the queue operation is performed. QS_1 and QS_0 provide status to allow external tracking of the internal 80C86 instruction queue.

QS_1	QS_0	Characteristics
0 (LOW)	0	No Operation
0	1	First Byte of Op Code from Queue
1 (HIGH)	0	Empty the Queue
1	1	Subsequent Byte from Queue

The following pin function descriptions are for the 80C86 in minimum mode (i.e., MN/$\overline{MX}$ = V_{CC}). Only the pin functions which are unique to minimum mode are described; all other pin functions are described above.

M/$\overline{IO}$	28	O	**STATUS LINE:** logically equivalent to S_2 in the maximum mode. It is used to distinguish a memory access from an I/O access. M/$\overline{IO}$ becomes valid in the T_4 preceding a bus cycle and remains valid until the final T_4 of the cycle (M = HIGH, IO = LOW). M/$\overline{IO}$ floats to 3-state OFF(1) in local bus "hold acknowledge."
$\overline{WR}$	29	O	**WRITE:** indicates that the processor is performing a write memory or write I/O cycle, depending on the state of the M/$\overline{IO}$ signal. $\overline{WR}$ is active for T_2, T_3 and T_W of any write cycle. It is active LOW, and floats to 3-state OFF(1) in local bus "hold acknowledge."
$\overline{INTA}$	24	O	$\overline{INTA}$ is used as a read strobe for interrupt acknowledge cycles. It is active LOW during T_2, T_3 and T_W of each interrupt acknowledge cycle.
ALE	25	O	**ADDRESS LATCH ENABLE:** provided by the processor to latch the address into an address latch. It is a HIGH pulse active during T_1 of any bus cycle. Note that ALE is never floated.
DT/$\overline{R}$	27	O	**DATA TRANSMIT/RECEIVE:** needed in minimum system that desires to use a data bus transceiver. It is used to control the direction of data flow through the transceiver. Logically DT/$\overline{R}$ is equivalent to $\overline{S_1}$ in the maximum mode, and its timing is the same as for M/$\overline{IO}$. (T = HIGH, R = LOW.) This signal floats to 3-state OFF(1) in local bus "hold acknowledge."

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{DEN}$	26	O	**DATA ENABLE:** provided as an output enable for the transceiver in a minimum system which uses the transceiver. $\overline{DEN}$ is active LOW during each memory and I/O access and for INTA cycles. For a read or $\overline{INTA}$ cycle it is active from the middle of T_2 until the middle of T_4, while for a write cycle it is active from the beginning of T_2 until the middle of T_4. $\overline{DEN}$ floats to 3-state OFF(1) in local bus "hold acknowledge."
HOLD, HLDA	31, 30	I/O	**HOLD:** indicates that another master is requesting a local bus "hold." To be acknowledged, HOLD must be active HIGH. The processor receiving the "hold" request will issue HLDA (HIGH) as an acknowledgement in the middle of a T_1 clock cycle. Simultaneous with the issuance of HLDA the processor will float the local bus and control lines. After HOLD is detected as being LOW, the processor will LOWer the HLDA, and when the processor needs to run another cycle, it will again drive the local bus and control lines. The same rules as for $\overline{RQ}/\overline{GT}$ apply regarding when the local bus will be released. HOLD is not an asynchronous input. External synchronization should be provided if the system cannot otherwise guarantee the setup time.

NOTE:
1. See the section on Bus Hold Circuitry.

FUNCTIONAL DESCRIPTION

STATIC OPERATION

All 80C86 circuitry is of static design. Internal registers, counters and latches are static and require no refresh as with dynamic circuit design. This eliminates the minimum operating frequency restriction placed on other microprocessors. The CMOS 80C86 can operate from DC to the appropriate upper frequency limit. The processor clock may be stopped in either state (high/low) and held there indefinitely. This type of operation is especially useful for system debug or power critical applications.

The 80C86 can be single stepped using only the CPU clock. This state can be maintained as long as is necessary. Single step clock operation allows simple interface circuitry to provide critical information for bringing up your system.

Static design also allows very low frequency operation (down to DC). In a power critical situation, this can provide extremely low power operation since 80C86 power dissipation is directly related to operating frequency. As the system frequency is reduced, so is the operating power until, ultimately, at a DC input frequency, the 80C86 power requirement is the standby current (500 μA maximum).

INTERNAL ARCHITECTURE

The internal functions of the 80C86 processor are partitioned logically into two processing units. The first is the Bus Interface Unit (BIU) and the second is the Execution Unit (EU) as shown in the block diagram of Figure 1.

These units can interact directly but for the most part perform as separate asynchronous operational processors. The bus interface unit provides the functions related to instruction fetching and queuing, operand fetch and store, and address relocation. This unit also provides the basic bus control. The overlap of instruction pre-fetching provided by this unit serves to increase processor performance through improved bus bandwidth utilization. Up to 6 bytes of the instruction stream can be queued while waiting for decoding and execution.

The instruction stream queuing mechanism allows the BIU to keep the memory utilized very efficiently. Whenever there is space for at least 2 bytes in the queue, the BIU will attempt a word fetch memory cycle. This greatly reduces "dead time" on the memory bus. The queue acts as a First-In-First Out (FIFO) buffer, from which the EU extracts instruction bytes as required. If the queue is empty (following a branch instruction, for example), the first byte into the queue immediately becomes available to the EU.

Memory Reference Need	Segment Register Used	Segment Selection Rule
Instructions	CODE (CS)	Automatic with all instruction prefetch.
Stack	STACK (SS)	All stack pushes and pops. Memory references relative to BP base register except data references.
Local Data	DATA (DS)	Data references when: relative to stack, destination of string operation, or explicitly overridden.
External (Global) Data	EXTRA (ES)	Destination of string operations: Explicitly selected using a segment override.

The execution units receives pre-fetched instructions from the BIU queue and provides un-relocated operand addresses to the BIU. Memory operands are passed through the BIU for processing by the EU, which passes results to the BIU for storage. See the Instruction Set description for further register set and architectural descriptions.

MEMORY ORGANIZATION

The processor provides a 20-bit address to memory which locates the byte being referenced. The memory is organized as a linear array of up to 1 million bytes, addressed as 00000(H) to FFFFF(H). The memory is logically divided into code, data, extra data, and stack segments of up to 64k bytes each, with each segment falling on 16-byte boundaries. (See Figure 3a.)

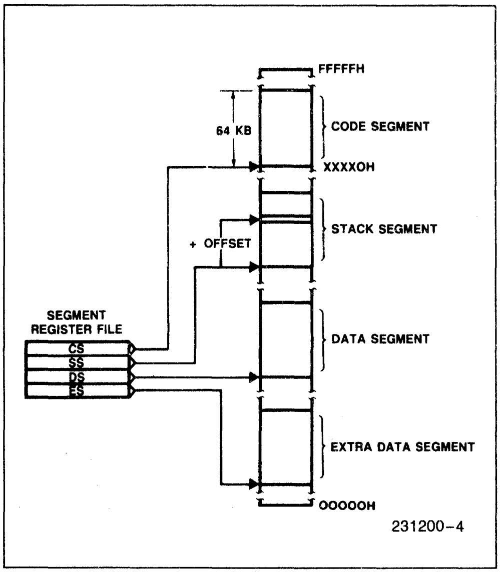


Figure 3a. Memory Organization

All memory references are made relative to base addresses contained in high speed segment registers. The segment types were chosen based on the addressing needs of programs. The segment register to be selected is automatically chosen according to the rules of the following table. All information in one segment type share the same logical attributes (e.g. code or data). By structuring memory into relocatable areas of similar characteristics and by automatically selecting segment registers, programs are shorter, faster, and more structured.

Word (16-bit) operands can be located on even or odd address boundaries and are thus not constrained to even boundaries as is the case in many 16-bit computers. For address and data operands, the least significant byte of the word is stored in the lower valued address location and the most significant byte in the next higher address location. The BIU automatically performs the proper number of memory accesses, one if the word operand is on an even byte boundary and two if it is on an odd byte boundary. Except for the performance penalty, this double access is transparent to the software. This performance penalty does not occur for instruction fetches, only word operands.

Physically, the memory is organized as a high bank ($D_{15}-D_8$) and a low bank (D_7-D_0) of 512k 8-bit bytes addressed in parallel by the processor's address lines.

$A_{19}-A_1$. Byte data with even addresses is transferred on the D_7-D_0 bus lines while odd addressed byte data (A_0 HIGH) is transferred on the $D_{15}-D_8$ bus lines. The processor provides two enable signals, $\overline{BHE}$ and A_0, to selectively allow reading from or writing into either an odd byte location, even byte location, or both. The instruction stream is fetched from memory as words and is addressed internally by the processor to the byte level as necessary.

In referencing word data the BIU requires one or two memory cycles depending on whether or not the starting byte of the word is on an even or odd address, respectively. Consequently, in referencing word operands performance can be optimized by locating data on even address boundaries. This is an especially useful technique for using the stack, since odd address references to the stack may adversely affect the context switching time for interrupt processing or task multiplexing.

Certain locations in memory are reserved for specific CPU operations (see Figure 3b.) Locations from address FFFF0H through FFFFFH are reserved for operations including a jump to the initial program loading routine. Following RESET, the CPU will always begin execution at location FFFF0H where the jump must be. Locations 00000H through 003FFH are reserved for interrupt operations. Each of the 256 possible interrupt types has its service routine pointed to by a 4-byte pointer element consisting of a 16-bit segment address and a 16-bit offset address. The pointer elements are assumed to have been stored at the respective places in reserved memory prior to occurrence of interrupts.

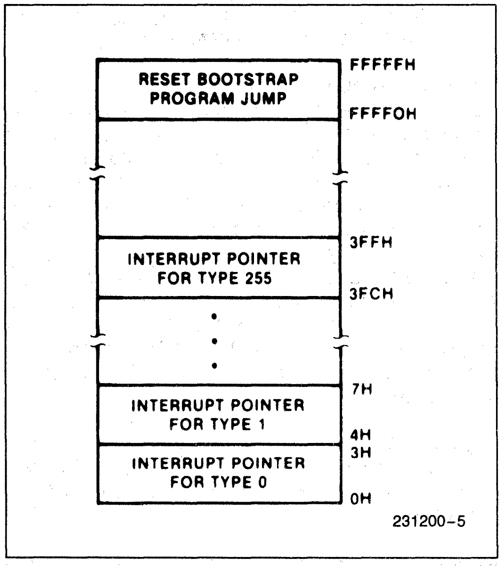


Figure 3b. Reserved Memory Locations

MINIMUM AND MAXIMUM MODES

The requirements for supporting minimum and maximum 80C86 systems are sufficiently different that they cannot be done efficiently with 40 uniquely defined pins. Consequently, the 80C86 is equipped with a strap pin (MN/$\overline{MX}$) which defines the system configuration. The definition of a certain subset of the pins changes dependent on the condition of the strap pin. When MN/$\overline{MX}$ pin is strapped to GND, the 80C86 treats pins 24 through 31 in maximum mode. An 82C88 bus controller interprets status information coded into $\overline{S}_0$, $\overline{S}_1$, $\overline{S}_2$ to generate bus timing and control signals compatible with the MULTIBUS® architecture. When the MN/$\overline{MX}$ pin is strapped to V_{CC}, the 80C86 generates bus control signals itself on pins 24 through 31, as shown in parentheses in Figure 2. Examples of minimum mode and maximum mode systems are shown in Figure 4.

BUS OPERATION

The 80C86 has a combined address and data bus commonly referred to as a time multiplexed bus. This technique provides the most efficient use of pins on the processor while permitting the use of a standard 40-lead package. This "local bus" can be buffered directly and used throughout the system with address latching provided on memory and I/O modules. In addition, the bus can also be demultiplexed at the processor with a single set of address latches if a standard non-multiplexed bus is desired for the system.

Each processor bus cycle consists of at least four CLK cycles. These are referred to as T_1, T_2, T_3 and T_4 (see Figure 5). The address is emitted from the processor during T_1 and data transfer occurs on the bus during T_3 and T_4. T_2 is used primarily for changing the direction of the bus during read operations. In the event that a "NOT READY" indication is given by the addressed device, "Wait" states (T_W) are inserted between T_3 and T_4. Each inserted "Wait" state is of the same duration as a CLK cycle. Periods can occur between 80C86 bus cycles. These are referred to as "Idle" states (T_1) or inactive CLK cycles. The processor uses these cycles for internal housekeeping.

During T_1 of any bus cycle the ALE (Address Latch Enable) signal is emitted (by either the processor or the 82C88 bus controller, depending on the MN/$\overline{MX}$ strap). At the trailing edge of this pulse, a valid address and certain status information for the cycle may be latched.

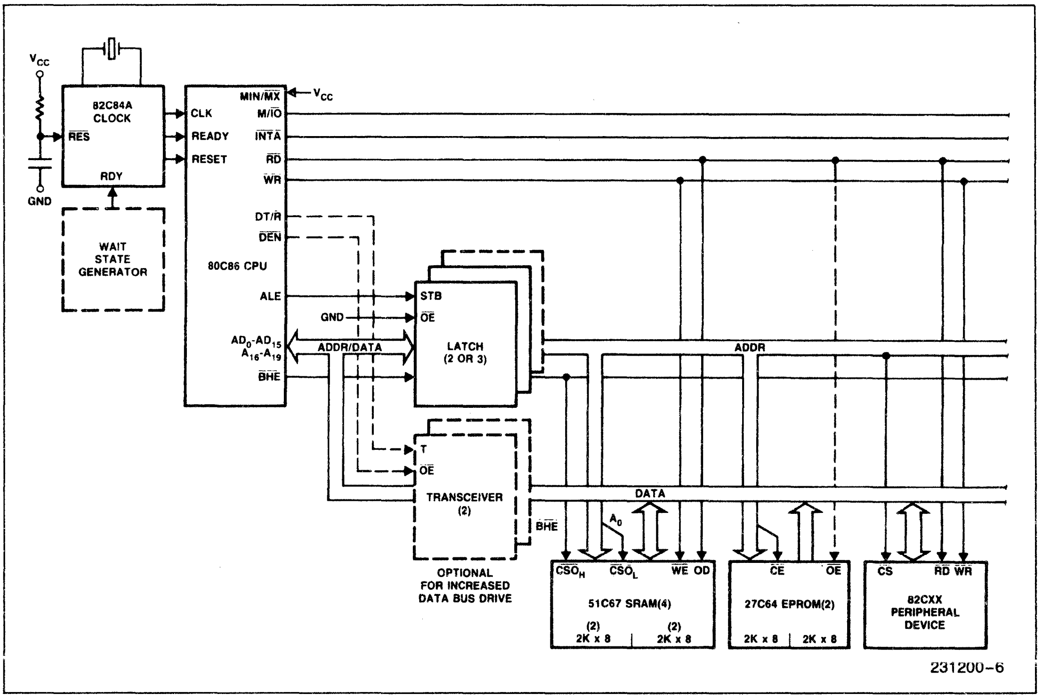


Figure 4a. Minimum Mode iAPX 80C86 Typical Configuration

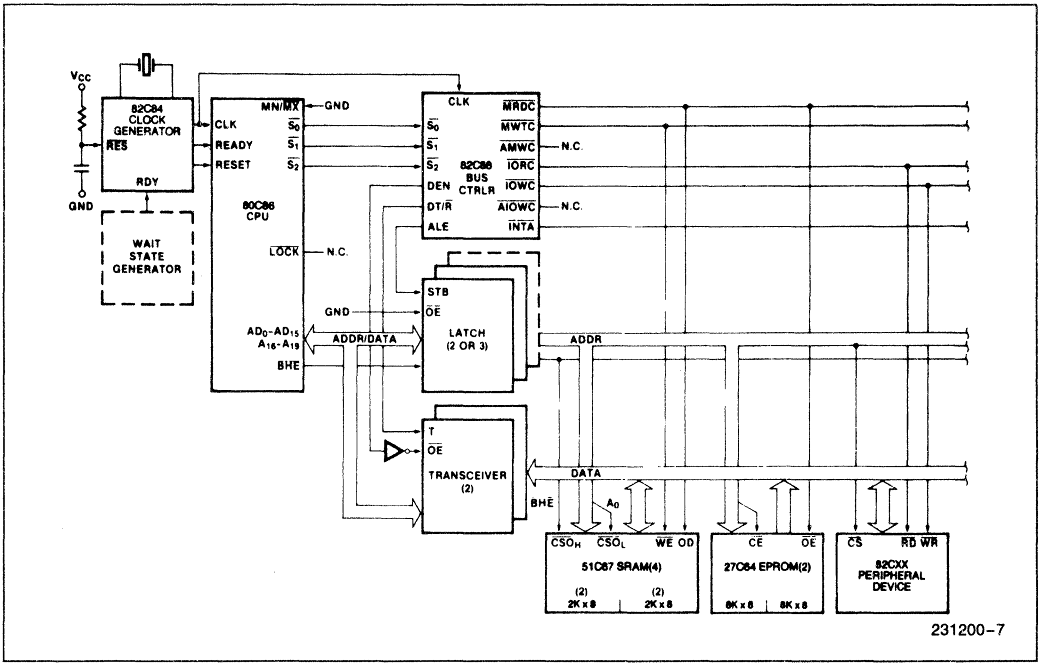


Figure 4b. Maximum Mode 80C86 Typical Configuration

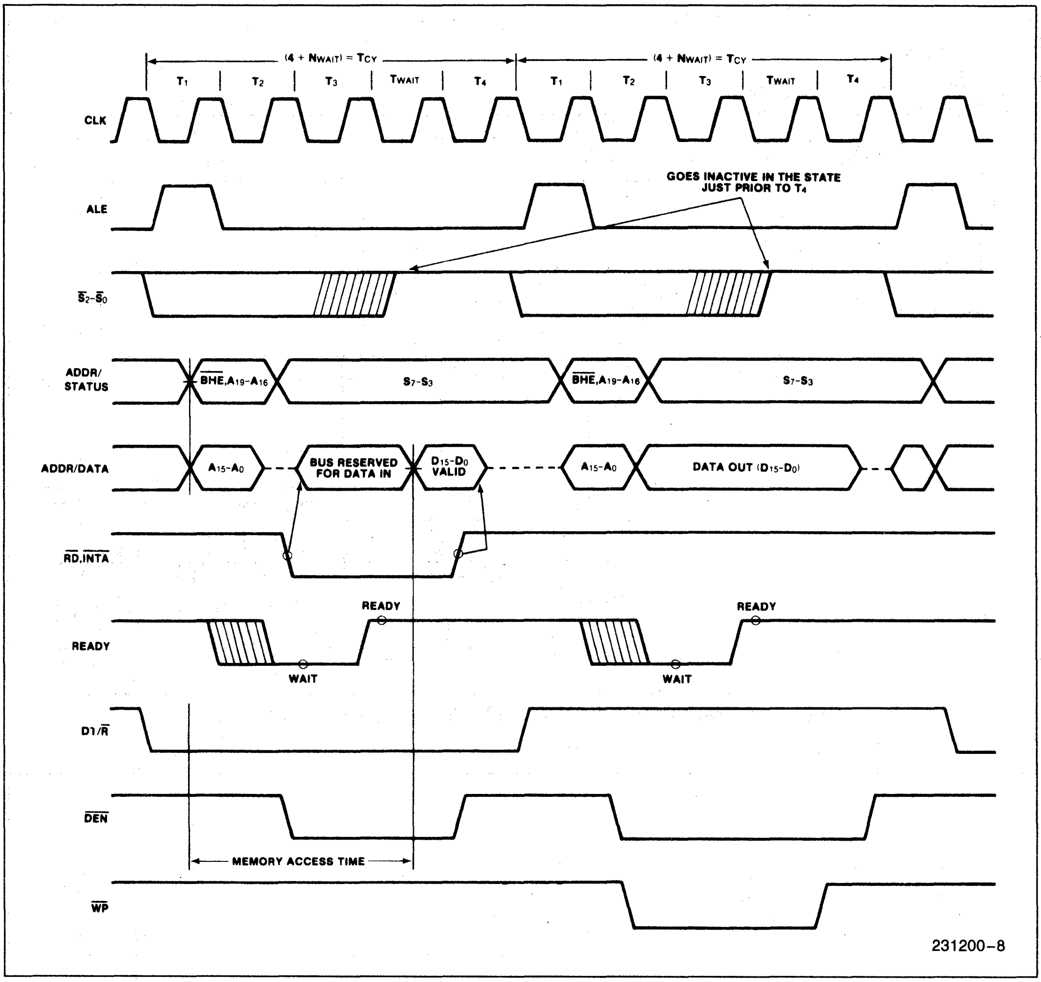


Figure 5. Basic System Timing

Status bits $\overline{S}_0$, $\overline{S}_1$, and $\overline{S}_2$ are used, in maximum mode, by the bus controller to identify the type of bus transaction according to the following table:

$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$	Characteristics
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1 (HIGH)	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

Status bits S_3 through S_7 are multiplexed with high-order address bits and the $\overline{BHE}$ signal, and are therefore valid during T_2 through T_4. S_3 and S_4 indicate which segment register (see Instruction Set description) was used for this bus cycle in forming the address, according to the following table:

S_4	S_3	Characteristics
0 (LOW)	0	Alternate Data (extra segment)
0	1	Stack
1 (HIGH)	0	Code or None
1	1	Data

S_5 is a reflection of the PSW interrupt enable bit. $S_6 = 0$ and S_7 is a spare status bit.

I/O ADDRESSING

In the 80C86, I/O operations can address up to a maximum of 64k I/O byte registers or 32k I/O word registers. The I/O address appears in the same format as the memory address on bus lines $A_{15}-A_0$. The address lines $A_{19}-A_{16}$ are zero in I/O operations. The variable I/O instructions which use register DX as a pointer have full address capability while the direct I/O instructions directly address one or two of the 256 I/O byte locations in page 0 of the I/O address space.

I/O ports are addressed in the same manner as memory locations. Even addressed bytes are transferred on the D_7-D_0 bus lines and odd addressed bytes on $D_{15}-D_8$. Care must be taken to assure that each register within an 8-bit peripheral located on the lower portion of the bus be addressed as even.

EXTERNAL INTERFACE

PROCESSOR RESET AND INITIALIZATION

Processor initialization or start up is accomplished with activation (HIGH) of the RESET pin. The 80C86 RESET is required to be HIGH for greater than 4 CLK cycles. The 80C86 will terminate operations on the high-going edge of RESET and will remain dormant as long as RESET is HIGH. The low-going transition of RESET triggers an internal reset sequence for approximately 10 CLK cycles. After this interval the 80C86 operates normally beginning with the instruction in absolute location FFFF0H (see Figure 3b). The details of this operation are specified in the Instruction Set description of the MCS®-86 Family User's Manual. The RESET input is internally synchronized to the processor clock. At initialization the HIGH-to-LOW transition of RESET must occur no sooner than 50 μs after power-up, to allow complete initialization of the 80C86.

NMI may not be asserted prior to the 2nd CLK cycle following the end of RESET.

BUS HOLD CIRCUITRY

To avoid high current conditions caused by floating inputs to CMOS devices and eliminate the need for pull-up/down resistors, "bus-hold" circuitry has been used on the 80C86 pins 2–16, 26–32, and 34–39 (Figures 6a, 6b). These circuits will maintain the last valid logic state if no driving source is present (i.e. an unconnected pin or a driving source which goes to a high impedance state). To overdrive the "bus hold" circuits, an external driver must be capable of supplying 350 μA minimum sink or source current at valid input voltage levels. Since this "bus hold" circuitry is active and not a "resistive" type element, the associated power supply current is negligible and power dissipation is significantly reduced when compared to the use of passive pull-up resistors.

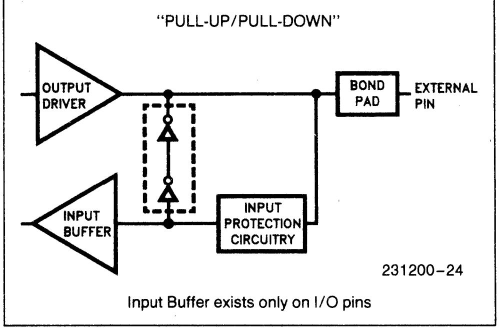


Figure 6a. Bus hold circuitry pin 2-16, 34-39.

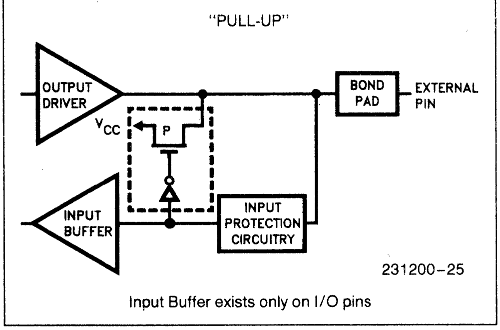


Figure 6b. Bus hold circuitry pin 26-32.

INTERRUPT OPERATIONS

Interrupt operations fall into two classes; software or hardware initiated. The software initiated interrupts and software aspects of hardware interrupts are specified in the Instruction Set description. Hardware interrupts can be classified as non-maskable or maskable.

Interrupts result in a transfer of control to a new program location. A 256-element table containing address pointers to the interrupt service program locations resides in absolute locations 0 through 3FFH (see Figure 3b), which are reserved for this purpose. Each element in the table is 4 bytes in size and corresponds to an interrupt "type". An interrupting device supplies an 8-bit type number, during the interrupt acknowledge sequence, which is used to "vector" through the appropriate element to the new interrupt service program location.

NON-MASKABLE INTERRUPT (NMI)

The processor provides a single non-maskable interrupt pin (NMI) which has higher priority than the maskable interrupt request pin (INTR). A typical use would be to activate a power failure routine. The NMI is edge-triggered on a LOW-to-HIGH transition. The activation of this pin causes a type 2 interrupt. (See Instruction Set description.) NMI is required to have a duration in the HIGH state of greater than two CLK cycles, but is not required to be synchronized to the clock. Any high-going transition of NMI is latched on-chip and will be serviced at the end of the current instruction or between whole moves of a block-type instruction. Worst case response to NMI would be for multiply, divide and variable shift instructions. There is no specification on the occurrence of the low-going edge; it may occur before, during, or after the servicing of NMI. Another high-going edge triggers another response if it occurs after the start of the NMI procedure. The signal must be free of logical spikes in general and be free of bounces on the low-going edge to avoid triggering extraneous responses.

MASKABLE INTERRUPT (INTR)

The 80C86 provides a single interrupt request input (INTR) which can be masked internally by software with the resetting of the interrupt enable FLAG status bit. The interrupt request signal is level triggered. It is internally synchronized during each clock cycle on the high-going edge of CLK. To be responded to, INTR must be present (HIGH) during the clock period preceding the end of the current instruction or the end of a whole move for a block-type instruction. During the interrupt response sequence further interrupts are disabled. The enable bit is reset as part of the response to any interrupt (INTR, NMI, software interrupt or single-step), although the FLAGS register which is automatically pushed onto the stack reflects the state of the processor prior to the interrupt. Until the old FLAGS register is restored the enable bit will be zero unless specifically set by an instruction.

During the response sequence (Figure 7) the processor executes two successive (back-to-back) interrupt acknowledge cycles. The 80C86 emits the LOCK signal from T_2 of the first bus cycle until T_2 of the second. A local bus "hold" request will not be honored until the end of the second bus cycle. In the second bus cycle a byte is fetched from the external interrupt system (e.g., 82C59 PIC) which identifies the source (type) of the interrupt. This byte is multiplied by four and used as a pointer into the interrupt vector lookup table. An INTR signal left HIGH will be continually responded to within the limitations of the enable bit and sample period. The INTERRUPT RETURN instruction includes a FLAGS pop which returns the status of the original interrupt enable bit when it restores the FLAGS.

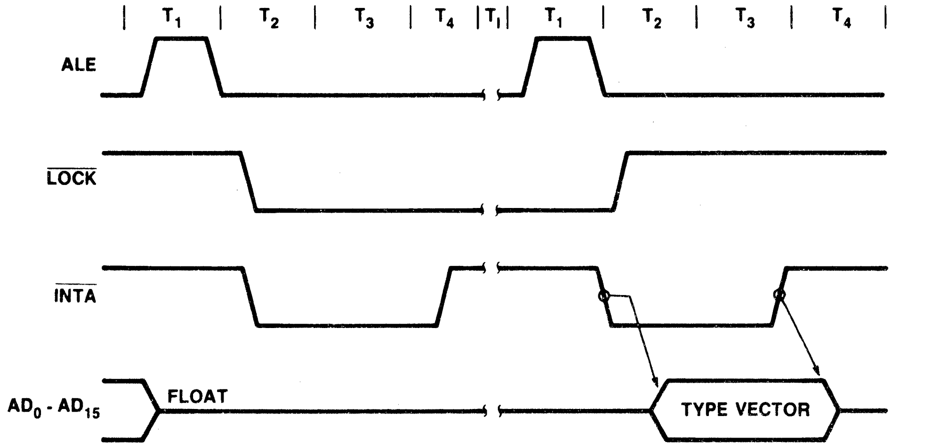


Figure 7. Interrupt Acknowledge Sequence

HALT

When a software "HALT" instruction is executed the processor indicates that it is entering the "HALT" state in one of two ways depending upon which mode is strapped. In minimum mode, the processor issues one ALE with no qualifying bus control signals. In Maximum Mode, the processor issues appropriate HALT status on $\overline{S_2}$, $\overline{S_1}$ and $\overline{S_0}$ and the 82C88 bus controller issues one ALE. The 80C86 will not leave the "HALT" state when a local bus "hold" is entered while in "HALT". In this case, the processor reissues the HALT indicator. An interrupt request or RESET will force the 80C86 out of the "HALT" state.

READ/MODIFY/WRITE (SEMAPHORE) OPERATIONS VIA LOCK

The $\overline{LOCK}$ status information is provided by the processor when directly consecutive bus cycles are required during the execution of an instruction. This provides the processor with the capability of performing read/modify/write operations on memory (via the Exchange Register With Memory instruction, for example) without the possibility of another system bus master receiving intervening memory cycles. This is useful in mutliprocessor system configurations to accomplish "test and set lock" operations. The $\overline{LOCK}$ signal is activated (forced LOW) in the clock cycle following the one in which the software "LOCK" prefix instruction is decoded by the EU. It is deactivated at the end of the last bus cycle of the instruction following the "LOCK" prefix instruction. While $\overline{LOCK}$ is active a request on a RQ/GT pin will be recorded and then honored at the end of the LOCK.

EXTERNAL SYNCHRONIZATION VIA TEST

As an alternative to the interrupts and general I/O capabilities, the 80C86 provides a single software-testable input known as the $\overline{TEST}$ signal. At any time the program may execute a WAIT instruction. If at that time the $\overline{TEST}$ signal is inactive (HIGH), program execution becomes suspended while the processor waits for $\overline{TEST}$ to become active. It must remain active for at least 5 CLK cycles. The WAIT instruction is re-executed repeatedly until that time. This activity does not consume bus cycles. The processor remains in an idle state while waiting. All 80C86 drivers go to 3-state OFF if bus "Hold" is entered. If interrupts are enabled, they may occur while the processor is waiting. When this occurs the processor fetches the WAIT instruction one extra time, processes the interrupt, and then re-fetches and re-executes the WAIT instruction upon returning from the interrupt.

BASIC SYSTEM TIMING

Typical system configurations for the processor operating in minimum mode and in maximum mode are shown in Figures 4a and 4b, respectively. In minimum mode, the MN/$\overline{MX}$ pin is strapped to V_{CC} and the processor emits bus control signals in a manner similar to the 8085. In maximum mode, the MN/$\overline{MX}$ pin is strapped to V_{SS} and the processor emits coded status information which the 82C88 bus controller uses to generate MULTIBUS compatible bus control signals. Figure 5 illustrates the signal timing relationships.

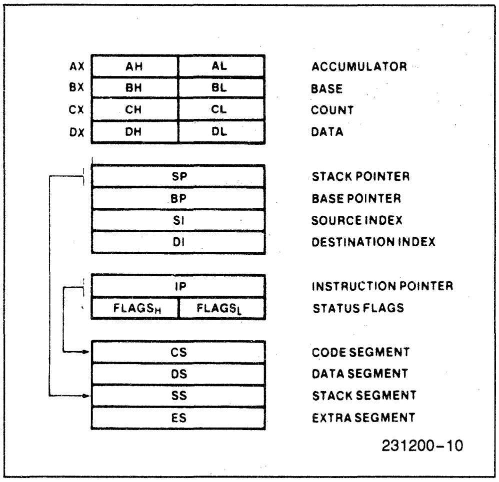


Figure 8. iAPX 80C86 Register Model

SYSTEM TIMING—MINIMUM SYSTEM

The read cycle begins in T_1 with the assertion of the Address Latch Enable (ALE) signal. The trailing (low-going) edge of this signal is used to latch the address information, which is valid on the local bus at this time, into a latch. The $\overline{\text{BHE}}$ and A_0 signals address the low, high, or both bytes. From T_1 to T_4 the M/$\overline{\text{IO}}$ signal indicates a memory or I/O operation. At T_2 the address is removed from the local bus and the bus goes to a high impedance state. The read control signal is also asserted at T_2. The read ($\overline{\text{RD}}$) signal causes the addressed device to enable its data bus drivers to the local bus. Some time later valid data will be available on the bus and the addressed device will drive the READY line HIGH. When the processor returns the read signal to a HIGH level, the addressed device will again 3-state its bus drivers. If a transceiver is required to buffer the 80C86 local bus, signals DT/$\overline{\text{R}}$ and $\overline{\text{DEN}}$ are provided by the 80C86.

A write cycle also begins with the assertion of ALE and the emission of the address. The M/$\overline{\text{IO}}$ signal is again asserted to indicate a memory or I/O write operation. In the T_2 immediately following the address emission the processor emits the data to be written into the addressed location. This data remains valid until the middle of T_4. During T_2, T_3, and T_W the processor asserts the write control signal. The write ($\overline{\text{WR}}$) signal becomes active at the beginning of T_2 as opposed to the read which is delayed somewhat into T_2 to provide time for the bus to float.

The $\overline{\text{BHE}}$ and A_0 signals are used to select the proper byte(s) of the memory/IO word to be read or written according to the following table:

$\overline{\text{BHE}}$	A0	Characteristics
0	0	Whole word
0	1	Upper byte from/ to odd address
1	0	Lower byte from/ to even address
1	1	None

I/O ports are addressed in the same manner as memory location. Even addressed bytes are transferred on the D_7-D_0 bus lines and odd addressed bytes on $D_{15}-D_8$.

The basic difference between the interrupt acknowledge cycle and a read cycle is that the interrupt acknowledge signal ($\overline{\text{INTA}}$) is asserted in place of the read ($\overline{\text{RD}}$) signal and the address bus is floated. (See Figure 7.) In the second of two successive INTA cycles, a byte of information is read from bus lines D_7-D_0 as supplied by the interrupt system logic (i.e., 82C59A Priority Interrupt Controller). This byte identifies the source (type) of the interrupt. It is multiplied by four and used as a pointer into an interrupt vector lookup table, as described earlier.

BUS TIMING—MEDIUM SIZE SYSTEMS

For medium size systems the MN/$\overline{\text{MX}}$ pin is connected to V_{SS} and the 82C88 Bus Controller is added to the system as well as a latch for latching the system address, and a transceiver to allow for bus loading greater than the 80C86 is capable of handling. Signals ALE, DEN, and DT/$\overline{\text{R}}$ are generated by the 82C88 instead of the processor in this configuration although their timing remains relatively the same. The 80C86 status outputs ($\overline{S_2}$, $\overline{S_1}$, and $\overline{S_0}$) provide type-of-cycle information and become 82C88 inputs. This bus cycle information specifies read (code, data, or I/O), write (data or I/O), interrupt acknowledge, or software halt. The 82C88 thus issues control signals specifying memory read or write, I/O read or write, or interrupt acknowledge. The 82C88 provides two types of write strobes, normal and advanced, to be applied as required. The normal write strobes have data valid at the leading edge of write. The advanced write strobes have the same timing as read strobes, and hence data isn't valid at the leading edge of write. The transceiver receives the usual T and OE inputs from the 82C88 DT/$\overline{\text{R}}$ and DEN.

The pointer into the interrupt vector table, which is passed during the second INTA cycle, can derive from an 82C59A located on either the local bus or the system bus. If the master 82C59A Priority Interrupt Controller is positioned on the local bus, a TTL gate is required to disable the transceiver when reading from the master 82C59A during the interrupt acknowledge sequence and software "poll".

ABSOLUTE MAXIMUM RATINGS*

Supply Voltage
(With respect to ground) .... −0.5 to 8.0V

Input Voltage Applied
(w.r.t. ground) .... −2.0 to V_{CC} + 0.5V

Output Voltage Applied
(w.r.t. ground) .... −0.5 to V_{CC} + 0.5V

Power Dissipation.......................... 1.0W

Storage Temperature............. −65°C to 150°C

Ambient Temperature Under Bias ...... 0°C to 70°C

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS (80C86: T_A = 0°C to 70°C, V_{CC} = 5V ±10%) (80C86-2: T_A = 0°C to 70°C, V_{CC} = 5V ±5%)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage (All input except $\overline{RQ}/\overline{GT0}$, $\overline{RQ}/\overline{GT}$, MN/$\overline{MX}$)	2.0	V_{CC} + 0.5	V	(Note 6)
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 2.5 mA
V_{OH}	Output High Voltage	3.0 V_{CC}−0.4		V	I_{OH} = −2.5 mA I_{OH} = −100 μA
I_{CC}	Power Supply Current		10 mA/MHz		V_{IL} = GND, V_{IH} = V_{CC} T_A = 25°C, V_{CC} = 5.5V
I_{CCS}	Standby Supply Current		750	μA	V_{CC} = 5.5V Ready = High V_{IN} (max) = V_{CC} or GND Outputs Unloaded CLK = GND or V_{CC} (Note 7)
I_{CCS}	Standby Supply Current		2.5	mA	V_{CC} = 5.5V Ready = Low V_{IN}(max) = V_{CC} or GND Outputs Unloaded (Note 7) CLK = GND or V_{CC}
I_{LI}	Input Leakage Current		±1.0	μA	0V ≤ V_{IN} ≤ V_{CC}
I_{BHL}	Input Leakage Current (Bus Hold Low)	50	300	μA	V_{IN} = 0.8V (Note 1)
I_{BHH}	Input Leakage Current (Bus Hold High)	−50	−300	μA	V_{IN} = 3.0V (Note 2)
I_{BHLO}	Bus Hold Low Overdrive		350	μA	(Note 4)
I_{BHHO}	Bus Hold High Overdrive		−350	μA	(Note 5)
I_{LO}	Output Leakage Current		±10	μA	0V ≤ V_{OUT} ≤ V_{CC}
V_{CL}	Clock Input Low Voltage	−0.5	+0.8	V	
V_{CH}	Clock Input High Voltage	V_{CC}−0.8	V_{CC}+0.5	V	
C_{IN}	Capacitance of Input Buffer (All input except AD_0–AD_{15}, $\overline{RQ}/\overline{GT}$)		5	pF	(Note 3)
C_{IO}	Capacitance of I/O Buffer (AD_0–AD_{15}, $\overline{RQ}/\overline{GT}$)		20	pF	(Note 3)
C_{OUT}	Output Capacitance		15	pF	(Note 3)

NOTES:
1. Test condition is to lower V_{IN} to GND and then raise V_{IN} to 0.8V on pins 2–16 and 34–39.
2. Test condition is to raise V_{IN} to V_{CC} and then lower V_{IN} to 3.0V on pins 2–16, 26–32, and 34–39.
3. Characterization conditions are a) Frequency = 1 MHz; b) Unmeasured pins at GND; c) V_{IN} at +5.0V or GND.
4. An external driver must source at least I_{BHLO} to switch this node from LOW to HIGH.
5. An external driver must sink at least I_{BHHO} to switch this node from HIGH to LOW.
6. V_{IH} for MN/$\overline{MX}$ is 2.5V.
7. This spec may improve to 500 μA during 1986.

A.C. CHARACTERISTICS (80C86: $T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$) (80C86-2: $T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 5\%$)

MINIMUM COMPLEXITY SYSTEM TIMING REQUIREMENTS

Symbol	Parameter	80C86		80C86-2		Units	Test Conditions
		Min	Max	Min	Max		
TCLCL	CLK Cycle Period	200	D.C.	125	D.C.	ns	
TCLCH	CLK Low Time	118		68		ns	
TCHCL	CLK High Time	69		44		ns	
TCH1CH2	CLK Rise Time		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		20		ns	$C_L = 20-100$ pF
TCLDX	Data in Hold Time	10		10		ns	
TR1VCL	RDY Setup Time into 82C84A (See Notes 1, 2)	35		35		ns	
TCLR1X	RDY Hold Time into 82C84A (See Notes 1, 2)	0		0		ns	
TRYHCH	READY Setup Time into 80C86	118		68		ns	
TCHRYX	READY Hold Time into 80C86	30		20		ns	
TRYLCL	READY Inactive to CLK (See Note 3)	−8		−8		ns	
THVCH	HOLD Setup Time	35		20		ns	
TINVCH	INTR, NMI, $\overline{TEST}$ Setup Time (See Note 2)	30		15		ns	
TILIH	Input Rise Time (Except CLK)		15		15	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		15		15	ns	From 2.0V to 0.8V

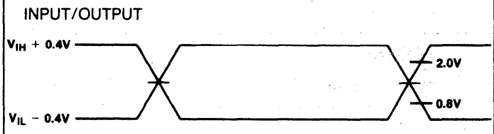
A.C. CHARACTERISTICS (Continued)

Timing Responses

Symbol	Parameter	80C86		80C86-2		Units	Test Conditions
		Min	Max	Min	Max		
TCLAV	Address Valid Delay	10	110	10	60	ns	*C_L = 20–100 pF for all 80C86 Outputs (in addition to 80C86 self-load)
TCLAX	Address Hold Time	10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	TCLAX	50	ns	
TLHLL	ALE Width	TCLCH − 20		TCLCH − 10		ns	
TCLLH	ALE Active Delay		80		50	ns	
TCHLL	ALE Inactive Delay		85		55	ns	
TLLAX	Address Hold Time to ALE Inactive	TCHCL − 10		TCHCL − 10		ns	
TCLDV	Data Valid Delay	10	110	10	60	ns	
TCHDX	Data Hold Time	10		10		ns	
TWHDX	Data Hold Time After WR	TCLCH − 30		TCLCH − 30		ns	
TCVCTV	Control Active Delay 1	10	110	10	70	ns	
TCHCTV	Control Active Delay 2	10	110	10	60	ns	
TCVCTX	Control Inactive Delay	10	110	10	70	ns	
TAZRL	Address Float to READ Active	0		0		ns	
TCLRL	$\overline{RD}$ Active Delay	10	165	10	100	ns	
TCLRH	$\overline{RD}$ Inactive Delay	10	150	10	80	ns	
TRHAV	$\overline{RD}$ Inactive to Next Address Active	TCLCL − 45		TCLCL − 40		ns	
TCLHAV	HLDA Valid Delay	10	160	10	100	ns	
TRLRH	$\overline{RD}$ Width	2TCLCL − 75		2TCLCL − 50		ns	
TWLWH	$\overline{WR}$ Width	2TCLCL − 60		2TCLCL − 40		ns	
TAVAL	Address Valid to ALE Low	TCLCH − 60		TCLCH − 40		ns	
TOLOH	Output Rise Time (Note 4)		15		15	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time (Note 4)		15		15	ns	From 2.0V to 0.8V

NOTES:
1. Signal at 82C84A shown for reference only.
2. Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
3. Applies only to T2 state. (8 ns into T3).
4. Characterization only.

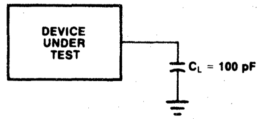
A.C. TESTING INPUT, OUTPUT WAVEFORM



231200-11

A.C. Testing inputs are driven at V_{IH} + 0.4V for a logic "1" and V_{IL} − 0.4V for a logic "0". The clock is driven at V_{CH} + 0.4V and V_{CL} − 0.4V. Timing measurements are made at 2.0V and 0.8V.

A.C. TESTING LOAD CIRCUIT

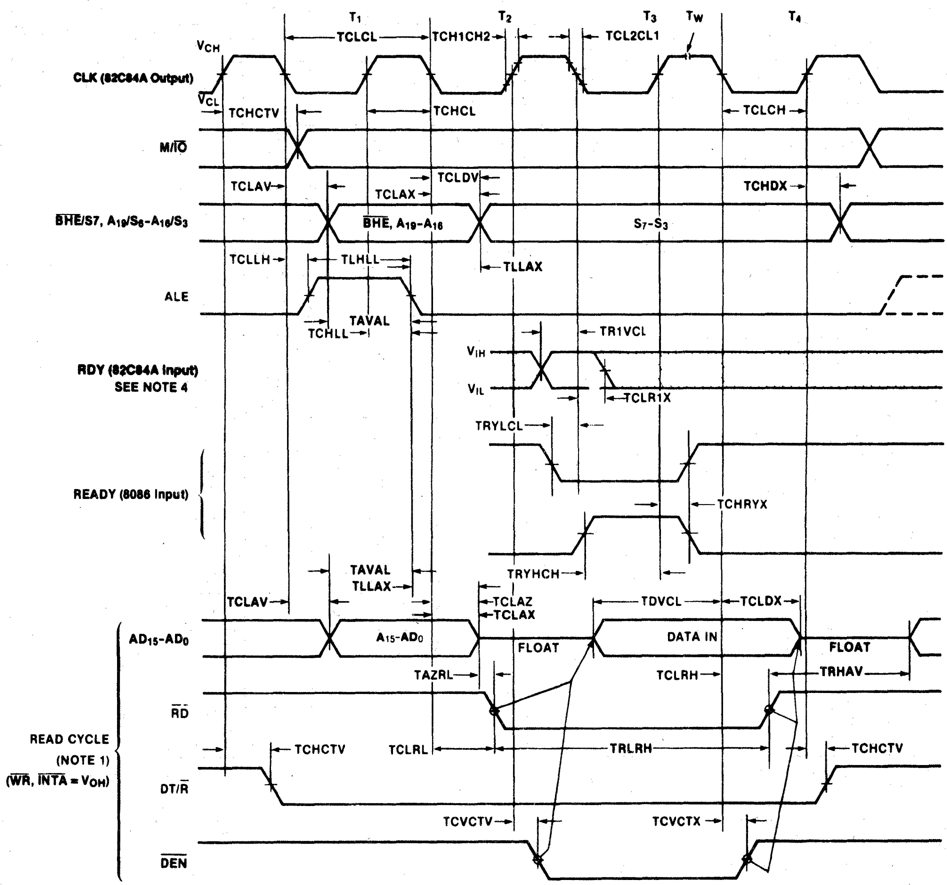


231200-12

C_L Includes Jig Capacitance

WAVEFORMS

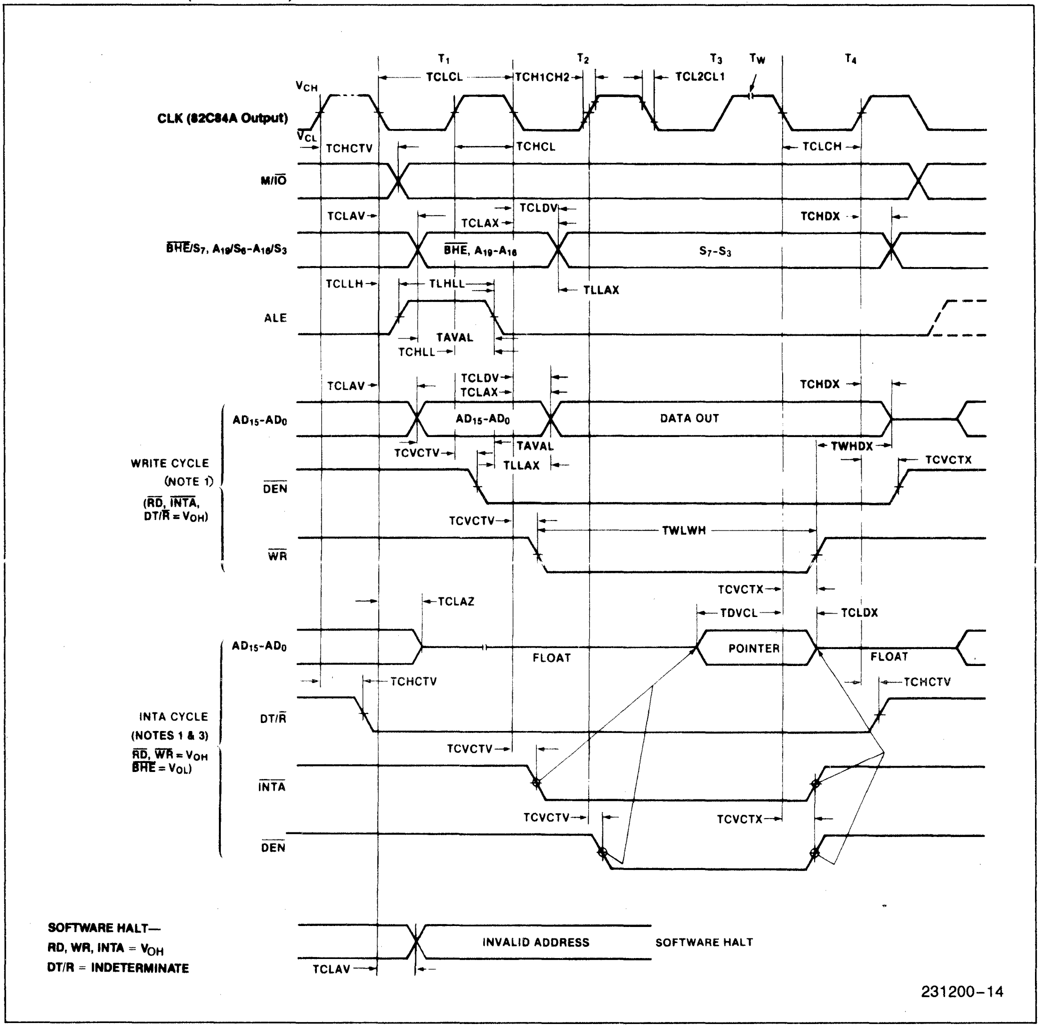
MINIMUM MODE



231200-13

WAVEFORMS (Continued)

MINIMUM MODE (Continued)



NOTES:

1. All timing measurements are made at 0.8V and 2.0V.
2. RDY is sampled near the end of T_2, T_3, T_W to determine if T_W machines states are to be inserted.
3. Two INTA cycles run back-to-back. The 80C86 LOCAL ADDR/DATA BUS is floating during both INTA cycles. Control signals shown for second INTA cycle.
4. Signals at 82C84A are shown for reference only.

A.C. CHARACTERISTICS

MAX MODE SYSTEM (USING 82C88 BUS CONTROLLER)
TIMING REQUIREMENTS

Symbol	Parameter	80C86		80C86-2		Units	Test Conditions
		Min	Max	Min	Max		
TCLCL	CLK Cycle Period	200	D.C.	125	D.C.	ns	
TCLCH	CLK Low Time	118		68		ns	
TCHCL	CLK High Time	69		44		ns	
TCH1CH2	CLK Rise Time		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		20		ns	C_L = 20–100 pF
TCLDX	Data in Hold Time	10		10		ns	
TR1VCL	RDY Setup Time into 82C84A (Notes 1, 2)	35		35		ns	
TCLR1X	RDY Hold Time into 82C84A (Notes 1, 2)	0		0		ns	
TRYHCH	READY Setup Time into 80C86	118		68		ns	
TCHRYX	READY Hold Time into 80C86	30		20		ns	
TRYLCL	READY Inactive to CLK (Note 4)	−8		−8		ns	
TINVCH	Setup Time for Recognition (INTR, NMI, $\overline{TEST}$) (Note 2)	30		15		ns	
TGVCH	$\overline{RQ}/\overline{GT}$ Setup Time	30		15		ns	
TCHGX	$\overline{RQ}$ Hold Time into 80C86	40		30		ns	
TILIH	Input Rise Time (Except CLK) (Note 5)		15		15	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK) (Note 5)		15		15	ns	From 2.0V to 0.8V

A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES

Symbol	Parameter	80C86		80C86-2		Units	Test Conditions
		Min	Max	Min	Max		
TCLML	Command Active Delay (Note 1)	5	35	5	35	ns	C_L = 20–100 pF for all 80C86 Outputs (in addition to 80C86 self-load)
TCLMH	Command Inactive Delay (Note 1)	5	35	5	35	ns	
TRYHSH	READY Active to Status Passive (Note 3)		110		65	ns	
TCHSV	Status Active Delay	10	110	10	60	ns	
TCLSH	Status Inactive Delay	10	130	10	70	ns	
TCLAV	Address Valid Delay	10	110	10	60	ns	
TCLAX	Address Hold Time	10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	TCLAX	50	ns	
TSVLH	Status Valid to ALE High (Note 1)		20		20	ns	
TSVMCH	Status Valid to MCE High (Note 1)		30		30	ns	
TCLLH	CLK Low to ALE Valid (Note 1)		20		20	ns	
TCLMCH	CLK Low to MCE High (Note 1)		25		25	ns	
TCHLL	ALE Inactive Delay (Note 1)	4	25	4	25	ns	
TCLDV	Data Valid Delay	10	110	10	60	ns	
TCHDX	Data Hold Time	10		10		ns	
TCVNV	Control Active Delay (Note 1)	5	45	5	45	ns	
TCVNX	Control Inactive Delay (Note 1)	10	45	10	45	ns	
TAZRL	Address Float to Read Active	0		0		ns	
TCLRL	RD Active Delay	10	165	10	100	ns	
TCLRH	RD Inactive Delay	10	150	10	80	ns	

A.C. CHARACTERISTICS (Continued)

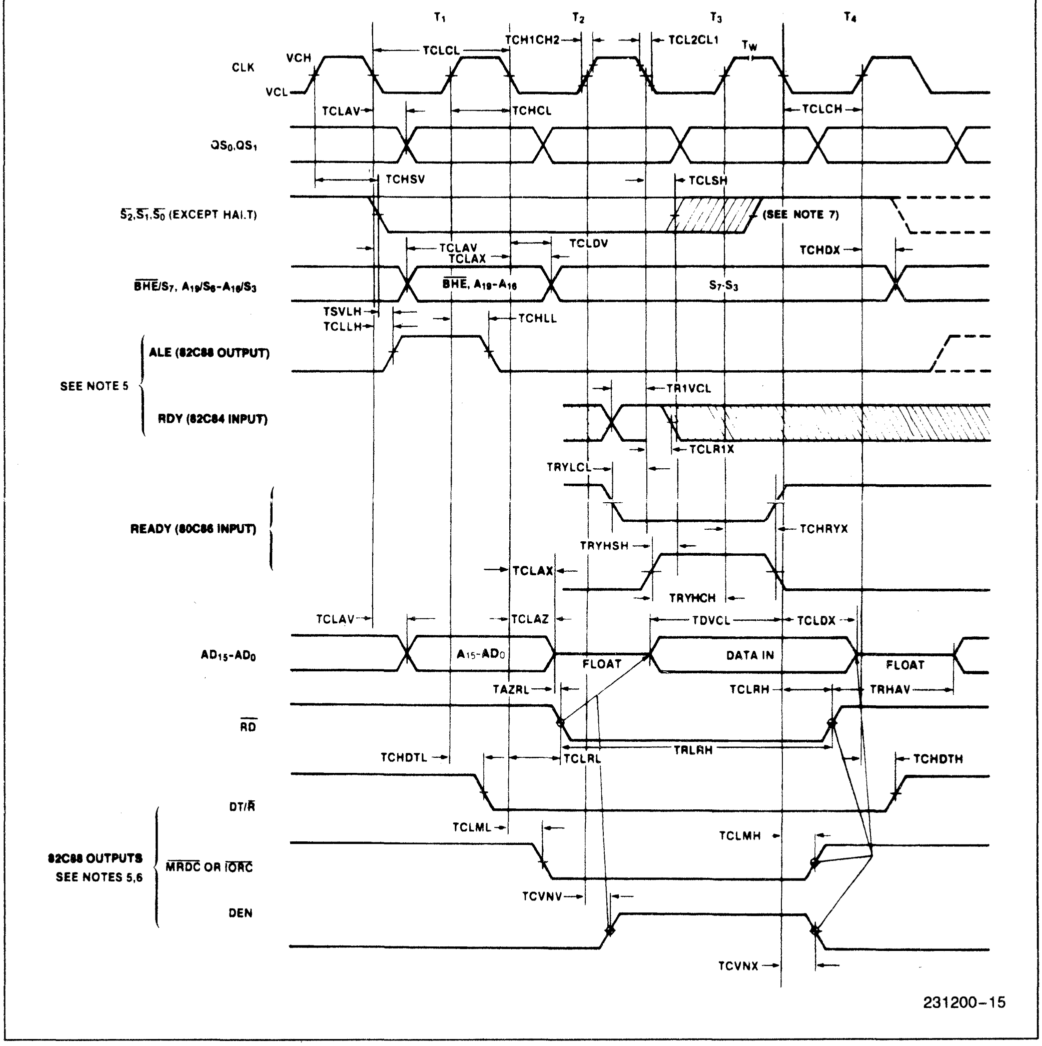
TIMING RESPONSES (Continued)

Symbol	Parameter	80C86		80C86-2		Units	Test Conditions
		Min	Max	Min	Max		
TRHAV	RD Inactive to Next Address Active	TCLCL−45		TCLCL−40		ns	C_L = 20−100 pF for all 80C86 Outputs (in addition to 80C86 self-load)
TCHDTL	Direction Control Active Delay (Note 1)		50		50	ns	
TCHDTH	Direction Control Inactive Delay (Note 1)		30		30	ns	
TCLGL	GT Active Delay	0	85	0	50	ns	
TCLGH	GT Inactive Delay	0	85	0	50	ns	
TRLRH	RD Width	2TCLCL−75		2TCLCL−50		ns	
TOLOH	Output Rise Time		15		15	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		15		15	ns	From 2.0V to 0.8V

NOTES:
1. Signal at 82C84A or 82C88 shown for reference only.
2. Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
3. Applies only to T3 and wait states.
4. Applies only to T2 state (8 ns into T3).
5. Characterization only.

WAVEFORMS

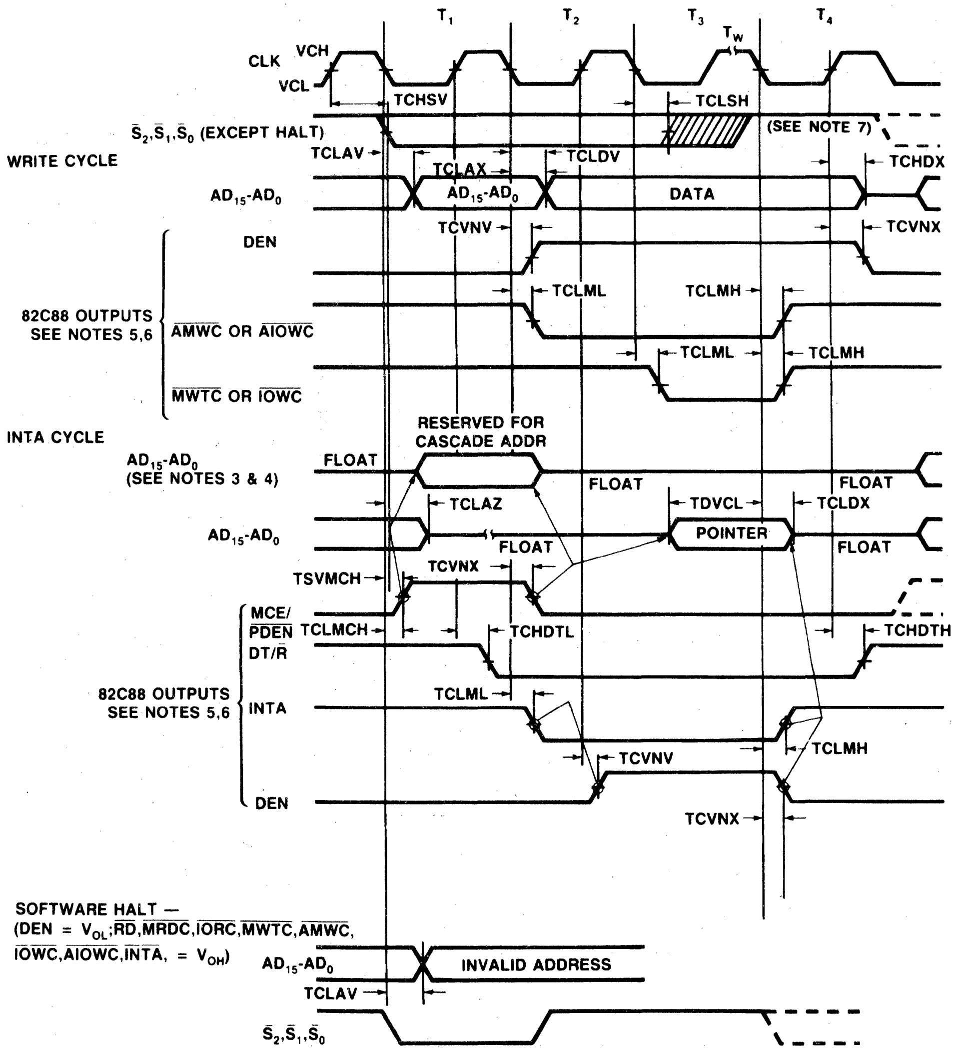
MAXIMUM MODE



231200–15

WAVEFORMS (Continued)

MAXIMUM MODE (Continued)



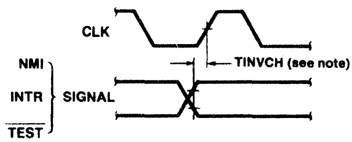
231200-16

NOTES:

1. All timing measurements are made at 2.0V and 0.8V.
2. RDY is sampled near the end of T_2, T_3, T_W to determine if T_W machines states are to be inserted.
3. Cascade address is valid between first and second INTA cycle.
4. Two INTA cycles run back-to-back. The 80C86 LOCAL ADDR/DATA BUS is floating during both INTA cycles. Control for pointer address is shown for second INTA cycle.
5. Signals at 82C84A or 82C88 are shown for reference only.
6. The issuance of the 82C88 command and control signals ($\overline{MRDC}$, $\overline{MWTC}$, $\overline{AMWC}$, $\overline{IORC}$, $\overline{IOWC}$, $\overline{AIOWC}$, $\overline{INTA}$ and DEN) lags the active high 82C88 CEN.
7. Status inactive in state just prior to T_4.

WAVEFORMS (Continued)

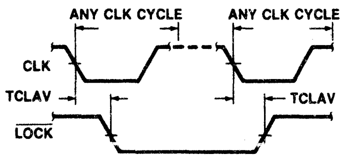
ASYNCHRONOUS SIGNAL RECOGNITION



231200-17

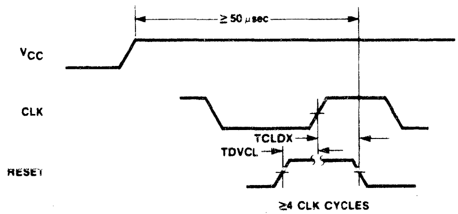
NOTE: Setup requirements for asynchronous signals only to guarantee recognition at next CLK.

BUS LOCK SIGNAL TIMING (MAXIMUM MODE ONLY)



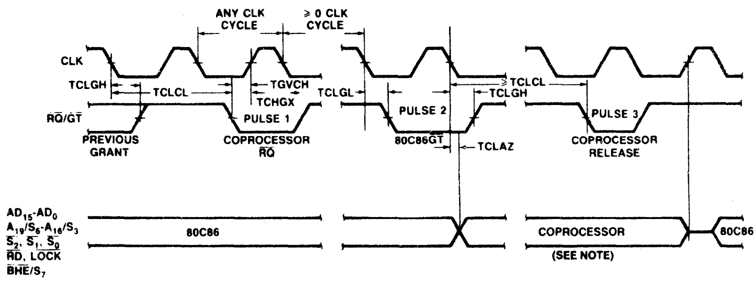
231200-18

RESET TIMING



231200-19

REQUEST/GRANT SEQUENCE TIMING (MAXIMUM MODE ONLY)

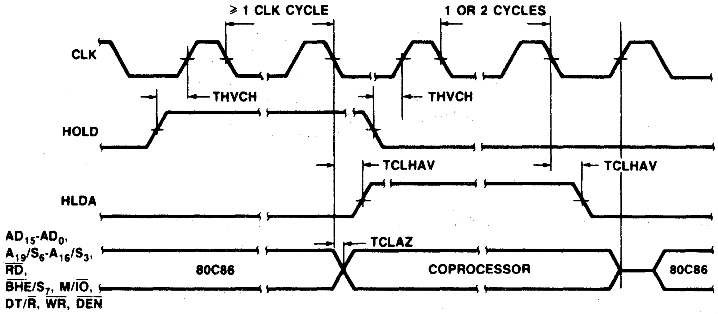


231200-20

NOTE: The coprocessor may not drive the buses outside the region shown without risking contention.

WAVEFORMS (Continued)

HOLD/HOLD ACKNOWLEDGE TIMING (MINIMUM MODE ONLY)



231200-21

Table 2. Instruction Set Summary

	76543210	76543210	76543210	76543210
DATA TRANSFER				
MOV = Move:				
Register/memory to/from register	100010dw	mod reg r/m		
Immediate to register/memory	1100011w	mod 000 r/m	data	data if w 1
Immediate to register	1011 w reg	data	data if w 1	
Memory to accumulator	1010000w	add-low	addr-high	
Accumulator to memory	1010001w	addr-low	addr-high	
Register/memory to segment register**	10001110	mod 0 reg r/m		
Segment register to register/memory	10001100	mod 0 reg r/m		
PUSH = Push:				
Register/memory	11111111	mod 110 r/m		
Register	01010 reg			
Segment register	000 reg 110			
POP = Pop:				
Register/memory	10001111	mod 000 r/m		
Register	01011 reg			
Segment register	000 reg 111			
XCHG = Exchange:				
Register/memory with register	1000011w	mod reg r/m		
Register with accumulator	10010 reg			
IN = Input from:				
Fixed port	1110010w	port		
Variable port	1110110w			
OUT = Output to:				
Fixed port	1110011w	port		
Variable port	1110111w			
XLAT = Translate byte to AL	11010111			
LEA = Load EA to register	10001101	mod reg r/m		
LDS = Load pointer to DS	11000101	mod reg r/m		
LES = Load pointer to ES	11000100	mod reg r/m		
LAHF = Load AH with flags	10011111			
SAHF = Store AH into flags	10011110			
PUSHF = Push flags	10011100			
POPF = Pop flags	10011101			
ARITHMETIC				
ADD = Add:				
Reg./memory with register to either	000000dw	mod reg r/m		
Immediate to register/memory	100000sw	mod 000 r/m	data	data if s:w = 01
Immediate to accumulator	0000010w	data	data if w 1	
ADC = Add with carry:				
Reg/memory with register to either	000100dw	mod reg r/m		
Immediate to register/memory	100000sw	mod 010 r/m	data	data if s:w = 01
Immediate to accumulator	0001010w	data	data if w = 1	
INC = Increment:				
Register/memory	1111111w	mod 000 r/m		
Register	01000 reg			
AAA = ASCII adjust for add	00110111			
DAA = Decimal adjust for add	00100111			
SUB = Subtract:				
Reg./memory and register to either	001010dw	mod reg r/m		
Immediate from register/memory	100000sw	mod 101 r/m	data	data if s:w = 01
Immediate from accumulator	0010110w	data	data if w = 1	
SBB = Subtract with borrow				
Reg./memory and register to either	000110dw	mod reg r/m		
Immediate from register/memory	100000sw	mod 011 r/m	data	data if s:w = 01
Immediate from accumulator	0001110w	data	data if w = 1	

	76543210	76543210	76543210	76543210
DEC Decrement:				
Register/memory	1111111w	mod 001 r/m		
Register	01001 reg			
NEG Change sign	1111011w	mod 011 r/m		
CMP Compare:				
Register/memory and register	001110dw	mod reg r/m		
Immediate with register/memory	100000sw	mod 111 r/m	data	data if s:w = 01
Immediate with accumulator	0011110w	data	data if w = 1	
AAS ASCII adjust for subtract	00111111			
DAS Decimal adjust for subtract	00101111			
MUL Multiply (unsigned)	1111011w	mod 100 r/m		
IMUL Integer multiply (signed)	1111011w	mod 101 r/m		
AAM ASCII adjust for multiply	11010100	00001010		
DIV Divide (unsigned)	1111011w	mod 110 r/m		
IDIV Integer divide (signed)	1111011w	mod 111 r/m		
AAD ASCII adjust for divide	11010101	00001010		
CBW Convert byte to word	10011000			
CWD Convert word to double word	10011001			
LOGIC				
NOT Invert	1111011w	mod 010 r/m		
SHL/SAL Shift logical/arithmetic left	110100vw	mod 100 r/m		
SHR Shift logical right	110100vw	mod 101 r/m		
SAR Shift arithmetic right	110100vw	mod 111 r/m		
ROL Rotate left	110100vw	mod 000 r/m		
ROR Rotate right	110100vw	mod 001 r/m		
RCL Rotate through carry flag left	110100vw	mod 010 r/m		
RCR Rotate through carry right	110100vw	mod 011 r/m		
AND And:				
Reg./memory and register to either	001000dw	mod reg r/m		
Immediate to register/memory	1000000w	mod 100 r/m	data	data if w = 1
Immediate to accumulator	0010010w	data	data if w = 1	
TEST And function to flags, no result:				
Register/memory and register	1000010w	mod reg r/m		
Immediate data and register/memory	1111011w	mod 000 r/m	data	data if w = 1
Immediate data and accumulator	1010100w	data	data if w = 1	
OR Or:				
Reg./memory and register to either	000010dw	mod reg r/m		
Immediate to register/memory	1000000w	mod 001 r/m	data	data if w = 1
Immediate to accumulator	0000110w	data	data if w = 1	
XOR = Exclusive or:				
Reg./memory and register to either	001100dw	mod reg r/m		
Immediate to register/memory	1000000w	mod 110 r/m	data	data if w = 1
Immediate to accumulator	0011010w	data	data if w = 1	
STRING MANIPULATION				
REP = Repeat	1111001z			
MOVS = Move byte/word	1010010w			
CMPS = Compare byte/word	1010011w			
SCAS = Scan byte/word	1010111w			
LODS = Load byte/wd to AL/AX	1010110w			
STOS = Stor byte/wd from AL/A	1010101w			

231200-22

Table 2. Instruction Set Summary (Continued)

CONTROL TRANSFER

Instruction	76543210	76543210	76543210
CALL = Call:			
Direct within segment	11101000	disp-low	disp-high
Indirect within segment	11111111	mod 0 1 0 r/m	
Direct intersegment	10011010	offset-low	offset-high
		seg-low	seg-high
Indirect intersegment	11111111	mod 0 1 1 r/m	
JMP = Unconditional Jump:			
Direct within segment	11101001	disp-low	disp-high
Direct within segment-short	11101011	disp	
Indirect within segment	11111111	mod 1 0 0 r/m	
Direct intersegment	11101010	offset-low	offset-high
		seg-low	seg-high
Indirect intersegment	11111111	mod 1 0 1 r/m	
RET = Return from CALL:			
Within segment	11000011		
Within seg. adding immed to SP	11000010	data-low	data-high
Intersegment	11001011		
Intersegment adding immediate to SP	11001010	data-low	data-high
JE/JZ = Jump on equal/zero	01110100	disp	
JL/JNGE = Jump on less/not greater or equal	01111100	disp	
JLE/JNG = Jump on less or equal/not greater	01111110	disp	
JB/JNAE = Jump on below/not above or equal	01110010	disp	
JBE/JNA = Jump on below or equal/not above	01110110	disp	
JP/JPE = Jump on parity/parity even	01111010	disp	
JO = Jump on overflow	01110000	disp	
JS = Jump on sign	01111000	disp	
JNE/JNZ = Jump on not equal/not zero	01110101	disp	
JNL/JGE = Jump on not less/greater or equal	01111101	disp	
JNLE/JG = Jump on not less or equal/greater	01111111	disp	

Instruction	76543210	76543210
JNB/JAE Jump on not below/above or equal	01110011	disp
JNBE/JA Jump on not below or equal/above	01110111	disp
JNP/JPO = Jump on not par/par odd	01111011	disp
JNO = Jump on not overflow	01110001	disp
JNS Jump on not sign	01111001	disp
LOOP Loop CX times	11100010	disp
LOOPZ/LOOPE Loop while zero/equal	11100001	disp
LOOPNZ/LOOPNE Loop while not zero/equal	11100000	disp
JCXZ Jump on CX zero	11100011	disp
INT Interrupt		
Type specified	11001101	type
Type 3	11001100	
INTO = Interrupt on overflow	11001110	
IRET Interrupt return	11001111	

PROCESSOR CONTROL

Instruction	76543210	76543210
CLC Clear carry	11111000	
CMC Complement carry	11110101	
STC Set carry	11111001	
CLD Clear direction	11111100	
STD Set direction	11111101	
CLI Clear interrupt	11111010	
STI Set interrupt	11111011	
HLT Halt	11110100	
WAIT Wait	10011011	
ESC Escape (to external device)	11011xxx	mod x x x r/m
LOCK Bus lock prefix	11110000	

Footnotes:

AL = 8-bit accumulator
AX = 16-bit accumulator
CX = Count register
DS = Data segment
ES = Extra segment
Above/below refers to unsigned value.
Greater = more positive;
Less = less positive (more negative) signed values
if d = 1 then "to" reg; if d = 0 then "from" reg
if w = 1 then word instruction; if w = 0 then byte instruction

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low

if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP
DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.
**MOV CS, Req. 1 Memory not allowed.
Mnemonics © Intel, 1978

if s:w = 01 then 16 bits of immediate data form the operand.
if s:w = 11 then an immediate data byte is sign extended to form the 16-bit operand.
if v = 0 then "count" = 1; if v = 1 then "count" in (CL)
x = don't care
z is used for string primitives for comparison with ZF FLAG.

SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

REG is assigned according to the following table:

16-Bit [w = 1]	8-bit [w = 0]	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:

FLAGS = X:X:X:X:(OF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF)

231200–23

intel®

80186
HIGH INTEGRATION 16-BIT MICROPROCESSOR

- Integrated Feature Set
  - —Enhanced 8086-2 CPU
  - —Clock Generator
  - —2 Independent, High-Speed DMA Channels
  - —Programmable Interrupt Controller
  - —3 Programmable 16-bit Timers
  - —Programmable Memory and Peripheral Chip-Select Logic
  - —Programmable Wait State Generator
  - —Local Bus Controller
- Available in 12.5 MHz (80186-12), 10 MHz (80186-10) and 8 MHz (80186) Versions
- High-Performance Processor
  - —At 8 MHz provides 2 times the Performance of the Standard 8086
  - —4 MByte/Sec Bus Bandwidth Interface @ 8 MHz
  - —6.25 MByte/Sec Bus Bandwidth Interface @ 12.5 MHz
- Direct Addressing Capability to 1 MByte of Memory and 64 KByte I/O
- Completely Object Code Compatible with All Existing 8086, 8088 Software
  - —10 New Instruction Types
- Complete System Development Support
  - —Development Software; Assembler, PL/M, Pascal, Fortan, and System Utilities
  - —In-Circuit-Emulator (I2ICE™-186)
- High Performance Numerical Coprocessing Capability Through 8087 Interface
- Available in 68 Pin:
  - —Plastic Leaded Chip Carrier (PLCC)
  - —Ceramic Pin Grid Array (PGA)
  - —Ceramic Leadless Chip Carrier (LCC)

(See Packaging Spec, Order #231369)

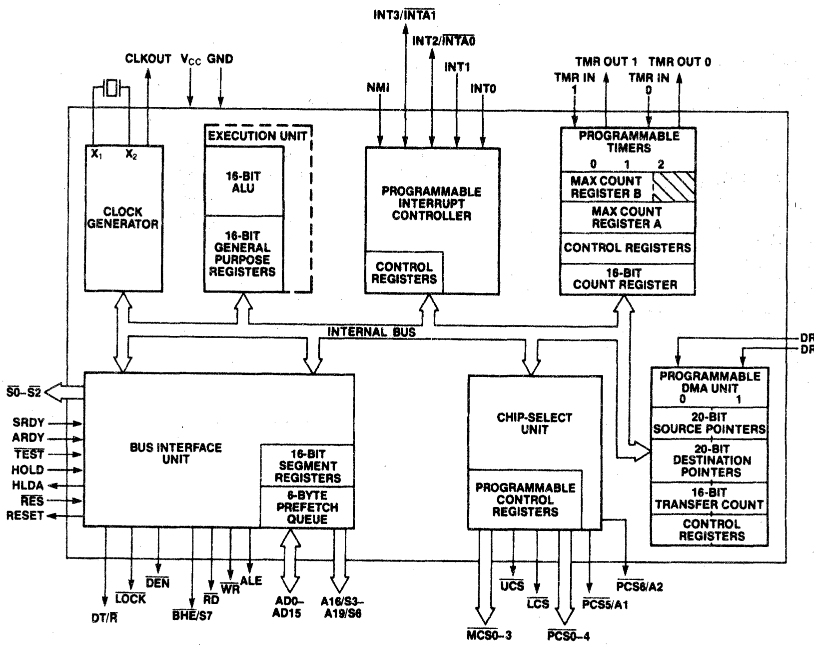


Figure 1. 80186 Block Diagram

Order Number: 210451-006

The Intel 80186 is a highly integrated 16-bit microprocessor. The 80186 effectively combines 15–20 of the most common 8086 system components onto one. The 80186 provides two times greater throughput than the standard 5 MHz 8086. The 80186 is upward compatible with 8086 and 8088 software and adds 10 new instruction types to the existing set.

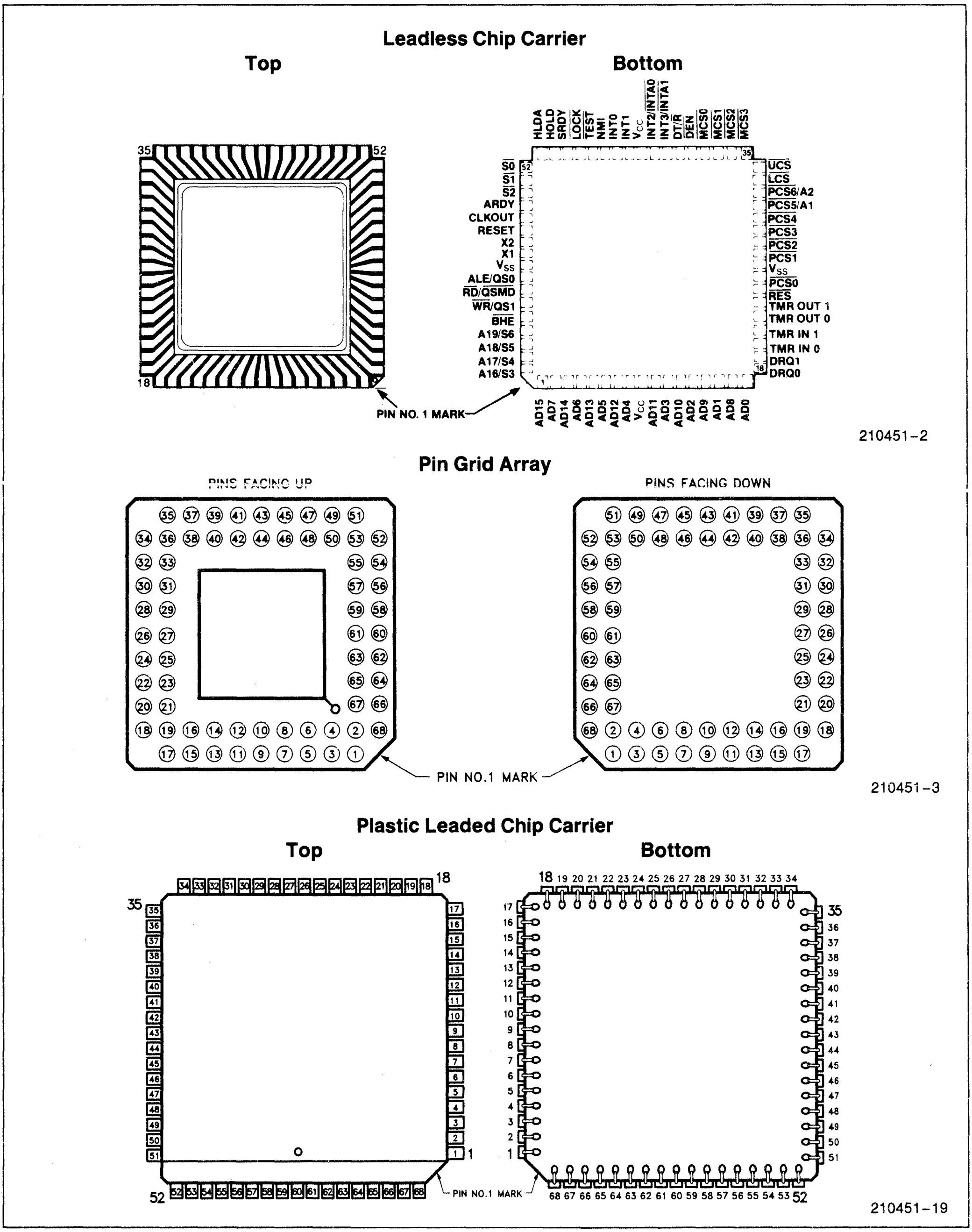


Figure 2. 80186 Pinout Diagrams

Table 1. 80186 Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}, V_{CC}	9, 43	I	**System Power:** +5 volt power supply.
V_{SS}, V_{SS}	26, 60	I	System Ground.
RESET	57	O	Reset Output indicates that the 80186 CPU is being reset, and can be used as a system reset. It is active HIGH, synchronized with the processor clock, and lasts an integer number of clock periods corresponding to the length of the $\overline{RES}$ signal.
X1, X2	59, 58	I	Crystal Inputs, X1 and X2, provide an external connection for a fundamental mode parallel resonant crystal for the internal crystal oscillator. X1 can interface to an external clock instead of a crystal. In this case, minimize the capacitance on X2 or drive X2 with complemented X1. The input or oscillator frequency is internally divided by two to generate the clock signal (CLKOUT).
CLKOUT	56	O	Clock Output provides the system with a 50% duty cycle waveform. All device pin timings are specified relative to CLKOUT. CLKOUT has sufficient MOS drive capabilities for the 8087 Numeric Processor Extension.
$\overline{RES}$	24	I	System Reset causes the 80186 to immediately terminate its present activity, clear the internal logic, and enter a dormant state. This signal may be asynchronous to the 80186 clock. The 80186 begins fetching instructions approximately 7 clock cycles after $\overline{RES}$ is returned HIGH. $\overline{RES}$ is required to be LOW for greater than 4 clock cycles and is internally synchronized. For proper initialization, the LOW-to-HIGH transition of $\overline{RES}$ must occur no sooner than 50 microseconds after power up. This input is provided with a Schmitt-trigger to facilitate power-on $\overline{RES}$ generation via an RC network. When $\overline{RES}$ occurs, the 80186 will drive the status lines to an inactive level for one clock, and then tri-state them.
$\overline{TEST}$	47	I	$\overline{TEST}$ is examined by the WAIT instruction. If the $\overline{TEST}$ input is HIGH when "WAIT" execution begins, instruction execution will suspend. $\overline{TEST}$ will be resampled until it goes LOW, at which time execution will resume. If interrupts are enabled while the 80186 is waiting for $\overline{TEST}$, interrupts will be serviced. This input is synchronized internally.
TMR IN 0, TMR IN 1	20 21	I I	Timer Inputs are used either as clock or control signals, depending upon the programmed timer mode. These inputs are active HIGH (or LOW-to-HIGH transitions are counted) and internally synchronized.
TMR OUT 0, TMR OUT 1	22 23	O O	Timer outputs are used to provide single pulse or continous waveform generation, depending upon the timer mode selected.
DRQ0 DRQ1	18 19	I I	DMA Request is driven HIGH by an external device when it desires that a DMA channel (Channel 0 or 1) perform a transfer. These signals are active HIGH, level-triggered, and internally synchronized.
NMI	46	I	Non-Maskable Interrupt is an edge-triggered input which causes a type 2 interrupt. NMI is not maskable internally. A transition from a LOW to HIGH initiates the interrupt at the next instruction boundary. NMI is latched internally. An NMI duration of one clock or more will guarantee service. This input is internally synchronized.
INT0, INT1 INT2/$\overline{INTA0}$ INT3/$\overline{INTA1}$	45, 44 42 41	I I/O I/O	Maskable Interrupt Requests can be requested by strobing one of these pins. When configured as inputs, these pins are active HIGH. Interrupt Requests are synchronized internally. INT2 and INT3 may be configured via software to provide active-LOW interrupt-acknowledge output signals. All interrupt inputs may be configured via software to be either edge- or level-triggered. To ensure recognition, all interrupt requests must remain active until the interrupt is acknowledged. When iRMX mode is selected, the function of these pins changes (see Interrupt Controller section of this data sheet).

Table 1. 80186 Pin Description (Continued)

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th colspan="3">Name and Function</th></tr>
<tr><td rowspan="4">A19/S6,
A18/S5,
A17/S4,
A16/S3</td><td rowspan="4">65
66
67
68</td><td rowspan="4">O
O
O
O</td><td colspan="3">Address Bus Outputs (16–19) and Bus Cycle Status (3–6) reflect the four most significant address bits during T_1. These signals are active HIGH. During T_2, T_3, T_W, and T_4, status information is available on these lines as encoded below:</td></tr>
<tr><td></td><td>Low</td><td>High</td></tr>
<tr><td>S6</td><td>Processor Cycle</td><td>DMA Cycle</td></tr>
<tr><td colspan="3">S3, S4, and S5 are defined as LOW during T_2–T_4.</td></tr>
<tr><td>AD15–AD0</td><td>10–17,
1–8</td><td>I/O</td><td colspan="3">Address/Data Bus (0–15) signals constitute the time multiplexed memory or I/O address (T_1) and data (T_2, T_3, T_W, and T_4) bus. The bus is active HIGH. A_0 is analogous to $\overline{BHE}$ for the lower byte of the data bus, pins D_7 through D_0. It is LOW during T_1 when a byte is to be transferred onto the lower portion of the bus in memory or I/O operations.</td></tr>
<tr><td rowspan="7">$\overline{BHE}$/S7</td><td rowspan="7">64</td><td rowspan="7">O</td><td colspan="3">During T_1 the Bus High Enable signal should be used to determine if data is to be enabled onto the most significant half of the data bus; pins D_{15}–D_8. $\overline{BHE}$ is LOW during T_1 for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the higher half of the bus. The S_7 status information is available during T_2, T_3, and T_4. S_7 is logically equivalent to $\overline{BHE}$. The signal is active LOW, and is tristated OFF during bus HOLD.</td></tr>
<tr><td colspan="3">$\overline{BHE}$ and A0 Encodings</td></tr>
<tr><td>$\overline{BHE}$ Value</td><td>A0 Value</td><td>Function</td></tr>
<tr><td>0</td><td>0</td><td>Word Transfer</td></tr>
<tr><td>0</td><td>1</td><td>Byte Transfer on upper half of data bus (D15–D8)</td></tr>
<tr><td>1</td><td>0</td><td>Byte Transfer on lower half of data bus (D_7–D_0)</td></tr>
<tr><td>1</td><td>1</td><td>Reserved</td></tr>
<tr><td>ALE/QS0</td><td>61</td><td>O</td><td colspan="3">Address Latch Enable/Queue Status 0 is provided by the 80186 to latch the address into the 8282/8283 address latches. ALE is active HIGH. Addresses are guaranteed to be valid on the trailing edge of ALE. The ALE rising edge is generated off the rising edge of the CLKOUT immediately preceding T_1 of the associated bus cycle, effectively one-half clock cycle earlier than in the standard 8086. The trailing edge is generated off the CLKOUT rising edge in T_1 as in the 8086. Note that ALE is never floated.</td></tr>
<tr><td rowspan="6">$\overline{WR}$/QS1</td><td rowspan="6">63</td><td rowspan="6">O</td><td colspan="3">Write Strobe/Queue Status 1 indicates that the data on the bus is to be written into a memory or an I/O device. $\overline{WR}$ is active for T_2, T_3, and T_W of any write cycle. It is active LOW, and floats during "HOLD." It is driven HIGH for one clock during Reset, and then floated. When the 80186 is in queue status mode, the ALE/QS0 and $\overline{WR}$/QS1 pins provide information about processor/instruction queue interaction.</td></tr>
<tr><td>QS1</td><td>QS0</td><td>Queue Operation</td></tr>
<tr><td>0</td><td>0</td><td>No queue operation</td></tr>
<tr><td>0</td><td>1</td><td>First opcode byte fetched from the queue</td></tr>
<tr><td>1</td><td>1</td><td>Subsequent byte fetched from the queue</td></tr>
<tr><td>1</td><td>0</td><td>Empty the queue</td></tr>
</table>

Table 1. 80186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{RD}/\overline{QSMD}$	62	O	Read Strobe indicates that the 80186 is performing a memory or I/O read cycle. $\overline{RD}$ is active LOW for T_2, T_3, and T_W of any read cycle. It is guaranteed not to go LOW in T_2 until after the Address Bus is floated. $\overline{RD}$ is active LOW, and floats during "HOLD". $\overline{RD}$ is driven HIGH for one clock during Reset, and then the output driver is floated. A weak internal pull-up mechanism of the $\overline{RD}$ line holds it HIGH when the line is not driven. During RESET the pin is sampled to determine whether the 80186 should provide ALE, $\overline{WR}$ and $\overline{RD}$, or if the Queue-Status should be provided. $\overline{RD}$ should be connected to GND to provide Queue-Status data.
ARDY	55	I	Asynchronous Ready informs the 80186 that the addressed memory space or I/O device will complete a data transfer. The ARDY input pin will accept an asynchronous input, and is active HIGH. Only the rising edge is internally synchronized by the 80186. This means that the falling edge of ARDY must be synchronized to the 80186 clock. If connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active to terminate a bus cycle. If unused, this line should be tied LOW.
SRDY	49	I	Synchronous Ready must be synchronized externally to the 80186. The use of SRDY provides a relaxed system-timing specification on the Ready input. This is accomplished by eliminating the one-half clock cycle which is required for internally resolving the signal level when using the ARDY input. This line is active HIGH. If this line is connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active before a bus cycle is terminated. If unused, this line should be tied LOW.
$\overline{LOCK}$	48	O	$\overline{LOCK}$ output indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active LOW. The $\overline{LOCK}$ signal is requested by the LOCK prefix instruction and is activated at the beginning of the first data cycle associated with the instruction following the LOCK prefix. It remains active until the completion of the instruction following the LOCK prefix. No prefetches will occur while $\overline{LOCK}$ is asserted. When executing more than one LOCK instruction, always make sure there are 6 bytes of code between the end of the first LOCK instruction and the start of the second LOCK instruction. $\overline{LOCK}$ is active LOW, is driven HIGH for one clock during RESET, and then floated.
$\overline{S0}$, $\overline{S1}$, $\overline{S2}$	52–54	O	Bus cycle status $\overline{S0}$–$\overline{S2}$ are encoded to provide bus-transaction information:

80186 Bus Cycle Status Information

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

The status pins float during "HOLD."
$\overline{S2}$ may be used as a logical M/$\overline{IO}$ indicator, and $\overline{S1}$ as a DT/$\overline{R}$ indicator.
The status lines are driven HIGH for one clock during Reset, and then floated until a bus cycle begins.

Table 1. 80186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
HOLD (input) HLDA (output)	50 51	I O	HOLD indicates that another bus master is requesting the local bus. The HOLD input is active HIGH. HOLD may be asynchronous with respect to the 80186 clock. The 80186 will issue a HLDA (HIGH) in response to a HOLD request at the end of T_4 or T_1. Simultaneous with the issuance of HLDA, the 80186 will float the local bus and control lines. After HOLD is detected as being LOW, the 80186 will lower HLDA. When the 80186 needs to run another bus cycle, it will again drive the local bus and control lines.
$\overline{UCS}$	34	O	Upper Memory Chip Select is an active LOW output whenever a memory reference is made to the defined upper portion (1K–256K block) of memory. This line is not floated during bus HOLD. The address range activating $\overline{UCS}$ is software programmable.
$\overline{LCS}$	33	O	Lower Memory Chip Select is active LOW whenever a memory reference is made to the defined lower portion (1K–256K) of memory. This line is not floated during bus HOLD. The address range activating $\overline{LCS}$ is software programmable.
$\overline{MCS0}$–3	38, 37, 36, 35	O	Mid-Range Memory Chip Select signals are active LOW when a memory reference is made to the defined mid-range portion of memory (8K–512K). These lines are not floated during bus HOLD. The address ranges activating $\overline{MCS0}$–3 are software programmable.
$\overline{PCS0}$ $\overline{PCS1}$–4	25 27, 28, 29, 30	O O	Peripheral Chip Select signals 0–4 are active LOW when a reference is made to the defined peripheral area (64K byte I/O space). These lines are not floated during bus HOLD. The address ranges activating $\overline{PCS0}$–4 are software programmable.
$\overline{PCS5}$/A1	31	O	Peripheral Chip Select 5 or Latched. A1 may be programmed to provide a sixth peripheral chip select, or to provide an internally latched A1 signal. The address range activating $\overline{PCS5}$ is software programmable. When programmed to provide latched. A1, rather than $\overline{PCS5}$, this pin will retain the previously latched value of A1 during a bus HOLD. A1 is active HIGH.
$\overline{PCS6}$/A2	32	O	Peripheral Chip Select 6 or Latched A2 may be programmed to provide a seventh peripheral chip select, or to provide an internally latched A2 signal. The address range activating $\overline{PCS6}$ is software programmable. When programmed to provide latched A2, rather than $\overline{PCS6}$, this pin will retain the previously latched value of A2 during a bus HOLD. A2 is active HIGH.
DT/$\overline{R}$	40	O	Data Transmit/Receive controls the direction of data flow through the external 8286/8287 data bus transceiver. When LOW, data is transferred to the 80186. When HIGH the 80186 places write data on the data bus.
$\overline{DEN}$	39	O	Data Enable is provided as an 8286/8287 data bus transceiver output enable. $\overline{DEN}$ is active LOW during each memory and I/O access. $\overline{DEN}$ is HIGH whenever DT/$\overline{R}$ changes state.

FUNCTIONAL DESCRIPTION

Introduction

The following Functional Description describes the base architecture of the 80186. This architecture is common to the 8086, 8088, and 80286 microprocessor families as well. The 80186 is a very high integration 16-bit microprocessor. It combines 15–20 of the most common microprocessor system components onto one chip while providing twice the performance of the standard 8086. The 80186 is object code compatible with the 8086/8088 microprocessors and adds 10 new instruction types to the existing 8086/8088 instruction set.

80186 BASE ARCHITECTURE

The 8086, 8088, 80186, and 80286 family all contain the same basic set of registers, instructions, and addressing modes. The 80186 processor is upward compatible with the 8086, 8088, and 80286 CPUs.

Register Set

The 80186 base architecture has fourteen registers as shown in Figures 3a and 3b. These registers are grouped into the following categories.

General Registers

Eight 16-bit general purpose registers may be used to contain arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used as 16-bit registers or split into pairs of separate 8-bit registers.

Segment Registers

Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

Base and Index Registers

Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode selects the specific registers for operand and address calculations.

Status and Control Registers

Two 16-bit special purpose registers record or alter certain aspects of the 80186 processor state. These are the Instruction Pointer Register, which contains the offset address of the next sequential instruction to be executed, and the Status Word Register, which contains status and control flag bits (see Figures 3a and 3b).

Status Word Description

The Status Word records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80186 within a given operating mode (bits 8, 9, and 10). The Status Word Register is 16-bits wide. The function of the Status Word bits is shown in Table 2.

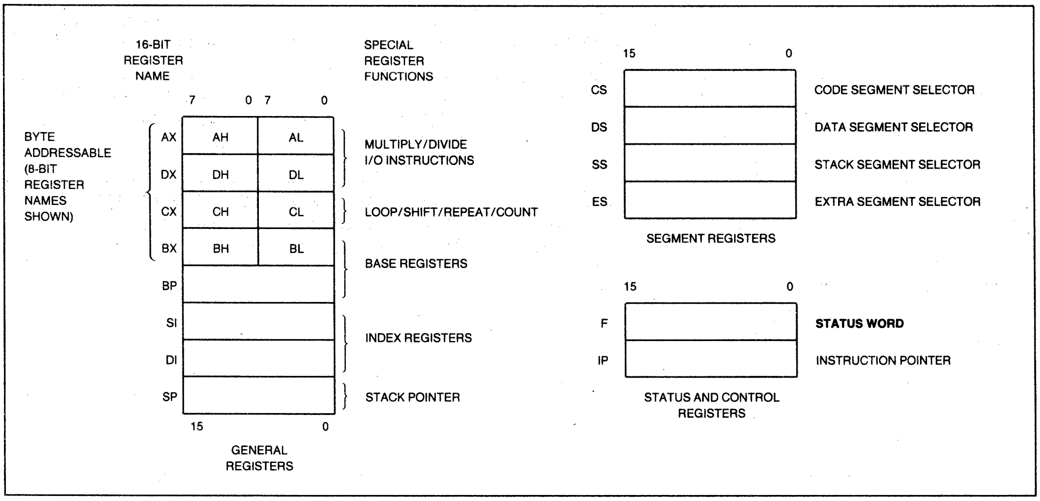


Figure 3a. 80186 General Purpose Register Set

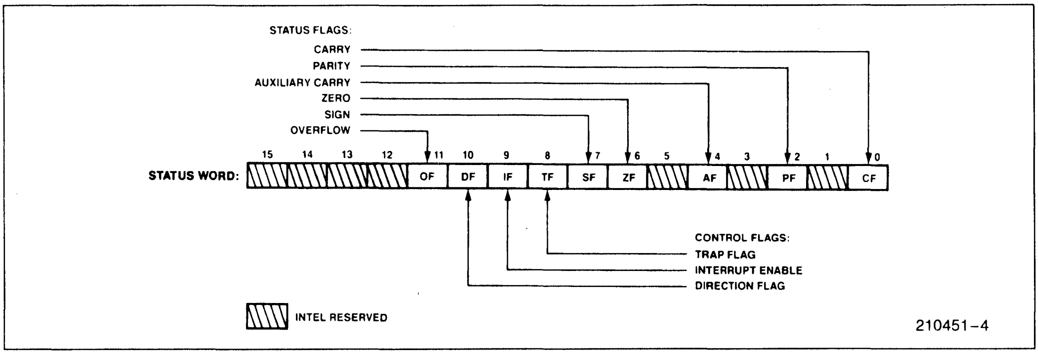


Figure 3b. Status Word Format

Table 2. Status Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag—Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index register when set. Clearing DF causes auto increment.
11	OF	Overflow Flag—Set if the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high-level instructions, and processor control. These categories are summarized in Figure 4.

An 80186 instruction can reference anywhere from zero to several operands. An operand can reside in a register, in the instruction itself, or in memory. Specific operand addressing modes are discussed later in this data sheet.

Memory Organization

Memory is organized in sets of segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two-component address (a pointer) that consists of a 16-bit base segment and a 16-bit offset. The 16-bit base values are contained in one of four internal segment register (code, data, stack, extra). The physical address is calculated by shifting the base value LEFT by four bits and adding the 16-bit offset value to yield a 20-bit physical address (see Figure 5). This allows for a 1 MByte physical address size.

All instructions that address operands in memory must specify the base segment and the 16-bit offset value. For speed and compact instruction encoding, the segment register used for physical address generation is implied by the addressing mode used (see Table 3). These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs.

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack
ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word
FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{TEST}}$ pin active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
HIGH LEVEL INSTRUCTIONS	
ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4. 80186 Instruction Set

CONDITIONAL TRANSFERS			
JA/JNBE	Jump if above/not below nor equal	JO	Jump if overflow
JAE/JNB	Jump if above or equal/not below	JP/JPE	Jump if parity/parity even
JB/JNAE	Jump if below/not above nor equal	JS	Jump if sign
JBE/JNA	Jump if below or equal/not above	**UNCONDITIONAL TRANSFERS**	
JC	Jump if carry	CALL	Call procedure
JE/JZ	Jump if equal/zero	RET	Return from procedure
JG/JNLE	Jump if greater/not less nor equal	JMP	Jump
JGE/JNL	Jump if greater or equal/not less	**ITERATION CONTROLS**	
JL/JNGE	Jump if less/not greater nor equal	LOOP	Loop
JLE/JNG	Jump if less or equal/not greater	LOOPE/LOOPZ	Loop if equal/zero
JNC	Jump if not carry	LOOPNE/LOOPNZ	Loop if not equal/not zero
JNE/JNZ	Jump if not equal/not zero	JCXZ	Jump if register CX = 0
JNO	Jump if not overflow	**INTERRUPTS**	
JNP/JPO	Jump if not parity/parity odd	INT	Interrupt
JNS	Jump if not sign	INTO	Interrupt if overflow
		IRET	Interrupt return

Figure 4. 80186 Instruction Set (Continued)

To access operands that do not reside in one of the four immediately available segments, a full 32-bit pointer can be used to reload both the base (segment) and offset values.

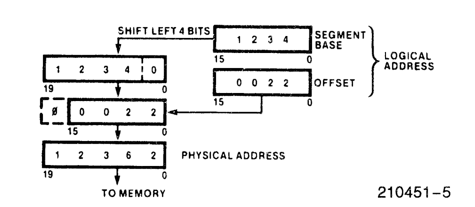


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Instruction prefetch and immediate data.
Stack	Stack (SS)	All stack pushes and pops; any memory references which use BP Register as a base register.
External Data (Global)	Extra (ES)	All string instruction references which use the DI register as an index.
Local Data	Data (DS)	All other data references.

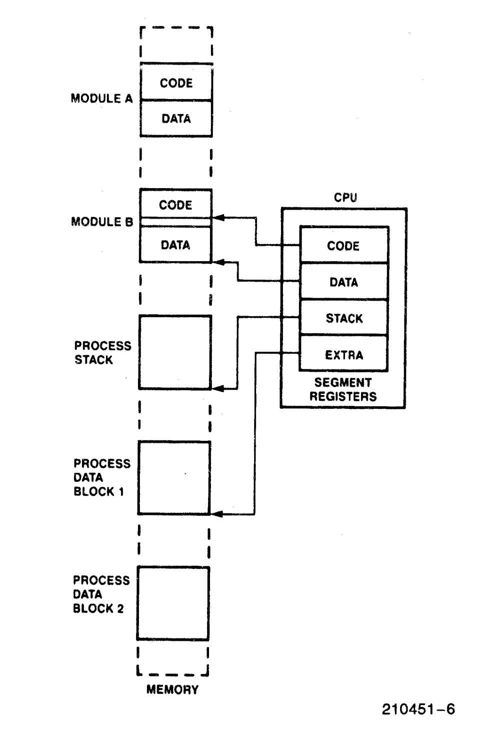


Figure 6. Segmented Memory Helps Structure Software

Addressing Modes

The 80186 provides eight categories of addressing modes to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

- *Register Operand Mode:* The operand is located in one of the 8- or 16-bit general registers.
- *Immediate Operand Mode:* The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: a segment base and an offset. The segment base is supplied by a 16-bit segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset, also called the effective address, is calculated by summing any combination of the following three address elements:

- the *displacement* (an 8- or 16-bit immediate value contained in the instruction);
- the *base* (contents of either the BX or BP base registers); and
- the *index* (contents of either the SI or DI index registers).

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

- *Direct Mode:* The operand's offset is contained in the instruction as an 8- or 16-bit displacement element.
- *Register Indirect Mode:* The operand's offset is in one of the registers SI, DI, BX, or BP.
- *Based Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of a base register (BX or BP).
- *Indexed Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of an index register (SI or DI).
- *Based Indexed Mode:* The operand's offset is the sum of the contents of a base register and an Index register.
- *Based indexed Mode with Displacement:* The operand's offset is the sum of a base register's contents, an index register's contents, and an 8- or 16-bit displacement.

Data Types

The 80186 directly supports the following data types:

- *Integer:* A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32- and 64-bit integers are supported using the 80186/20 Numeric Data Processor.
- *Ordinal:* An unsigned binary numeric value contained in an 8-bit byte or a 16-bit word.
- *Pointer:* A 16- or 32-bit quantity, composed of a 16-bit offset component or a 16-bit segment base component in addition to a 16-bit offset component.
- *String:* A contiguous sequence of bytes or words. A string may contain from 1 to 64K bytes.
- *ASCII:* A byte representation of alphanumeric and control characters using the ASCII standard of character representation.
- *BCD:* A byte (unpacked) representation of the decimal digits 0–9.
- *Packed BCD:* A byte (packed) representation of two decimal digits (0–9). One digit is stored in each nibble (4-bits) of the byte.
- *Floating Point:* A signed 32-, 64-, or 80-bit real number representation. (Floating point operands are supported using the 80186/20 Numeric Data Processor configuration.)

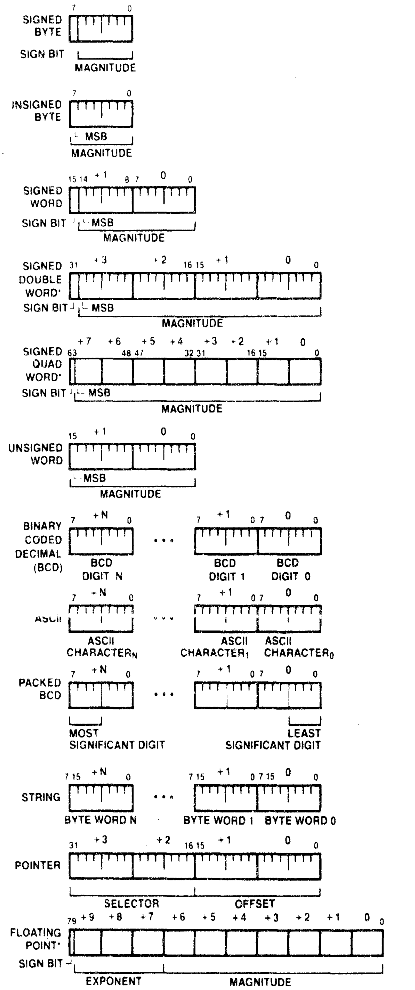
In general, individual data elements must fit within defined segment limits. Figure 7 graphically represents the data types supported by the 80186.

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. Separate instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A_{15}–A_8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Status Word) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable.



210451-7

NOTE:
*Supported by 80186/20 Numeric Data Processor Configuration.

Figure 7. 80186 Supported Data Types

Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. If the exception was caused by executing an ESC instruction with the ESC trap bit set in the relocation register, the return instruction will point to the ESC instruction, or to the segment override prefix immediately preceding the ESC instruction if the prefix was present. In all other cases, the return address from an exception will point at the instruction immediately following the instruction causing the exception.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. Table 4 shows the 80186 predefined types and default priority levels. For each interrupt, an 8-bit vector must be supplied to the 80186 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. In addition, internal peripherals and noncascaded external interrupts will generate their own vectors through the internal interrupt controller. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

Interrupt Sources

The 80186 can service interrupts generated by software or hardware. The software interrupts are generated by specific instructions (INT, ESC, unused OP, etc.) or the results of conditions specified by instructions (array bounds check, INTO, DIV, IDIV, etc.). All interrupt sources are serviced by an indirect call through an element of a vector table. This vector table is indexed by using the interrupt vector type (Table 4), multiplied by four. All hardware-generated interrupts are sampled at the end of each instruction. Thus, the software interrupts will begin service first. Once the service routine is entered and interrupts are enabled, any hardware source of sufficient priority can interrupt the service routine in progress.

The software generated 80186 interrupts are described below.

DIVIDE ERROR EXCEPTION (TYPE 0)

Generated when a DIV or IDIV instruction quotient cannot be expressed in the number of bits in the destination.

SINGLE-STEP INTERRUPT (TYPE 1)

Generated after most instructions if the TF flag is set. Interrupts will not be generated after prefix instructions (e.g., REP), instructions which modify segment registers (e.g., POP DS), or the WAIT instruction.

NON-MASKABLE INTERRUPT—NMI (TYPE 2)

An external interrupt source which cannot be masked.

Table 4. 80186 Interrupt Vectors

Interrupt Name	Vector Type	Default Priority	Related Instructions
Divide Error Exception	0	*1	DIV, IDIV
Single Step Interrupt	1	12**2	All
NMI	2	1	All
Breakpoint Interrupt	3	*1	INT
INTO Detected Overflow Exception	4	*1	INTO
Array Bounds Exception	5	*1	BOUND
Unused-Opcode Exception	6	*1	Undefined Opcodes
ESC Opcode Exception	7	*1***	ESC Opcodes
Timer 0 Interrupt	8	2A****	
Timer 1 Interrupt	18	2B****	
Timer 2 Interrupt	19	2C****	
Reserved	9	3	
DMA 0 Interrupt	10	4	
DMA 1 Interrupt	11	5	
INT0 Interrupt	12	6	
INT1 Interrupt	13	7	
INT2 Interrupt	14	8	
INT3 Interrupt	15	9	

NOTES:
*1. These are generated as the result of an instruction execution.
**2. This is handled as in the 8086.
****3.All three timers constitute one source of request to the interrupt controller. The Timer interrupts all have the same default priority level with respect to all other interrupt sources. However, they have a defined priority ordering amongst themselves. (Priority 2A is higher priority than 2B.) Each Timer interrupt has a separate vector type number.
4. Default priorities for the interrupt sources are used only if the user does not program each source into a unique priority level.
***5. An escape opcode will cause a trap only if the proper bit is set in the peripheral control block relocation register.

BREAKPOINT INTERRUPT (TYPE 3)

A one-byte version of the INT instruction. It uses 12 as an index into the service routine address table (because it is a type 3 interrupt).

INTO DETECTED OVERFLOW EXCEPTION (TYPE4)

Generated during an INTO instruction if the OF bit is set.

ARRAY BOUNDS EXCEPTION (TYPE 5)

Generated during a BOUND instruction if the array index is outside the array bounds. The array bounds are located in memory at a location indicated by one of the instruction operands. The other operand indicates the value of the index to be checked.

UNUSED OPCODE EXCEPTION (TYPE 6)

Generated if execution is attempted on undefined opcodes.

ESCAPE OPCODE EXCEPTION (TYPE 7)

Generated if execution is attempted of ESC opcodes (D8H–DFH). This exception will only be generated if a bit in the relocation register is set. The return address of this exception will point to the ESC instruction causing the exception. If a segment override prefix preceded the ESC instruction, the return address will point to the segment override prefix.

Hardware-generated interrupts are divided into two groups: maskable interrupts and non-maskable interrupts. The 80186 provides maskable hardware interrupt request pins INT0–INT3. In addition, maskable interrupts may be generated by the 80186 integrated DMA controller and the integrated timer unit. The vector types for these interrupts is shown in Table 4. Software enables these inputs by setting the interrupt flag bit (IF) in the Status Word. The interrupt controller is discussed in the peripheral section of this data sheet.

Further maskable interrupts are disabled while servicing an interrupt because the IF bit is reset as part of the response to an interrupt or exception. The saved Status Word will reflect the enable status of the processor prior to the interrupt. The interrupt flag will remain zero unless specifically set. The interrupt return instruction restores the Status Word, thereby restoring the original status of IF bit. If the interrupt return re-enables interrupts, and another interrupt is pending, the 80186 will immediately service the highest-priority interrupt pending, i.e., no instructions of the main line program will be executed.

Non-Maskable Interrupt Request (NMI)

A non-maskable interrupt (NMI) is also provided. This interrupt is serviced regardless of the state of the IF bit. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed. The IF bit is cleared at the beginning of an NMI interrupt to prevent maskable interrupts from being serviced.

Single-Step Interrupt

The 80186 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single-step interrupt and is controlled by the single-step flag bit (TF) in the Status Word. Once this bit is set, an internal single-step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single-stepped.

Initialization and Processor Reset

Processor initialization or startup is accomplished by driving the $\overline{RES}$ input pin LOW. $\overline{RES}$ forces the 80186 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as $\overline{RES}$ is active. After $\overline{RES}$ becomes inactive and an internal processing interval elapses, the 80186 begins execution with the instruction at physical location FFFF0(H). $\overline{RES}$ also sets some registers to predefined values as shown in Table 5.

Table 5. 80186 Initial Register State after RESET

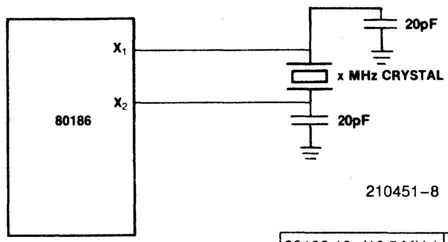
Status Word	F002(H)
Instruction Pointer	0000(H)
Code Segment	FFFF(H)
Data Segment	0000(H)
Extra Segment	0000(H)
Stack Segment	0000(H)
Relocation Register	20FF(H)
UMCS	FFFB(H)

80186 CLOCK GENERATOR

The 80186 provides an on-chip clock generator for both internal and external clock generation. The clock generator features a crystal oscillator, a divide-by-two counter, synchronous and asynchronous ready inputs, and reset circuitry.

Oscillator

The oscillator circuit of the 80186 is designed to be used with a parallel resonant fundamental mode crystal. This is used as the time base for the 80186. The crystal frequency selected will be double the CPU clock frequency. Use of an LC or RC circuit is not recommended with this oscillator. If an external oscillator is used, it can be connected directly to input pin X1 in lieu of a crystal. The output of the oscillator is not directly available outside the 80186. The recommended crystal configuration is shown in Figure 8.



		x
80186-12	(12.5 MHz)	25
80186-10	(10 MHz)	20
80186	(8 MHz)	16

Figure 8. Recommended 80186 Crystal Configuration

The following parameters may be used for choosing a crystal:

Temperature Range:	0 to 70°C
ESR (Equivalent Series Resistance):	30Ω max
C_0 (Shunt Capacitance of Crystal):	7.0 pf max
C_1 (Load Capacitance):	20 pf ± 2 pf
Drive Level:	1 mw max

Clock Generator

The 80186 clock generator provides the 50% duty cycle processor clock for the 80186. It does this by dividing the oscillator output by 2 forming the symmetrical clock. If an external oscillator is used, the state of the clock generator will change on the falling edge of the oscillator signal. The CLKOUT pin provides the processor clock signal for use outside the 80186. This may be used to drive other system components. All timings are referenced to the output clock.

READY Synchronization

The 80186 provides both synchronous and asynchronous ready inputs. Asynchronous ready synchronization is accomplished by circuitry which samples ARDY in the middle of T_2, T_3 and again in the middle of each T_W until ARDY is sampled HIGH. One-half CLKOUT cycle of resolution time is used. Full synchronization is performed only on the rising edge of ARDY, i.e., the falling edge of ARDY must be synchronized to the CLKOUT signal if it will occur during T_2, T_3, or T_W. High-to-LOW transitions of ARDY must be performed synchronously to the CPU clock.

A second ready input (SRDY) is provided to interface with externally synchronized ready signals. This input is sampled at the end of T_2, T_3 and again at the end of each T_W until it is sampled HIGH. By using this input rather than the asynchronous ready input, the half-clock cycle resolution time penalty is eliminated.

This input must satisfy set-up and hold times to guarantee proper operation of the circuit.

In addition, the 80186, as part of the integrated chip-select logic, has the capability to program WAIT states for memory and peripheral blocks. This is discussed in the Chip Select/Ready Logic description.

RESET Logic

The 80186 provides both a $\overline{RES}$ input pin and a synchronized RESET pin for use with other system components. The $\overline{RES}$ input pin on the 80186 is provided with hysteresis in order to facilitate power-on Reset generation via an RC network. RESET is guaranteed to remain active for at least five clocks given a $\overline{RES}$ input of at least six clocks. RESET may be delayed up to two and one-half clocks behind $\overline{RES}$.

Multiple 80186 processors may be synchronized through the $\overline{RES}$ input pin, since this input resets both the processor and divide-by-two internal counter in the clock generator. In order to insure that the divide-by-two counters all begin counting at the same time, the active going edge of $\overline{RES}$ must satisfy a 25 ns setup time before the falling edge of the 80186 clock input. In addition, in order to insure that all CPUs begin executing in the same clock cycle, the reset must satisfy a 25 ns setup time before the rising edge of the CLKOUT signal of all the processors.

LOCAL BUS CONTROLLER

The 80186 provides a local bus controller to generate the local bus control signals. In addition, it employs a HOLD/HLDA protocol for relinquishing the local bus to other bus masters. It also provides control lines that can be used to enable external buffers and to direct the flow of data on and off the local bus.

Memory/Peripheral Control

The 80186 provides ALE, $\overline{RD}$, and $\overline{WR}$ bus control signals. The $\overline{RD}$ and $\overline{WR}$ signals are used to strobe data from memory to the 80186 or to strobe data from the 80186 to memory. The ALE line provides a strobe to address latches for the multiplexed address/data bus. The 80186 local bus controller does not provide a memory/$\overline{I/O}$ signal. If this is required, the user will have to use the $\overline{S2}$ signal (which will require external latching), make the memory and I/O spaces nonoverlapping, or use only the integrated chip-select circuitry.

Transceiver Control

The 80186 generates two control signals to be connected to 8286/8287 transceiver chips. This capability allows the addition of transceivers for extra buffering without adding external logic. These control lines, DT/$\overline{R}$ and $\overline{DEN}$, are generated to control the flow of data through the transceivers. The operation of these signals is shown in Table 6.

Table 6. Transceiver Control Signals Description

Pin Name	Function
$\overline{DEN}$ (Data Enable)	Enables the output drivers of the transceivers. It is active LOW during memory, I/O, or INTA cycles.
DT/$\overline{R}$ (Data Transmit/ Receive)	Determines the direction of travel through the transceivers. A HIGH level directs data away from the processor during write operations, while a LOW level directs data toward the processor during a read operation.

Local Bus Arbitration

The 80186 uses a HOLD/HLDA system of local bus exchange. This provides an asynchronous bus exchange mechanism. This means multiple masters utilizing the same bus can operate at separate clock frequencies. The 80186 provides a single HOLD/HLDA pair through which all other bus masters may gain control of the local bus. This requires external circuitry to arbitrate which external device will gain control of the bus from the 80186 when there is more than one alternate local bus master. When the 80186 relinquishes control of the local bus, it floats $\overline{DEN}$, $\overline{RD}$, $\overline{WR}$, $\overline{S0}$–$\overline{S2}$, $\overline{LOCK}$, AD0–AD15, A16–A19, $\overline{BHE}$, and DT/$\overline{R}$ to allow another master to drive these lines directly.

The 80186 HOLD latency time, i.e., the time between HOLD request and HOLD acknowledge, is a function of the activity occurring in the processor when the HOLD request is received. A HOLD request is the highest-priority activity request which the processor may receive: higher than instruction fetching or internal DMA cycles. However, if a DMA cycle is in progress, the 80186 will complete the transfer before relinquishing the bus. This implies that if a HOLD request is received just as a DMA transfer begins, the HOLD latency time can be as great as 4 bus cycles. This will occur if a DMA word transfer operation is taking place from an odd address to an odd address. This is a total of 16 clocks or more, if WAIT states are required. In addition, if locked transfers are performed, the HOLD latency time will be increased by the length of the locked transfer.

Local Bus Controller and Reset

Upon receipt of a RESET pulse from the RES input, the local bus controller will perform the following action:

- Drive DEN, RD, and WR HIGH for one clock cycle, then float.

NOTE:
RD is also provided with an internal pull-up device to prevent the processor from inadvertently entering Queue Status mode during reset.

- Drive S0–S2 to the passive state (all HIGH) and then float.
- Drive LOCK HIGH and then float.
- TRISTATE AD0–15, A16–19, BHE, DT/R.
- Drive ALE LOW (ALE is never floated).
- Drive HLDA LOW.

INTERNAL PERIPHERAL INTERFACE

All the 80186 integrated peripherals are controlled via 16-bit registers contained within an internal 256-byte control block. This control block may be mapped into either memory or I/O space. Internal logic will recognize the address and respond to the bus cycle. During bus cycles to internal registers, the bus controller will signal the operation externally (i.e., the RD, WR, status, address, data, etc., lines will be driven as in a normal bus cycle), but D_{15-0}, SRDY, and ARDY will be ignored. The base address of the control block must be on an even 256-byte boundary (i.e., the lower 8 bits of the base address are all zeros). All of the defined registers within this control block may be read or written by the 80186 CPU at any time. The location of any register contained within the 256-byte control block is determined by the current base address of the control block.

The control block base address is programmed via a 16-bit relocation register contained within the control block at offset FEH from the base address of the control block (see Figure 9). It provides the upper 12 bits of the base address of the control block. The control block is effectively an internal chip select range and must abide by all the rules concerning chip selects (the chip select circuitry is discussed later in this data sheet). Any access to the 256 bytes of the control block activates an internal chip select. Other chip selects may overlap the control block only if they are programmed to zero wait states and ignore external ready. In addition, bit 12 of this register determines whether the control block will be mapped into I/O or memory space. If this bit is 1, the control block will be located in memory space, whereas if the bit is 0, the control block will be located in I/O space. If the control register block is mapped into I/O space, the upper 4 bits of the base address must be programmed as 0 (since I/O addresses are only 16 bits wide).

In addition to providing relocation information for the control block, the relocation register contains bits which place the interrupt controller into iRMX mode, and cause the CPU to interrupt upon encountering ESC instructions. At RESET, the relocation register is set to 20FFH. This causes the control block to start at FF00H in I/O space. An offset map of the 256-byte control register block is shown in Figure 10.

The integrated 80186 peripherals operate semi-autonomously from the CPU. Access to them for the most part is via software read/write of the control block. Most of these registers can be both read and written. A few dedicated lines, such as interrupts and DMA request provide real-time communication between the CPU and peripherals as in a more conventional system utilizing discrete peripheral blocks. The overall interaction and function of the peripheral blocks has not substantially changed.

CHIP-SELECT/READY GENERATION LOGIC

The 80186 contains logic which provides programmable chip-select generation for both memories and peripherals. In addition, it can be programmed to provide READY (or WAIT state) generation. It can also povide latched address bits A1 and A2. The chip-select lines are active for all memory and I/O cycles in their programmed areas, whether they be generated by the CPU or by the integrated DMA unit.

Memory Chip Selects

The 80186 provides 6 memory chip select outputs for 3 address areas; upper memory, lower memory, and midrange memory. One each is provided for upper memory and lower memory, while four are provided for midrange memory.

The range for each chip select is user-programmable and can be set to 2K, 4K, 8K, 16K, 32K, 64K, 128K (plus 1K and 256K for upper and lower chip selects). In addition, the beginning or base address

OFFSET: FEH

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ET	RMX	X	M/IO	Relocation Address Bits R19–R8											

ET = ESC Trap / No ESC Trap (1/0)
M/IO = Register block located in Memory / I/O Space (1/0)
RMX = Master Interrupt Controller mode /IRMX compatible Interrupt Controller(0/1)
Interrupt Controller mode (0/1)

Figure 9. Relocation Register

	OFFSET
Relocation Register	FEH
DMA Descriptors Channel 1	DAH D0H
DMA Descriptors Channel 0	CAH C0H
Chip-Select Control Registers	A8H A0H
Time 2 Control Registers	66H 60H
Time 1 Control Registers	5EH 58H
Time 0 Control Registers	56H 50H
Interrupt Controller Registers	3EH 20H

Figure 10. Internal Register Map

of the midrange memory chip select may also be selected. Only one chip select may be programmed to be active for any memory location at a time. All chip select sizes are in bytes, whereas 80186 memory is arranged in words. This means that if, for example, 16 64K x 1 memories are used, the memory block size will be 128K, not 64K.

Upper Memory $\overline{CS}$

The 80186 provides a chip select, called $\overline{UCS}$, for the top of memory. The top of memory is usually used as the system memory because after reset the 80186 begins executing at memory location FFFF0H.

The upper limit of memory defined by this chip select is always FFFFFH, while the lower limit is programmable. By programming the lower limit, the size of the select block is also defined. Table 7 shows the relationship between the base address selected and the size of the memory block obtained.

Table 7. UMCS Programming Values

Starting Address (Base Address)	Memory Block Size	UMCS Value (Assuming R0=R1=R2=0)
FFC00	1K	FFF8H
FF800	2K	FFB8H
FF000	4K	FF38H
FE000	8K	FE38H
FC000	16K	FC38H
F8000	32K	F838H
F0000	64K	F038H
E0000	128K	E038H
C0000	256K	C038H

The lower limit of this memory block is defined in the UMCS register (see Figure 11). This register is at offset A0H in the internal control block. The legal values for bits 6–13 and the resulting starting address and memory block sizes are given in Table 7. Any combination of bits 6–13 not shown in Table 7 will result in undefined operation. After reset, the UMCS register is programmed for a 1K area. It must be reprogrammed if a larger upper memory area is desired.

Any internally generated 20-bit address whose upper 16 bits are greater than or equal to UMCS (with bits 0–5 "0") will cause UCS to be activated. UMCS bits R2–R0 are used to specify READY mode for the area of memory defined by this chip-select register, as explained below.

Lower Memory $\overline{CS}$

The 80186 provides a chip select for low memory called $\overline{LCS}$. The bottom of memory contains the interrupt vector table, starting at location 00000H.

The lower limit of memory defined by this chip select is always 0H, while the upper limit is programmable. By programming the upper limit, the size of the memory block is also defined. Table 8 shows the relationship between the upper address selected and the size of the memory block obtained.

Table 8. LMCS Programming Values

Upper Address	Memory Block Size	LMCS Value (Assuming R0 = R1 = R2 = 0)
003FFH	1K	0038H
007FFH	2K	0078H
00FFFH	4K	00F8H
01FFFH	8K	01F8H
03FFFH	16K	03F8H
07FFFH	32K	07F8H
0FFFFH	64K	0FF8H
1FFFFH	128K	1FF8H
3FFFFH	256K	3FF8H

The upper limit of this memory block is defined in the LMCS register (see Figure 12). This register is at offset A2H in the internal control block. The legal values for bits 6–15 and the resulting upper address and memory block sizes are given in Table 8. Any combination of bits 6–15 not shown in Table 8 will result in undefined operation. After reset, the LMCS register value is undefined. However, the $\overline{\text{LCS}}$ chip-select line will not become active until the LMCS register is accessed.

Any internally generated 20-bit address whose upper 16 bits are less than or equal to LMCS (with bits 0–5 "1") will cause $\overline{\text{LCS}}$ to be active. LMCS register bits R2–R0 are used to specify the READY mode for the area of memory defined by this chip-select register.

Mid-Range Memory $\overline{\text{CS}}$

The 80186 provides four $\overline{\text{MCS}}$ lines which are active within a user-locatable memory block. This block can be located anywhere within the 80186 1M byte memory address space exclusive of the areas defined by $\overline{\text{UCS}}$ and $\overline{\text{LCS}}$. Both the base address and size of this memory block are programmable.

The size of the memory block defined by the mid-range select lines, as shown in Table 9, is determined by bits 8–14 of the MPCS register (see Figure 13). This register is at location A8H in the internal control block. One and only one of bits 8–14 must be set at a time. Unpredictable operation of the $\overline{\text{MCS}}$ lines will otherwise occur. Each of the four chip-select lines is active for one of the four equal contiguous divisions of the mid-range block. Thus, if the total block size is 32K, each chip select is active for 8K of memory with $\overline{\text{MCS0}}$ being active for the first range and $\overline{\text{MCS3}}$ being active for the last range.

The EX and MS in MPCS relate to peripheral functionally as described in a later section.

Table 9. MPCS Programming Values

Total Block Size	Individual Select Size	MPCS Bits 14–8
8K	2K	0000001B
16K	4K	0000010B
32K	8K	0000100B
64K	16K	0001000B
128K	32K	0010000B
256K	64K	0100000B
512K	128K	1000000B

The base address of the mid-range memory block is defined by bits 15–9 of the MMCS register (see Figure 14). This register is at offset A6H in the internal control block. These bits correspond to bits A19–A13 of the 20-bit memory address. Bits A12–A0 of the base address are always 0. The base address may be set at any integer multiple of the size of the total memory block selected. For example, if the mid-range block size is 32K (or the size of the block for which each $\overline{\text{MCS}}$ line is active is 8K), the block could be located at 10000H or 18000H, but not at 14000H, since the first few integer multiples of a 32K memory block are 0H, 8000H, 10000H, 18000H, etc. After reset, the contents of both of these registers is undefined. However, none of the $\overline{\text{MCS}}$ lines will be active until both the MMCS and MPCS registers are accessed.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A0H	1	1	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0
	A19								A11							

Figure 11. UMCS Register

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A2H	0	0	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0
	A19								A11							

Figure 12. LMCS Register

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A8H	1	M6	M5	M4	M3	M2	M1	M0	EX	MS	1	1	1	R2	R1	R0

Figure 13. MPCS Register

	15						9						3			0
OFFSET: A6H	U	U	U	U	U	U	U	1	1	1	1	1	1	R2	R1	R0
	A19						A13									

Figure 14. MMCS Register

MMCS bits R2–R0 specify READY mode of operation for all mid-range chip selects. All devices in mid-range memory must use the same number of WAIT states.

The 512K block size for the mid-range memory chip selects is a special case. When using 512K, the base address would have to be at either locations 00000H or 80000H. If it were to be programmed at 00000H when the $\overline{\text{LCS}}$ line was programmed, there would be an internal conflict between the $\overline{\text{LCS}}$ ready generation logic and the $\overline{\text{MCS}}$ ready generation logic. Likewise, if the base address were programmed at 80000H, there would be a conflict with the $\overline{\text{UCS}}$ ready generation logic. Since the $\overline{\text{LCS}}$ chip-select line does not become active until programmed, while the $\overline{\text{UCS}}$ line is active at reset, the memory base can be set only at 00000H. If this base address is selected, however, the $\overline{\text{LCS}}$ range must not be programmed.

Peripheral Chip Selects

The 80186 can generate chip selects for up to seven peripheral devices. These chip selects are active for seven contiguous blocks of 128 bytes above a programmable base address. This base address may be located in either memory or I/O space.

Seven $\overline{\text{CS}}$ lines called $\overline{\text{PCS0}}$–6 are generated by the 80186. The base address is user-programmable; however it can only be a multiple of 1K bytes, i.e., the least significant 10 bits of the starting address are always 0.

$\overline{\text{PCS5}}$ and $\overline{\text{PCS6}}$ can also be programmed to provide latched address bits A1, A2. If so programmed, they cannot be used as peripheral selects. These outputs can be connected directly to the A0, A1 pins used for selecting internal registers of 8-bit peripheral chips. This scheme simplifies the hardware interface because the 8-bit registers of peripherals are simply treated as 16-bit registers located on even boundaries in I/O space or memory space where only the lower 8-bits of the register are significant: the upper 8-bits are "don't cares."

The starting address of the peripheral chip-select block is defined by the PACS register (see Figure 15). This register is located at offset A4H in the internal control block. Bits 15–6 of this register correspond to bits 19–10 of the 20-bit Programmable Base Address (PBA) of the peripheral chip-select block. Bits 9–0 of the PBA of the peripheral chip-select block are all zeros. If the chip-select block is located in I/O space, bits 12–15 must be programmed zero, since the I/O address is only 16 bits wide. Table 10 shows the address range of each peripheral chip select with respect to the PBA contained in PACS register.

	15									6	5		3			0
OFFSET: A4H	U	U	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0
	A19									A10						

Figure 15. PACS Register

The user should program bits 15–6 to correspond to the desired peripheral base location. PACS bits 0–2 are used to specify READY mode for PSC0–PCS3.

Table 10. PCS Address Ranges

PCS Line	Active between Locations
PCS0	PBA —PBA+127
PCS1	PBA+128—PBA+255
PCS2	PBA+256—PBA+383
PCS3	PBA+384—PBA+511
PCS4	PBA+512—PBA+639
PCS5	PBA+640—PBA+767
PCS6	PBA+768—PBA+895

The mode of operation of the peripheral chip selects is defined by the MPCS register (which is also used to set the size of the mid-range memory chip-select block, see Figure 16). This register is located at offset A8H in the internal control block. Bit 7 is used to select the function of PCS5 and PCS6, while bit 6 is used to select whether the peripheral chip selects are mapped into memory or I/O space. Table 11 describes the programming of these bits. After reset, the contents of both the MPCS and the PACS registers are undefined, however none of the PCS lines will be active until both of the MPCS and PACS registers are accessed.

Table 11. MS, EX Programming Values

Bit	Description
MS	1 = Peripherals mapped into memory space.
	0 = Peripherals mapped into I/O space.
EX	0 = 5 PCS lines. A1, A2 provided.
	1 = 7 PCS lines. A1, A2 are not provided.

MPCS bits 0–2 are used to specify READY mode for PCS4–PCS6 as outlined below.

READY Generation Logic

The 80186 can generate a "READY" signal internally for each of the memory or peripheral CS lines. The number of WAIT states to be inserted for each peripheral or memory is programmable to provide 0–3 wait states for all accesses to the area for which the chip select is active. In addition, the 80186 may be programmed to either ignore external READY for each chip-select range individually or to factor external READY with the integrated ready generator.

READY control consists of 3 bits for each CS line or group of lines generated by the 80186. The interpretation of the ready bits is shown in Table 12.

Table 12. READY Bits Programming

R2	R1	R0	Number of WAIT States Generated
0	0	0	0 wait states, external RDY also used.
0	0	1	1 wait state inserted, external RDY also used.
0	1	0	2 wait states inserted, external RDY also used.
0	1	1	3 wait states inserted, external RDY also used.
1	0	0	0 wait states, external RDY ignored.
1	0	1	1 wait state inserted, external RDY ignored.
1	1	0	2 wait states inserted, external RDY ignored.
1	1	1	3 wait states inserted, external RDY ignored.

The internal ready generator operates in parallel with external READY, not in series if the external READY is used (R2 = 0). This means, for example, if the internal generator is set to insert two wait states, but activity on the external READY lines will insert four wait states, the processor will only insert four wait states, not six. This is because the two wait states generated by the internal generator overlapped the first two wait states generated by the external ready signal. Note that the external ARDY and SRDY lines are always ignored during cycles accessing internal peripherals.

R2–R0 of each control word specifies the READY mode for the corresponding block, with the exception of the peripheral chip selects: R2–R0 of PACS set the PCS0–3 READY mode, R2–R0 of MPCS set the PCS4–6 READY mode.

Chip Select/Ready Logic and Reset

Upon reset, the Chip-Select/Ready Logic will perform the following actions:

- All chip-select outputs will be driven HIGH.
- Upon leaving RESET, the UCS line will be programmed to provide chip selects to a 1K block with the accompanying READY control bits set at 011 to allow the maximum number of internal wait states in conjunction with external Ready consideration (i.e., UMCS resets to FFFBH).
- No other chip select or READY control registers have any predefined values after RESET. They will not become active until the CPU accesses their control registers. Both the PACS and MPCS registers must be accessed before the PCS lines will become active.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A8H	1	M6	M5	M4	M3	M2	M1	M0	EX	MS	1	1	1	R2	R1	R0

Figure 16. MPCS Register

DMA CHANNELS

The 80186 DMA controller provides two independent high-speed DMA channels. Data transfers can occur between memory and I/O spaces (e.g., Memory to I/O) or within the same space (e.g., Memory to Memory or I/O to I/O). Data can be transferred either in bytes (8 bits) or in words (16 bits) to or from even or odd addresses. Each DMA channel maintains both a 20-bit source and destination pointer which can be optionally incremented or decremented after each data transfer (by one or two depending on byte or word transfers). Each data transfer consumes 2 bus cycles (a minimum of 8 clocks), one cycle to fetch data and the other to store data. This provides a maximum data transfer rate of one Mword/sec or 2 MBytes/sec.

DMA Operation

Each channel has six registers in the control block which define each channel's specific operation. The control registers consist of a 20-bit Source pointer (2 words), a 20-bit destination pointer (2 words), a 16-bit Transfer Counter, and a 16-bit Control Word. The format of the DMA Control Blocks is shown in Table 13. The Transfer Count Register (TC) specifies the number of DMA transfers to be performed. Up to 64K byte or word transfers can be performed with automatic termination. The Control Word defines the channel's operation (see Figure 18). All registers may be modified or altered during any DMA activity. Any changes made to these registers will be reflected immediately in DMA operation.

Table 13. DMA Control Block Format

Register Name	Register Address	
	Ch. 0	Ch. 1
Control Word	CAH	DAH
Transfer Count	C8H	D8H
Destination Pointer (upper 4 bits)	C6H	D6H
Destination Pointer	C4H	D4H
Source Pointer (upper 4 bits)	C2H	D2H
Source Pointer	C0H	D0H

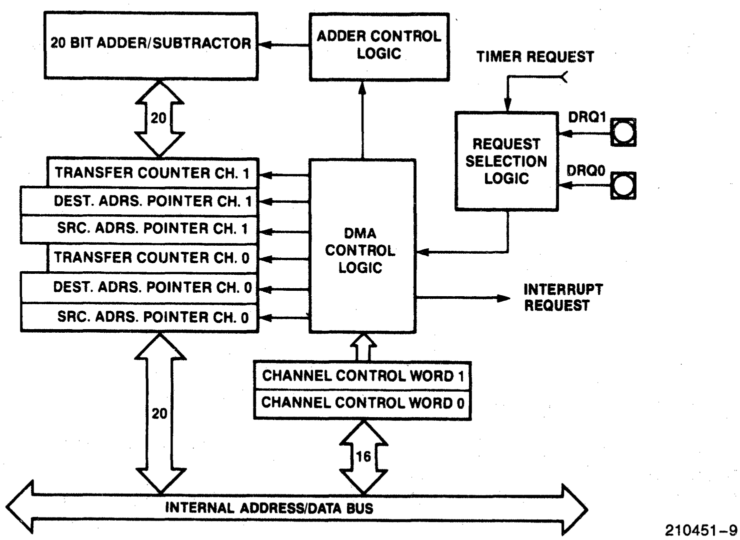


Figure 17. DMA Unit Block Diagram

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M/$\overline{IO}$	DESTINATION DEC	INC	M/$\overline{IO}$	SOURCE DEC	INC	TC	INT	SYN		P	TDQ	X	CHG/$\overline{NOCHG}$	ST/$\overline{STOP}$	$\overline{B}$/W

X = DON'T CARE.

Figure 18. DMA Control Register

DMA Channel Control Word Register

Each DMA Channel Control Word determines the mode of operation for the particular 81086 DMA channel. This register specifies:

- the mode of synchronization;
- whether bytes or words will be transferred;
- whether interrupts will be generated after the last transfer;
- whether DMA activity will cease after a programmed number of DMA cycles;
- the relative priority of the DMA channel with respect to the other DMA channel;
- whether the source pointer will be incremented, decremented, or maintained constant after each transfer;
- whether the source pointer addresses memory or I/O space;
- whether the destination pointer will be incremented, decremented, or maintained constant after each transfer; and
- whether the destination pointer will address memory or I/O space.

The DMA channel control registers may be changed while the channel is operating. However, any changes made during operation will affect the current DMA transfer.

DMA Control Word Bit Descriptions

$\overline{B}$/W: Byte/Word (0/1) Transfers.

ST/$\overline{STOP}$: Start/stop (1/0) Channel.

CHG/$\overline{NOCHG}$: Change/Do not change (1/0) ST/$\overline{STOP}$ bit. If this bit is set when writing to the control word, the ST/$\overline{STOP}$ bit will be programmed by the write to the control word. If this bit is cleared when writing the control word, the ST/$\overline{STOP}$ bit will not be altered. This bit is not stored; it will always be a 0 on read.

INT: Enable Interrupts to CPU on Transfer Count termination.

TC: If set, DMA will terminate when the contents of the Transfer Count register reach zero. The ST/$\overline{STOP}$ bit will also be reset at this point if TC is set. If this bit is cleared, the DMA unit will decrement the transfer count register for each DMA cycle, but the DMA transfer will not stop when the contents of the TC register reach zero.

SYN (2 bits)
- 00 No synchronization.

  NOTE: The ST bit will be cleared automatically when the contents of the TC register reach zero regardless of the state of the TC bit.
- 01 Source synchronization.
- 10 Destination synchronization.
- 11 Unused.

SOURCE:
- INC Increment source pointer by 1 or 2 (depends on $\overline{B}$/W) after each transfer.
- M/$\overline{IO}$ Source pointer is in M/IO space (1/0).
- DEC Decrement source pointer by 1 or 2 (depends on $\overline{B}$/W) after each transfer.

DEST:
- INC Increment destination pointer by 1 or 2 ($\overline{B}$/W) after each transfer.
- M/$\overline{IO}$ Destination pointer is in M/IO space (1/0).
- DEC Decrement destination pointer by 1 or 2 (depending on $\overline{B}$/W) after each transfer.

P Channel priority—relative to other channel.
- 0 low priority.
- 1 high priority.

Channels will alternate cycles if both set at same priority level.

TDRQ
- 0: Disable DMA requests from timer 2.
- 1: Enable DMA requests from timer 2.

Bit 3 Bit 3 is not used.

If both INC and DEC are specified for the same pointer, the pointer will remain constant after each cycle.

DMA Destination and Source Pointer Registers

Each DMA channel maintains a 20-bit source and a 20-bit destination pointer. Each of these pointers takes up two full 16-bit registers in the peripheral control block. The lower four bits of the upper register contain the upper four bits of the 20-bit physical address (see Figure 18a). These pointers may be individually incremented or decremented after each transfer. If word transfers are performed the pointer is incremented or decremented by two. Each pointer may point into either memory or I/O space. Since the DMA channels can perform transfers to or from odd addresses, there is no restriction on values for the pointer registers. Higher transfer rates can be obtained if all word transfers are performed to even addresses, since this will allow data to be accessed in a single memory access.

DMA Transfer Count Register

Each DMA channel maintains a 16-bit transfer count register (TC). This register is decremented after every DMA cycle, regardless of the state of the TC bit in the DMA Control Register. If the TC bit in the DMA control word is set or unsynchronized transfers are programmed, however, DMA activity will terminate when the transfer count register reaches zero.

DMA Requests

Data transfers may be either source or destination synchronized, that is either the source of the data or the destination of the data may request the data transfer. In addition, DMA transfers may be unsynchronized; that is, the transfer will take place continually until the correct number of transfers has occurred. When source or unsynchronized transfers are performed, the DMA channel may begin another transfer immediately after the end of a previous DMA transfer. This allows a complete transfer to take place every 2 bus cycles or eight clock cycles (assuming no wait states). No prefetching occurs when destination synchronization is performed, however. Data will not be fetched from the source address until the destination device signals that it is ready to receive it. When destination synchronized transfers are requested, the DMA controller will relinquish control of the bus after every transfer. If no other bus activity is initiated, another DMA cycle will begin after two processor clocks. This is done to allow the destination device time to remove its request if another transfer is not desired. Since the DMA controller will relinquish the bus, the CPU can initiate a bus cycle. As a result, a complete bus cycle will often be inserted between destination synchronized transfers. These lead to the maximum DMA transfer rates shown in Table 14.

Table 14. Maximum DMA Transfer Rates

Type of Synchronization Selected	CPU Running	CPU Halted
Unsynchronized	2MBytes/sec	2MBytes/sec
Source Synch	2MBytes/sec	2MBytes/sec
Destination Synch	1.3MBytes/sec	1.5MByte/sec

	15			0
HIGHER REGISTER ADDRESS	XXX	XXX	XXX	A19–A16
LOWER REGISTER ADDRESS	A15–A12	A11–A8	A7–A4	A3–A0

XXX = DON'T CARE

Figure 18a. DMA Memory Pointer Register Format

DMA Acknowledge

No explicit DMA acknowledge pulse is provided. Since both source and destination pointers are maintained, a read from a requesting source, or a write to a requesting destination, should be used as the DMA acknowledge signal. Since the chip-select lines can be programmed to be active for a given block of memory or I/O space, and the DMA pointers can be programmed to point to the same given block, a chip-select line could be used to indicate a DMA acknowledge.

DMA Priority

The DMA channels may be programmed such that one channel is always given priority over the other, or they may be programmed such as to alternate cycles when both have DMA requests pending. DMA cycles always have priority over internal CPU cycles except between locked memory accesses or word accesses the odd memory locations; however, an external bus hold takes priority over an internal DMA cycle. Because an interrupt request cannot suspend a DMA operation and the CPU cannot access memory during a DMA cycle, interrupt latency time will suffer during sequences of continuous DMA cycles. An NMI request, however, will cause all internal DMA activity to halt. This allows the CPU to quickly respond to the NMI request.

DMA Programming

DMA cycles will occur whenever the ST/STOP bit of the Control Register is set. If synchronized transfers are programmed, a DRQ must also have been generated. Therefore the source and destination transfer pointers, and the transfer count register (if used) must be programmed before this bit is set.

Each DMA register may be modified while the channel is operating. If the CHG/NOCHG bit is cleared when the control register is written, the ST/STOP bit of the control register will not be modified by the write. If multiple channel registers are modified, it is recommended that a LOCKED string transfer be used to prevent a DMA transfer from occurring between updates to the channel registers.

DMA Channels and Reset

Upon RESET, the DMA channels will perform the following actions:

- The Start/Stop bit for each channel will be reset to STOP.
- Any transfer in progress is aborted.

TIMERS

The 80186 provides three internal 16-bit programmable timers (see Figure 19). Two of these are highly flexible and are connected to four external pins (2 per timer). They can be used to count external events, time external events, generate nonrepetitive waveforms, etc. The third timer is not connected to any external pins, and is useful for real-time coding and time delay applications. In addition, this third timer can be used as a prescaler to the other two, or as a DMA request source.

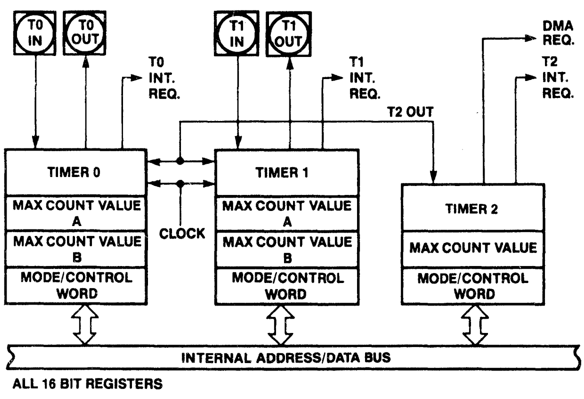


Figure 19. Timer Block Diagram

Timer Operation

The timers are controlled by 11 16-bit registers in the internal peripheral control block. The configuration of these registers is shown in Table 15. The count register contains the current value of the timer. It can be read or written at any time independent of whether the timer is running or not. The value of this register will be incremented for each timer event. Each of the timers is equipped with a MAX COUNT register, which defines the maximum count the timer will reach. After reaching the MAX COUNT register value, the timer count value will reset to zero during that same clock, i.e., the maximum count value is never stored in the count register itself. Timers 0 and 1 are, in addition, equipped with a second MAX COUNT register, which enables the timers to alternate their count between two different MAX COUNT values programmed by the user. If a single MAX COUNT register is used, the timer output pin will switch LOW for a single clock, 1 clock after the maximum count value has been reached. In the dual MAX COUNT register mode, the output pin will indicate which MAX COUNT register is currently in use, thus allowing nearly complete freedom in selecting waveform duty cycles. For the timers with two MAX COUNT registers, the RIU bit in the control register determines which is used for the comparison.

Each timer gets serviced every fourth CPU-clock cycle, and thus can operate at speeds up to one-quarter the internal clock frequency (one-eighth the crystal rate). External clocking of the timers may be done at up to a rate of one-quarter of the internal CPU-clock rate (2 MHz for an 8 MHz CPU clock). Due to internal synchronization and pipelining of the timer circuitry, a timer output may take up to 6 clocks to respond to any individual clock or gate input.

Since the count registers and the maximum count registers are all 16 bits wide, 16 bits of resolution are provided. Any Read or Write access to the timers will add one wait state to the minimum four-clock bus cycle, however. This is needed to synchronize and coordinate the internal data flows between the internal timers and the internal bus.

The timers have several programmable options.

- All three timers can be set to halt or continue on a terminal count.
- Timers 0 and 1 can select between internal and external clocks, alternate between MAX COUNT registers and be set to retrigger on external events.
- The timers may be programmed to cause an interrupt on terminal count.

These options are selectable via the timer mode/control word.

Timer Mode/Control Register

The mode/control register (see Figure 20) allows the user to program the specific mode of operation or check the current programmed status for any of the three integrated timers.

Table 15. Timer Control Block Format

Register Name	Register Offset		
	Tmr. 0	**Tmr. 1**	**Tmr. 2**
Mode/Control Word	56H	5EH	66H
Max Count B	54H	5CH	not present
Max Count A	52H	5AH	62H
Count Register	50H	58H	60H

15	14	13	12	11		5	4	3	2	1	0
EN	$\overline{INH}$	INT	RIU	0	...	MC	RTG	P	EXT	ALT	CONT

Figure 20. Timer Mode/Control Register

ALT:

The ALT bit determines which of two MAX COUNT registers is used for count comparison. If ALT = 0, register A for that timer is always used, while if ALT = 1, the comparison will alternate between register A and register B when each maximum count is reached. This alternation allows the user to change one MAX COUNT register while the other is being used, and thus provides a method of generating non-repetitive waveforms. Square waves and pulse outputs of any duty cycle are a subset of available signals obtained by not changing the final count registers. The ALT bit also determines the function of the timer output pin. If ALT is zero, the output pin will go LOW for one clock, the clock after the maximum count is reached. If ALT is one, the output pin will reflect the current MAX COUNT register being used (0/1 for B/A).

CONT:

Setting the CONT bit causes the associated timer to run continuously, while resetting it causes the timer to halt upon maximum count. If COUNT = 0 and ALT = 1, the timer will count to the MAX COUNT register A value, reset, count to the register B value, reset, and halt.

EXT:

The external bit selects between internal and external clocking for the timer. The external signal may be asynchronous with respect to the 80186 clock. If this bit is set, the timer will count LOW-to-HIGH transitions on the input pin. If cleared, it will count an internal clock while using the input pin for control. In this mode, the function of the external pin is defined by the RTG bit. The maximum input to output transition latency time may be as much as 6 clocks. However, clock inputs may be pipelined as closely together as every 4 clocks without losing clock pulses.

P:

The prescaler bit is ignored unless internal clocking has been selected (EXT = 0). If the P bit is a zero, the timer will count at one-fourth the internal CPU clock rate. If the P bit is a one, the output of timer 2 will be used as a clock for the timer. Note that the user must initialize and start timer 2 to obtain the prescaled clock.

RTG:

Retrigger bit is only active for internal clocking (EXT = 0). In this case it determines the control function provided by the input pin.

If RTG = 0, the input level gates the internal clock on and off. If the input pin is HIGH, the timer will count; if the input pin is LOW, the timer will hold its value. As indicated previously, the input signal may be asynchronous with respect to the 80186 clock.

When RTG = 1, the input pin detects LOW-to-HIGH transitions. The first such transition starts the timer running, clearing the timer value to zero on the first clock, and then incrementing thereafter. Further transitions on the input pin will again reset the timer to zero, from which it will start counting up again. If CONT = 0, when the timer has reached maximum count, the EN bit will be cleared, inhibiting further timer activity.

EN:

The enable bit provides programmer control over the timer's RUN/HALT status. When set, the timer is enabled to increment subject to the input pin constraints in the internal clock mode (discussed previously). When cleared, the timer will be inhibited from counting. All input pin transistions during the time EN is zero will be ignored. If CONT is zero, the EN bit is automatically cleared upon maximum count.

INH:

The inhibit bit allows for selective updating of the enable (EN) bit. If INH is a one during the write to the mode/control word, then the state of the EN bit will be modified by the write. If INH is a zero during the write, the EN bit will be unaffected by the operation.This bit is not stored; it will always be a 0 on a read.

INT:

When set, the INT bit enables interrupts from the timer, which will be generated on every terminal count. If the timer is configured in dual MAX COUNT register mode, an interrupt will be generated each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. If this enable bit is cleared after the interrupt request has been generated, but before a pending interrupt is serviced, the interrupt request will still be in force. (The request is latched in the Interrupt Controller).

MC:

The Maximum Count bit is set whenever the timer reaches its final maximum count value. If the timer is configured in dual MAX COUNT register mode, this bit will be set each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. This bit is set regardless of the timer's interrupt-enable bit. The MC bit gives the user the ability to monitor timer status through software instead of through interrupts.

Programmer intervention is required to clear this bit.

RIU:

The Register In Use bit indicates which MAX COUNT register is currently being used for comparison to the timer count value. A zero value indicates register A. The RIU bit cannot be written, i.e., its value is not affected when the control register is written. It is always cleared when the ALT bit is zero.

Not all mode bits are provided for timer 2. Certain bits are hardwired as indicated below:

ALT = 0, EXT = 0, P = 0, RTG = 0, RIU = 0

Count Registers

Each of the three timers has a 16-bit count register. The current contents of this register may be read or written by the processor at any time. If the register is written into while the timer is counting,the new value will take effect in the current count cycle.

Max Count Registers

Timers 0 and 1 have two MAX COUNT registers, while timer 2 has a single MAX COUNT register. These contain the number of events the timer will count. In timers 0 and 1, the MAX COUNT register used can alternate between the two max count values whenever the current maximum count is reached. The condition which causes a timer to reset is equivalent between the current count value and the max count being used. This means that if the count is changed to be above the max count value, or if the max count value is changed to be below the current value, the timer will not reset to zero, but rather will count to its maximum value, "wrap around" to zero, then count until the max count is reached.

Timers and Reset

Upon RESET, the Timers will perform the following actions:

- All EN (Enable) bits are reset preventing timer counting.
- All SEL (Select) bits are reset to zero. This selects MAX COUNT register A, resulting in the Timer Out pins going HIGH upon RESET.

INTERRUPT CONTROLLER

The 80186 can receive interrupts from a number of sources, both internal and external. The internal interrupt controller serves to merge these requests on a priority basis, for individual service by the CPU.

Internal interrupt sources (Timers and DMA channels) can be disabled by their own control registers or by mask bits within the interrupt controller. The 80186 interrupt controller has its own control register that set the mode of operation for the controller.

The interrupt controller will resolve priority among requests that are pending simultaneously. Nesting is provided so interrupt service routines for lower priority interrupts may themselves be interrupted by higher priority interrupts. A block diagram of the interrupt controller is shown in Figure 21.

The interrupt controller has a special iRMX 86 compatibility mode that allows the use of the 80186 within the iRMX 86 operating system interrupt structure. The conroller is set in this mode by setting bit 14 in the peripheral control block relocation register (see iRMX 86 Compatibility Mode section). In this mode, the internal 80186 interrupt controller functions as a "slave" controller to an external "master" controller. Special initialization software must be included to properly set up the 80186 interrupt controller in iRMX 86 mode.

MASTER MODE OPERATION

Interrupt Controller External Interface

For external interrupt sources, five dedicated pins are provided. One of these pins is dedicated to NMI, non-maskable interrupt. This is typically used for power-fail interrupts, etc. The other four pins may function either as four interrupt input lines with internally generated interrupt vectors, as an interrupt line and an interrupt acknowledge line (called the "cascade mode") along with two other input lines with internally generated interrupt vectors, or as two interrupt input lines and two dedicated interrupt acknowledge output lines. When the interrupt lines are configured in cascade mode, the 80186 interrupt controller will not generate internal interrupt vectors.

External sources in the cascade mode use externally generated interrupt vectors. When an interrupt is acknowledged, two $\overline{\text{INTA}}$ cycles are initiated and the vector is read into the 80186 on the second cycle. The capability to interface to external 8259A programmable interrupt controllers is thus provided when the inputs are configured in cascade mode.

Interrupt Controller Modes of Operation

The basic modes of operation of the interrupt controller in master mode are similar to the 8259A. The interrupt controller responds indentically to internal interrupts in all three modes: the difference is only in the interpretation of function of the four external interrupt pins. The interrupt controller is set into one of these three modes by programming the correct bits in the INT0 and INT1 control registers. The modes of interrupt controller operation are as follows:

Fully Nested Mode

When in the fully nested mode four pins are used as direct interrupt requests. The vectors for these four inputs are generated internally. An in-service bit is provided for every interrupt source. If a lower-priority device requests an interrupt while the in service bit (IS) is set, no interrupt will be generated by the interrupt controller. In addition, if another interrupt request occurs from the same interrupt source while the in-service bit is set, no interrupt will be generated by the interrupt controller. This allows interrupt service routines to operate with interrupts enabled without being themselves interrupted by lower-priority interrupts. Since interrupts are enabled, higher-priority interrupts will be serviced.

When a service routine is completed, the proper IS bit must be reset by writing the proper pattern to the EOI register. This is required to allow subsequent interrupts from this interrupt source and to allow servicing of lower-priority interrupts. An EOI command is issued at the end of the service routine just before the issuance of the return from interrupt instruction. If the fully nested structure has been upheld, the next highest-priority source with its IS bit set is then serviced.

Cascade Mode

The 80186 has four interrupt pins and two of them have dual functions. In the fully nested mode the four pins are used as direct interrupt inputs and the corresponding vectors are generated internally. In the cascade mode, the four pins are configured into interrupt input-dedicated acknowledge signal pairs. The interconnection is shown in Figure 22. INT0 is an interrupt input interfaced to an 8259A, while INT2/$\overline{\text{INTA0}}$ serves as the dedicated interrupt acknowledge signal to that peripheral. The same is true for INT1 and INT3/$\overline{\text{INTA1}}$. Each pair can selectively be placed in the cascade or non-cascade mode by programming the proper value into INT0 and INT1 control registers. The use of the dedicated acknowledge signals eliminates the need for the use of external logic to generate $\overline{\text{INTA}}$ and device select signals.

The primary cascade mode allows the capability to serve up to 128 external interrupt sources through the use of external master and slave 8259As. Three levels of priority are created, requiring priority resolution in the 80186 interrupt controller, the master 8259As, and the slave 8259As. If an external interrupt is serviced, one IS bit is set at each of these levels. When the interrupt service routine is completed, up to three end-of-interrupt commands must be issued by the programmer.

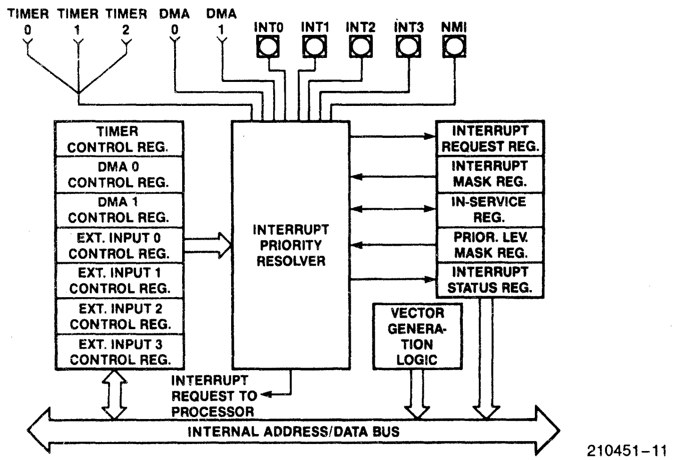


Figure 21. Interrupt Controller Block Diagram

Special Fully Nested Mode

This mode is entered by setting the SFNM bit in INT0 or INT1 control register. It enables complete nestability with external 8259A masters. Normally, an interrupt request from an interrupt source will not be recognized unless the in-service bit for that source is reset. If more than one interrupt source is connected to an external interrupt controller, all of the interrupts will be funneled through the same 80186 interrupt request pin. As a result, if the external interrupt controller receives a higher-priority interrupt, its interrupt will not be recognized by the 80186 controller until the 80186 in-service bit is reset. In special fully nested mode, the 80186 interrupt controller will allow interrupts from an external pin regardless of the state of the in-service bit for an interrupt source in order to allow multiple interrupts from a single pin. An in-service bit will continue to be set, however, to inhibit interrupts from other lower-priority 80186 interrupt sources.

Special procedures should be followed when resetting IS bits at the end of interrupt service routines. Software polling of the external master's IS register is required to determine if there is more than one bit set. If so, the IS bit in the 80186 remains active and the next interrupt service routine is entered.

Operation in a Polled Environment

The controller may be used in a polled mode if interrupts are undesirable. When polling, the processor disables interrupts and then polls the interrupt controller whenever it is convenient. Polling the interrupt controller is accomplished by reading the Poll Word (Figure 31). Bit 15 in the poll word indicates to the processor that an interrupt of high enough priority is requesting service. Bits 0–4 indicate to the processor the type vector of the highest-priority source requesting service. Reading the Poll Word causes the In-Service bit of the highest priority source to be set.

It is desirable to be able to read the Poll Word information without guaranteeing service of any pending interrupt, i.e., not set the indicated in-service bit. The 80186 provides a Poll Status Word in addition to the conventional Poll Word to allow this to be done. Poll Word information is duplicated in the Poll Status Word, but reading the Poll Status Word does not set the associated in-service bit. These words are located in two adjacent memory locations in the register file.

Master Mode Features

Programmable Priority

The user can program the interrupt sources into any of eight different priority levels. The programming is done by placing a 3-bit priority level (0–7) in the control register of each interrupt source. (A source with a priority level of 4 has higher priority over all priority levels from 5 to 7. Priority registers containing values lower than 4 have greater priority). All interrupt sources have preprogrammed default priority levels (see Table 4).

If two requests with the same programmed priority level are pending at once, the priority ordering scheme shown in Table 4 is used. If the serviced interrupt routine reenables interrupts, it allows other requests to be serviced.

End-of-Interrupt Command

The end-of-interrupt (EOI) command is used by the programmer to reset the In-Service (IS) bit when an interrupt service routine is completed. The EOI command is issued by writing the proper pattern to the EOI register. There are two types of EOI commands, specific and nonspecific. The nonspecific command does not specify which IS bit is reset. When issued, the interrupt controller automatically resets the IS bit of the highest priority source with an active service routine. A specific EOI command requires that the programmer send the interrupt vector type to the interrupt controller indicating which source's IS bit is to be reset. This command is used when the fully nested structure has been disturbed or the highest priority IS bit that was set does not belong to the service routine in progress.

Trigger Mode

The four external interrupt pins can be programmed in either edge- or level-trigger mode. The control register for each external source has a level-trigger mode (LTM) bit. All interrupt inputs are active HIGH. In the edge sense mode or the level-trigger mode, the interrupt request must remain active (HIGH) until the interrupt request is acknowledged by the 80186 CPU. In the edge-sense mode, if the level remains high after the interrupt is acknowledged, the input is disabled and no further requests will be generated. The input level must go LOW for at least one clock cycle to reenable the input. In the level-trigger mode, no such provision is made: holding the interrupt input HIGH will cause continuous interrupt requests.

Interrupt Vectoring

The 80186 Interrupt Controller will generate interrupt vectors for the integrated DMA channels and the integrated Timers. In addition, the Interrupt Controller will generate interrupt vectors for the external interrupt lines if they are not configured in Cascade or Special Fully Nested Mode. The interrupt vectors generated are fixed and cannot be changed (see Table 4).

Interrupt Controller Registers

The Interrupt Controller register model is shown in Figure 23. It contains 15 registers. All registers can both be read or written unless specified otherwise.

In-Service Register

This register can be read from or written into. The format is shown in Figure 24. It contains the In-Service bit for each of the interrupt sources. The In-Service bit is set to indicate that a source's service routine is in progress. When an In-Service bit is set, the interrupt controller will not generate interrupts to the CPU when it receives interrupt requests from devices with a lower programmed priority level. The TMR bit is the In-Service bit for all three timers; the D0 and D1 bits are the In-Service bits for the two DMA channels; the I0–I3 are the In-Service bits for the external interrupt pins. The IS bit is set when the processor acknowledges an interrupt request either by an interrupt acknowledge or by reading the poll register. The IS bit is reset at the end of the interrupt service routine by an end-of-interrupt command issued by the CPU.

Interrupt Request Register

The internal interrupt sources have interrupt request bits inside the interrupt controller. The format of this register is shown in Figure 24. A read from this register yields the status of these bits. The TMR bit is the logical OR of all timer interrupt requests. D0 and D1 are the interrupt request bits for the DMA channels.

The state of the external interrupt input pins is also indicated. The state of the external interrupt pins is not a stored condition inside the interrupt controller, therefore the external interrupt bits cannot be written. The external interrupt request bits show exactly when an interrupt request is given to the interrupt controller, so if edge-triggered mode is selected, the bit in the register will be HIGH only after an inactive-to-active transition. For internal interrupt sources, the register bits are set when a request arrives and are reset when the processor acknowledges the requests.

Mask Register

This is a 16-bit register that contains a mask bit for each interrupt source. The format for this register is shown in Figure 24. A one in a bit position corresponding to a particular source serves to mask the source from generating interrupts. These mask bits are the exact same bits which are used in the individual control registers; programming a mask bit using the mask register will also change this bit in the individual control registers, and vice versa.

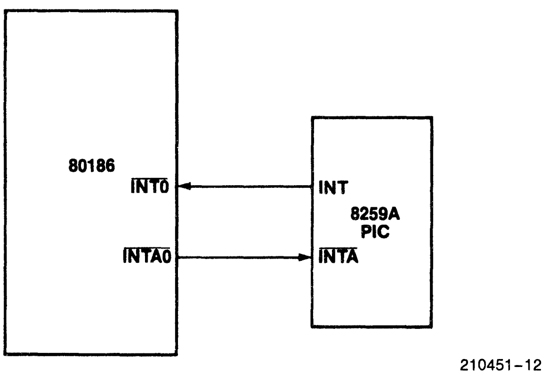


Figure 22. Cascade Mode Interrupt Connection

Register	OFFSET
INT3 CONTROL REGISTER	3EH
INT2 CONTROL REGISTER	3CH
INT1 CONTROL REGISTER	3AH
INT0 CONTROL REGISTER	38H
DMA 1 CONTROL REGISTER	36H
DMA 0 CONTROL REGISTER	34H
TIMER CONTROL REGISTER	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
POLL STATUS REGISTER	26H
POLL REGISTER	24H
EOI REGISTER	22H

Figure 23. Interrupt Controller Registers (Non-iRMX™ 86 Mode)

Priority Mask Register

This register is used to mask all interrupts below particular interrupt priority levels. The format of this register is shown in Figure 25. The code in the lower three bits of this register inhibits interrupts of priority lower (a higher priority number) than the code specified. For example, 100 written into this register masks interrupts of level five (101), six (110), and seven (111). The register is reset to seven (111) upon RESET so all interrupts are unmasked.

Interrupt Status Register

This register contains general interrupt controller status information. The format of this register is shown in Figure 26. The bits in the status register have the following functions:

DHLT: DMA Halt Transfer; setting this bit halts all DMA transfers. It is automatically set whenever a non-maskable interrupt occurs, and it is reset when an IRET instruction is executed. The purpose of this bit is to allow prompt service of all non-maskable interrupts. This bit may also be set by the CPU.

IRTx: These three bits represent the individual timer interrupt request bits. These bits are used to differentiate the timer interrupts, since the timer IR bit in the interrupt request register is the "OR" function of all timer interrupt request. Note that setting any one of these three bits initiates an interrupt request to the interrupt controller.

15	14				10	9	8	7	6	5	4	3	2	1	0
0	0	•	•	•	0	0	0	13	12	I1	I0	D1	D0	0	TMR

Figure 24. In-Service, Interrupt Request, and Mask Register Formats

15	14											3	2	1	0
0	0	•	•	•	•	•	•	•	•	•		0	PRM2	PRM1	PRM0

Figure 25. Priority Mask Register Format

15	14							7	6	5	4	3	2	1	0
DHLT	0	•	•	•	•	•		0	0	0	0	0	IRT2	IRT1	IRT0

Figure 26. Interrupt Status Register Format

Timer, DMA 0, 1; Control Register

These registers are the control words for all the internal interrupt sources. The format for these registers is shown in Figure 27. The three bit positions PR0, PR1, and PR2 represent the programmable priority level of the interrupt source. The MSK bit inhibits interrupt requests from the interrupt source. The MSK bits in the individual control registers are the exact same bits as are in the Mask Register; modifying them in the individual control registers will also modify them in the Mask Register, and vice versa.

INT0-INT3 Control Registers

These registers are the control words for the four external input pins. Figure 28 shows the format of the INT0 and INT1 Control registers; Figure 29 shows the format of the INT2 and INT3 Control registers. In cascade mode or special fully nested mode, the control words for INT2 and INT3 are not used.

The bits in the various control registers are encoded as follows:

PRO-2: Priority programming information. Highest Priority = 000, Lowest Priority = 111

LTM: Level-trigger mode bit. 1 = level-triggered; 0 = edge-triggered. Interrupt Input levels are active high. In level-triggered mode, an interrupt is generated whenever the external line is high. In edge-triggered mode, an interrupt will be generated only when this level is proceded by an inactive-to-active transition on the line. In both cases, the level must remain active until the interrupt is acknowledged.

MSK: Mask bit, 1 = mask; 0 = non-mask.

C: Cascade mode bit, 1 = cascade; 0 = direct

SFNM: Special fully nested mode bit, 1 = SFNM

EOI Register

The end of the interrupt register is a command register which can only be written into. The format of this register is shown in Figure 30. It initiates an EOI command when written to by the 80186 CPU.

The bits in the EOI register are encoded as follows:

S_X: Encoded information that specifies an interrupt source vector type as shown in Table 4. For example, to reset the In-Service bit for DMA channel 0, these bits should be set to 01010, since the vector type for DMA channel 0 is 10. Note that to reset the single In-Service bit for any of the three timers, the vector type for timer 0 (8) should be written in this register.

15	14									4	3	2	1	0
0	0	•	•	•	•	•	•	•	•	0	MSK	PR2	PR1	PR0

Figure 27. Timer/DMA Control Registers Formats

15	14						7	6	5	4	3	2	1	0
0	0	•	•	•	•	•	0	SFNM	C	LTM	MSK	PR2	PR1	PR0

Figure 28. INT0/INT1 Control Register Formats

15	14								5	4	3	2	1	0
0	0	•	•	•	•	•	•	•	0	LTM	MSK	PR2	PR1	PR0

Figure 29. INT2/INT3 Control Register Formats

NSPEC/: A bit that determines the type of EOI com-
SPEC mand. Nonspecific = 1, Specific = 0.

Poll and Poll Status Registers

These registers contain polling information. The format of these registers is shown in Figure 31. They can only be read. Reading the Poll register constitutes a software poll. This will set the IS bit of the highest priority pending interrupt. Reading the poll status register will not set the IS bit of the highest priority pending interrupt; only the status of pending interrupts will be provided.

Encoding of the Poll and Poll Status register bits are as follows:

S_X: Encoded information that indicates the vector type of the highest priority interrupting source. Valid only when INTREQ = 1.

INTREQ: This bit determines if an interrupt request is present. Interrupt Request = 1; no Interrupt Request = 0.

iRMX™ 86 COMPATIBILITY MODE

This mode allows iRMX 86-80186 compatibility. The interrupt model of iRMX 86 requires one master and multiple slave 8259As in cascaded fashion. When iRMX mode is used, the internal 80186 interrupt controller will be used as a slave controller to an external master interrupt controller. The internal 80186 resources will be monitored through the internal interrupt controller, while the external controller functions as the system master interrupt controller.

Upon reset, the 80186 interrupt controller will be in the non-iRMX 86 mode of operation. To set the controller in the iRMX 86 mode, bit 14 of the Relocation Register should be set.

Because of pin limitations caused by the need to interface to an external 8259A master, the internal interrupt controller will no longer accept external inputs. There are however, enough 80186 interrupt controller inputs (internally) to dedicate one to each timer. In this mode, each timer interrupt source has its own mask bit, IS bit, and control word.

The iRMX 86 operating system requires peripherals to be assigned fixed priority levels. This is incompatible with the normal operation of the 80186 interrupt controller. Therefore, the initialization software must program the proper priority levels for each source. The required priority levels for the internal interrupt sources in iRMX mode are shown in Table 16.

Table 16. Internal Source Priority Level

Priority Level	Interrupt Source
0	Timer 0
1	(reserved)
2	DMA 0
3	DMA 1
4	Timer 1
5	Timer 2

These level assignments must remain fixed in the iRMX 86 mode of operation.

iRMX™ 86 Mode External Interface

The configuration of the 80186 with respect to an external 8259A master is shown in Figure 32. The INT0 input is used as the 80186 CPU interrupt input. INT3 functions as an output to send the 80186 slave-interrupt-request to one of the 8 master-PIC-inputs.

15	14	13								5	4	3	2	1	0
SPEC/ NSPEC	0	0	•	•	•	•	•	•	•	0	S4	S3	S2	S1	S0

Figure 30. EOI Register Format

15	14	13								5	4	3	2	1	0
INT REQ	0	0	•	•	•	•	•	•	•	0	S4	S3	S2	S1	S0

Figure 31. Poll Register Format

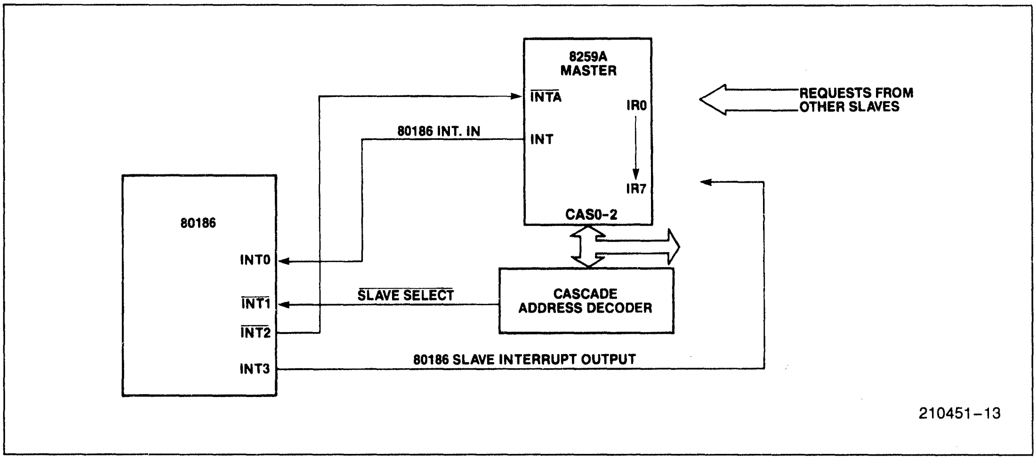


Figure 32. iRMX™ 86 Interrupt Controller Interconnection

Correct master-slave interface requires decoding of the slave addresses (CAS0-2). Slave 8259As do this internally. Because of pin limitations, the 80186 slave address will have to be decoded externally. INT1 is used as a slave-select input. Note that the slave vector address is transferred internally, but the READY input must be supplied externally.

INT2 is used as an acknowledge output, suitable to drive the INTA input of an 8259A.

Interrupt Nesting

iRMX 86 mode operation allows nesting of interrupt requests. When an interrupt is acknowledged, the priority logic masks off all priority levels except those with equal or higher priority.

Vector Generation in the iRMX™ 86 Mode

Vector generation in iRMX mode is exactly like that of an 8259A slave. The interrupt controller generates an 8-bit vector which the CPU multiplies by four and uses as an address into a vector table. The significant five bits of the vector are user-programmable while the lower three bits are generated by the priority logic. These bits represent the encoding of the priority level requesting service. The significant five bits of the vector are programmed by writing to the Interrupt Vector register at offset 20H.

Specific End-of-Interrupt

In iRMX mode the specific EOI command operates to reset an in-service bit of a specific priority. The user supplies a 3-bit priority-level value that points to an in-service bit to be reset. The command is executed by writing the correct value in the Specific EOI register at offset 22H.

Interrupt Controller Registers in the iRMX™ 86 Mode

All control and command registers are located inside the internal peripheral control block. Figure 33 shows the offsets of these registers.

End-of-Interrupt Register

The end-of-interrupt register is a command register which can only be written. The format of this register is shown in Figure 34. It initiates an EOI command when written by the 80186 CPU.

The bits in the EOI register are encoded as follows:

L_X: Encoded value indicating the priority of the IS bit to be reset.

In-Service Register

This register can be read from or written into. It contains the in-service bit for each of the internal interrupt sources. The format for this register is shown in Figure 35. Bit positions 2 and 3 correspond to the DMA channels; positions 0, 4, and 5 correspond to the integral timers. The source's IS bit is set when the processor acknowledges its interrupt request.

Interrupt Request Register

This register indicates which internal peripherals have interrupt requests pending. The format of this register is shown in Figure 35. The interrupt request bits are set when a request arrives from an internal source, and are reset when the processor acknowledges the request.

Mask Register

The register contains a mask bit for each interrupt source. The format for this register is shown in Figure 35. If the bit in this register corresponding to a particular interrupt source is set, any interrupts from that source will be masked. These mask bits are exactly the same bits which are used in the individual control registers, i.e., changing the state of a mask bit in this register will also change the state of the mask bit in the individual interrupt control register corresponding to the bit.

Control Registers

These registers are the control words for all the internal interrupt sources. The format of these registers is shown in Figure 36. Each of the timers and both of the DMA channels have their own Control Register.

The bits of the Control Registers are encoded as follows:

pr_x: 3-bit encoded field indicating a priority level for the source; note that each source must be programmed at specified levels.

msk: mask bit for the priority level indicated by pr_x bits.

Register	OFFSET
LEVEL 5 CONTROL REGISTER (TIMER 2)	3AH
LEVEL 4 CONTROL REGISTER (TIMER 1)	38H
LEVEL 3 CONTROL REGISTER (DMA 1)	36H
LEVEL 2 CONTROL REGISTER (DMA 0)	34H
LEVEL 0 CONTROL REGISTER (TIMER 0)	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT-REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY-LEVEL MASK REGISTER	2AH
MASK REGISTER	28H
SPECIFIC EOI REGISTER	22H
INTERRUPT VECTOR REGISTER	20H

Figure 33. Interrupt Controller Registers (iRMX™ 86 Mode)

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	0	L2	L1	L0

Figure 34. Specific EOI Register Format

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	TMR2	TMR1	D1	D0	0	TMR0

Figure 35. In-Service, Interrupt Request, and Mask Register Format

Interrupt Vector Register

This register provides the upper five bits of the interrupt vector address. The format of this register is shown in Figure 37. The interrupt controller itself provides the lower three bits of the interrupt vector as determined by the priority level of the interrupt request.

The format of the bits in this register is:

t_x: 5-bit field indicating the upper five bits of the vector address.

Priority-Level Mask Register

This register indicates the lowest priority-level interrupt which will be serviced.

The encoding of the bits in this register is:

m_x: 3-bit encoded field indication priority-level value. All levels of lower priority will be masked.

Interrupt Status Register

This register is defined exactly as in Non-iRMX Mode. (See Fig. 26.)

Interrupt Controller and Reset

Upon RESET, the interrupt controller will perform the following actions:

- All SFNM bits reset to 0, implying Fully Nested Mode.
- All PR bits in the various control registers set to 1. This places all sources at lowest priority (level 111).
- All LTM bits reset to 0, resulting in edge-sense mode.
- All Interrupt Service bits reset to 0.
- All Interrupt Request bits reset to 0.
- All MSK (Interrupt Mask) bits set to 1 (mask).
- All C (Cascade) bits reset to 0 (non-cascade).
- All PRM (Priority Mask) bits set to 1, implying no levels masked.
- Initialized to non-iRMX 86 mode.

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	MSK	PR2	PR1	PR0

Figure 36. Control Word Format

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	t4	t3	t2	t1	t0	0	0	0

Figure 37. Interrupt Vector Register Format

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	0	m2	m1	m0

Figure 38. Priority Level Mask Register

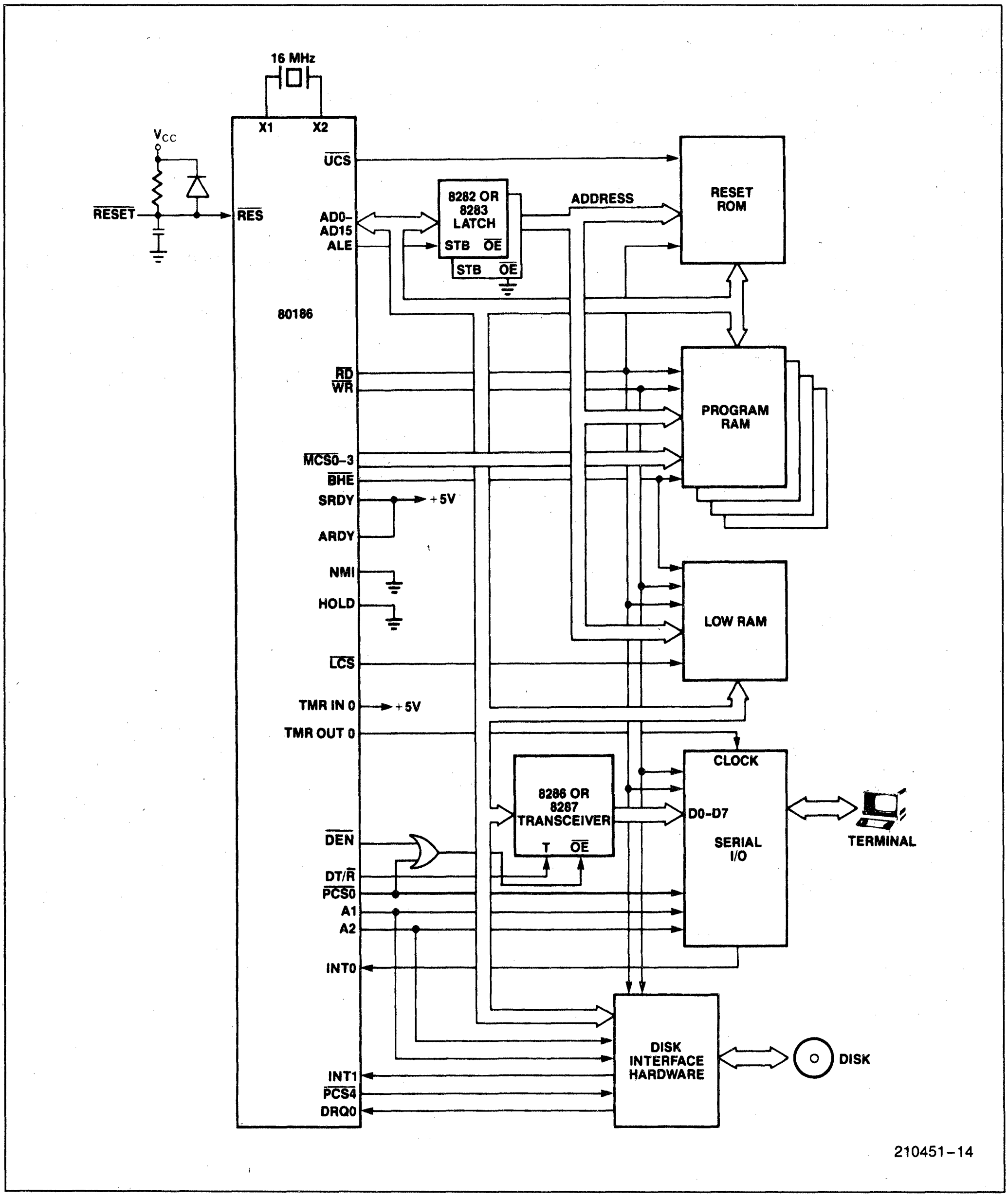


Figure 39. Typical 80186 Computer

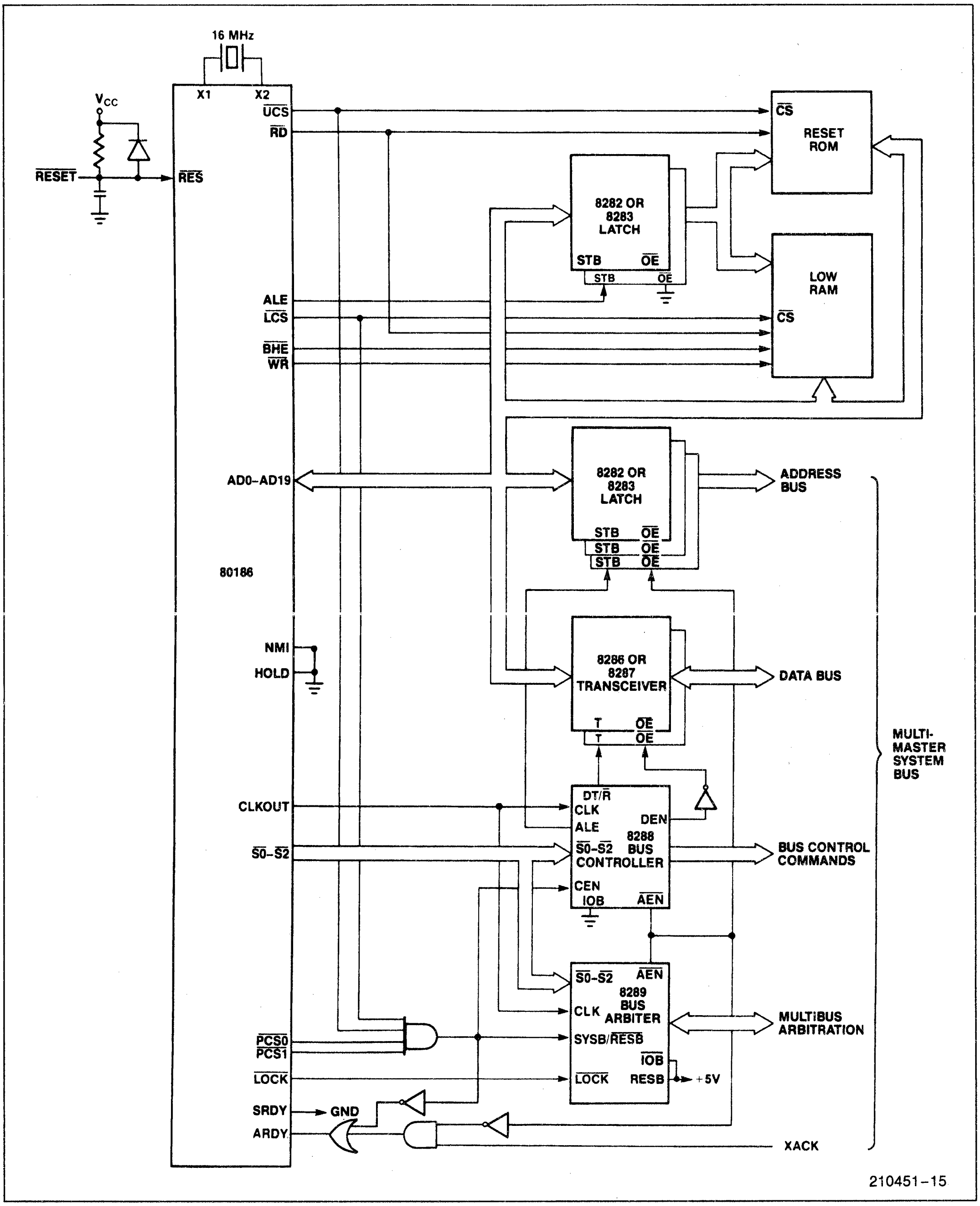


Figure 40. Typical 80186 Multi-Master Bus Interface

PACKAGE

The 80186 is available in two 68 pin hermetic packages. They are the JEDEC type A leadless chip carrier and the JEDEC type CG pin grid array. Figures 41A and 41B illustrate the package dimensions.

NOTE:
The IDT 3M Textool 68-pin JEDEC Socket is required for I2ICE™ 186 operation. See Figure 42 for details.

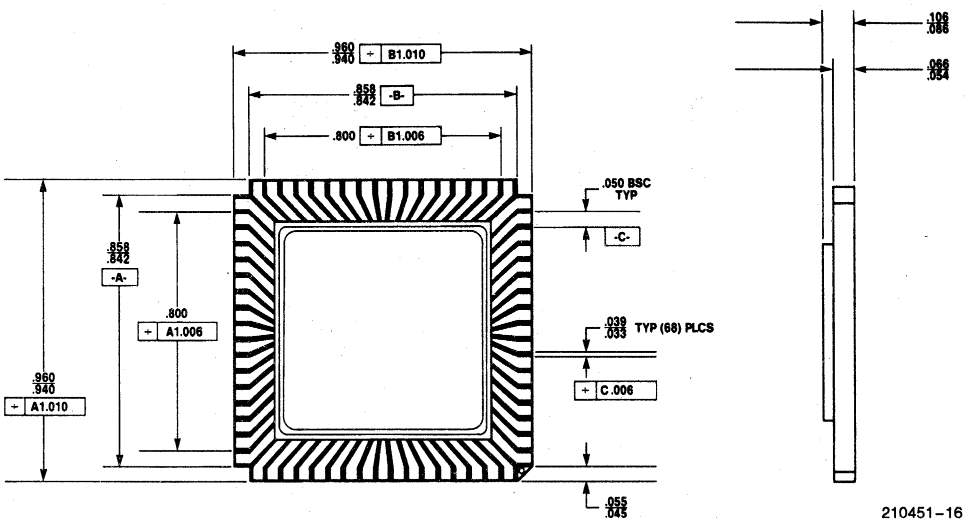
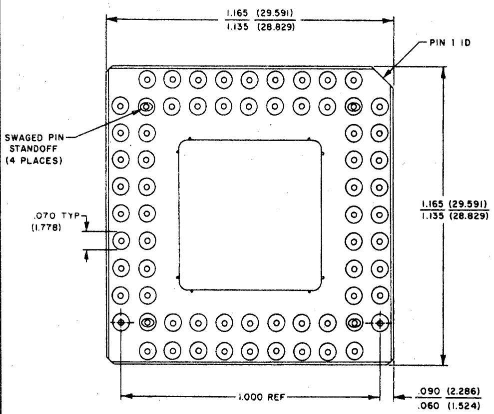
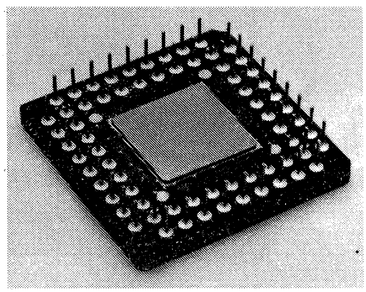


Figure 41A. 80186 JEDEC Type A Package



68-Lead Ceramic Pin Grid Array Package Type CG



210451-17

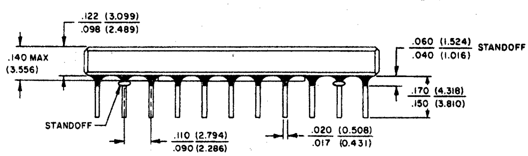
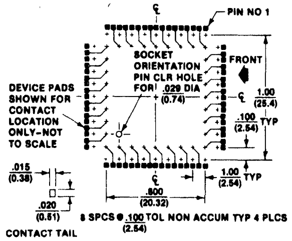
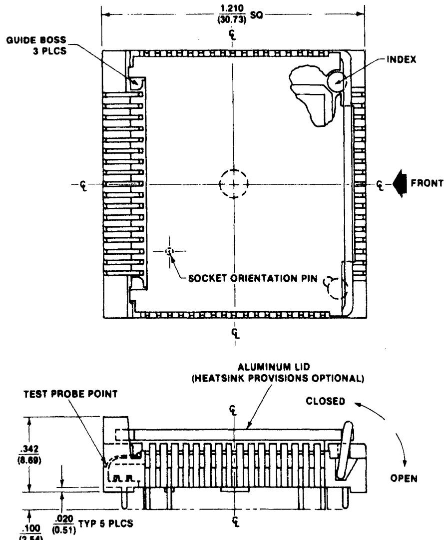


Figure 41B. Ceramic Pin Grid Array Package Type CG

PC BOARD PATTERN



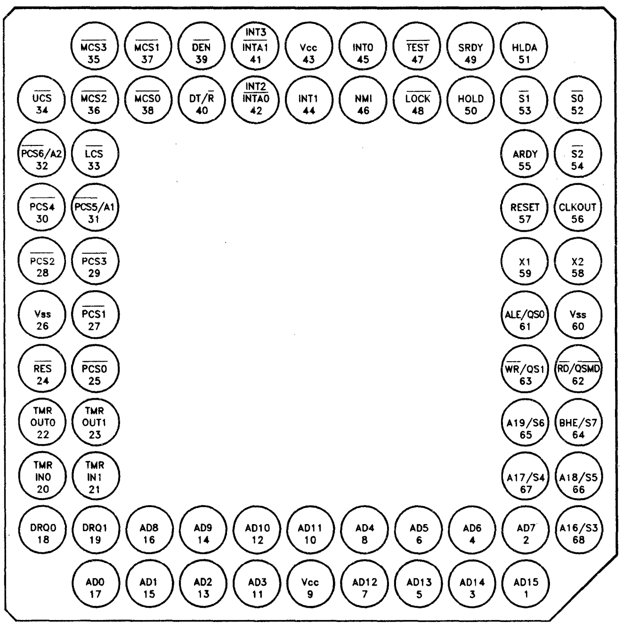
210451-20



210451-21

NOTE:
Physical dimensions shown are for reference only. Please consult 3M Textool for complete information on the socket.

Figure 42. Textool 68 Lead Chip Carrier Socket



210451-22

Bottom View (Pins Facing Up)

Figure 43. Pin Grid Array, PLCC and LCC Socket Pinout

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias ...... 0°C to 70°C
Case Temperature under Bias ...... 0°C to +110°C
Storage Temperature .... −65°C to +150°C
Voltage on any Pin with Respect to Ground .... −1.0V to +7V
Power Dissipation .... 3 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to +70°C, T_C = 0°C to +110°C, V_{CC} = 5V ±10%)

Applicable to 80186 (8 MHz), 80186-10 (10 MHz), and 80186-12 (12.5 MHz)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	0.5	+0.8	Volts	
V_{IH}	Input High Voltage (All except X1 and ($\overline{RES}$)	2.0	V_{CC} + 0.5	Volts	
V_{IH1}	Input High Voltage ($\overline{RES}$)	3.0	V_{CC} + 0.5	Volts	
V_{OL}	Output Low Voltage		0.45	Volts	I_a = 2.5 mA for $\overline{S0}$–$\overline{S2}$ I_a = 2.0 mA for all other outputs
V_{OH}	Output High Voltage	2.4		Volts	I_{oa} = −400 μA
I_{CC}	Power Supply Current		550 450	mA	Max measured at T_A = 0°C T_A = 70°C
I_{LI}	Input Leakage Current		±10	μA	0V < V_{IN} < V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V < V_{OUT} < V_{CC}
V_{CLO}	Clock Output Low		0.6	Volts	I_a = 4.0 mA
V_{CHO}	Clock Output High	4.0		Volts	I_{oa} = −200 μA
V_{CLI}	Clock Input Low Voltage	−0.5	0.6	Volts	
V_{CHI}	Clock Input High Voltage	3.9	V_{CC} + 1.0	Volts	
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		20	pF	

PIN TIMINGS

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, T_C = 0°C to +110°C, V_{CC} = 5V ±10%)

80186 Timing Requirements All Timings Measured At 1.5 Volts Unless Otherwise Noted.

Symbol	Parameter	80186 (8 MHz) Min	80186 (8 MHz) Max	80186-10 (10 MHz) Min	80186-10 (10 MHz) Max	80186-12 (12.5 MHz) Preliminary Min	80186-12 (12.5 MHz) Preliminary Max	Units	Test Conditions
T_{DVCL}	Data in Setup (A/D)	20		15		15		ns	
T_{CLDX}	Data in Hold (A/D)	10		8		8		ns	
T_{ARYHCH}	Asynchronous Ready (AREADY) active setup time*	20		15		15		ns	
T_{ARYLCL}	AREADY inactive setup time	35		25		25		ns	
T_{CHARYX}	AREADY hold time	15		15		15		ns	
T_{ARYCHL}	Asynchronous Ready inactive hold time	15		15		15		ns	
T_{SRYCL}	Synchronous Ready (SREADY) transition setup time	20		20		20		ns	
T_{CLSRY}	SREADY transition hold time	15		15		15		ns	
T_{HVCL}	HOLD Setup*	25		20		20		ns	
T_{INVCH}	INTR, NMI, $\overline{TEST}$, TIMERIN, Setup*	25		25		25		ns	
T_{INVCL}	DRQ0, DRQ1, Setup*	25		20		20		ns	

80186 Master Interface Timing Responses

Symbol	Parameter	80186 (8 MHz) Min	80186 (8 MHz) Max	80186-10 (10 MHz) Min	80186-10 (10 MHz) Max	80186-12 (12.5 MHz) Preliminary Min	80186-12 (12.5 MHz) Preliminary Max	Units	Test Conditions
T_{CLAV}	Address Valid Delay	5	55	5	50	4	33	ns	C_L = 20–200 pF all outputs (except T_{CLTMV}) @ 8 & 10 MHz C_L = 20–100 pF all outputs @ 12.5 MHz
T_{CLAX}	Address Hold	10		10		8		ns	
T_{CLAZ}	Address Float Delay	T_{CLAX}	35	T_{CLAX}	30	8	25	ns	
T_{CHCZ}	Command Lines Float Delay		45		40		33	ns	
T_{CHCV}	Command Lines Valid Delay (after float)		55		45		37	ns	
T_{LHLL}	ALE Width	$T_{CLCL}-35$		$T_{CLCL}-30$		$T_{CLCL}-30$		ns	
T_{CHLH}	ALE Active Delay		35		30		25	ns	
T_{CHLL}	ALE Inactive Delay		35		30		25	ns	
T_{LLAX}	Address Hold to ALE Inactive	$T_{CHCL}-25$		$T_{CHCL}-20$		$T_{CHCL}-15$		ns	
T_{CLDV}	Data Valid Delay	10	44	10	40	8	33	ns	
T_{CLDOX}	Data Hold Time	10		10		8		ns	
T_{WHDX}	Data Hold after WR	$T_{CLCL}-40$		$T_{CLCL}-34$		$T_{CLCL}-20$		ns	
T_{CVCTV}	Control Active Delay 1	10	70	5	56	8	47	ns	
T_{CHCTV}	Control Active Delay 2	10	55	10	44	8	37	ns	
T_{CVCTX}	Control Inactive Delay	5	55	5	44	4	37	ns	
T_{CVDEX}	$\overline{DEN}$ Inactive Delay (Non-Write Cycle)	10	70	10	56	8	47	ns	

*To guarantee recognition at next clock.

PIN TIMINGS (Continued)

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, T_C = 0°C to +110°C, V_{CC} = 5V ±10%) (Continued)

80186 Master Interface Timing Responses (Continued)

Symbol	Parameter	80186 (8 MHz)		80186-10 (10 MHz)		80186-12 (12.5 MHz) Preliminary		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
T_{AZRL}	Address Float to $\overline{RD}$ Active	0		0		0		ns	
T_{CLRL}	$\overline{RD}$ Active Delay	10	70	10	56	8	47	ns	
T_{CLRH}	$\overline{RD}$ Inactive Delay	10	55	10	44	8	37	ns	
T_{RHAV}	$\overline{RD}$ Inactive to Address Active	$T_{CLCL}-40$		$T_{CLCL}-40$		$T_{CLCL}-20$		ns	
T_{CLHAV}	HLDA Valid Delay	5	50	5	40	4	33	ns	
T_{RLRH}	$\overline{RD}$ Width	$2T_{CLCL}-50$		$2T_{CLCL}-46$		$2T_{CLCL}-40$		ns	
T_{WLWH}	$\overline{WR}$ Width	$2T_{CLCL}-40$		$2T_{CLCL}-34$		$2T_{CLCL}-30$		ns	
T_{AVAL}	Address Valid to ALE Low	$T_{CLCH}-25$		$T_{CLCH}-19$		$T_{CLCH}-15$		ns	
T_{CHSV}	Status Active Delay	10	55	10	45	8	35	ns	
T_{CLSH}	Status Inactive Delay	10	65	10	50	8	35	ns	
T_{CLTMV}	Timer Output Delay		60		48		40	ns	100 pF max @ 8 & 10 MHz
T_{CLRO}	Reset Delay		60		48		40	ns	
T_{CHQSV}	Queue Status Delay		35		28		23	ns	
T_{CHDX}	Status Hold Time	10		10		8		ns	
T_{AVCH}	Address Valid to Clock High	10		10		8		ns	
T_{CLLV}	$\overline{LOCK}$ Valid/Invalid Delay	5	65	5	60	5	55	ns	

80186 Chip-Select Timing Responses

Symbol	Parameter	80186 Min	80186 Max	80186-10 Min	80186-10 Max	80186-12 Min	80186-12 Max	Units	Test Conditions
T_{CLCSV}	Chip-Select Active Delay		66		45		33	ns	
T_{CXCSX}	Chip-Select Hold from Command Inactive	35		35		29		ns	
T_{CHCSX}	Chip-Select Inactive Delay	5	35	5	32	4	23	ns	

80186 CLKIN Requirements

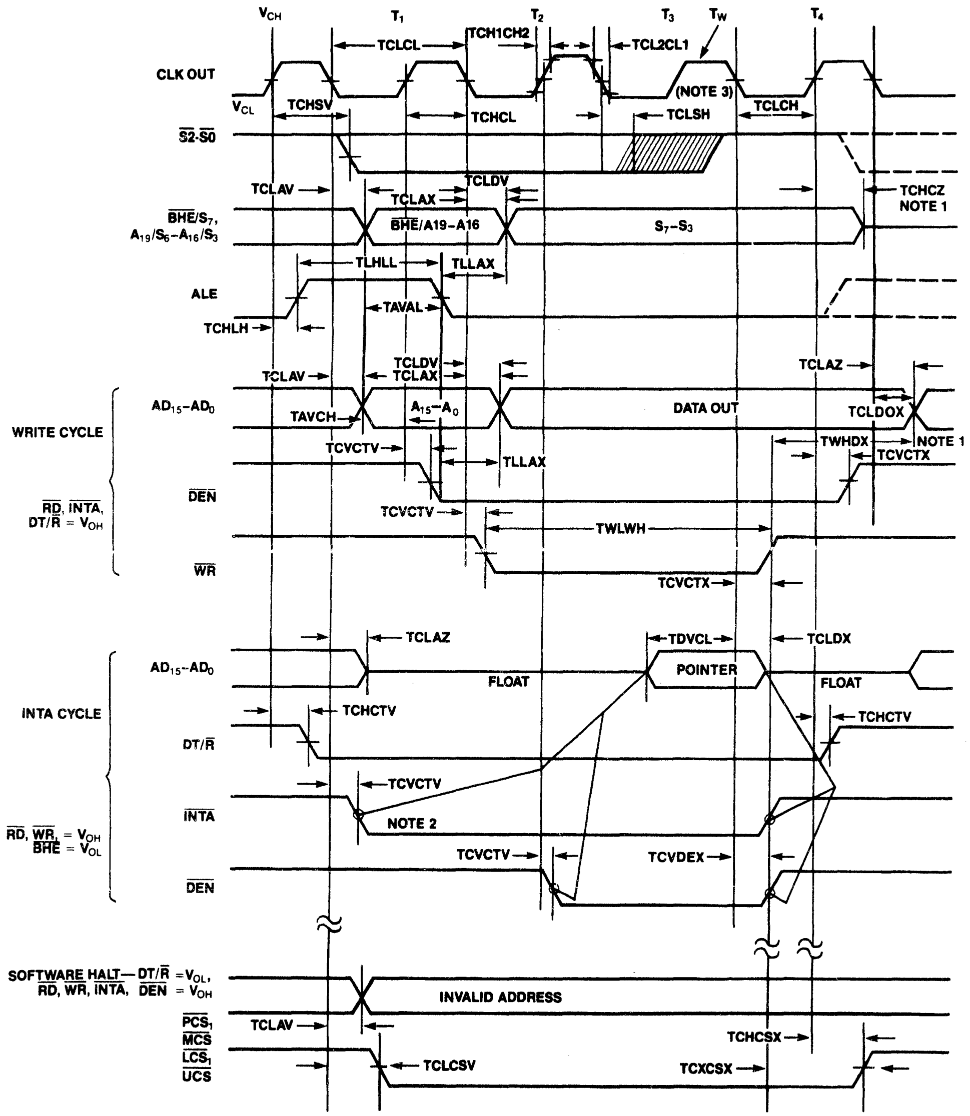
Symbol	Parameter	80186 Min	80186 Max	80186-10 Min	80186-10 Max	80186-12 Min	80186-12 Max	Units	Test Conditions
T_{CKIN}	CLKIN Period	62.5	250	50	250	40	250	ns	
T_{CKHL}	CLKIN Fall Time		10		10		8	ns	3.5 to 1.0V
T_{CKLH}	CLKIN Rise Time		10		10		8	ns	1.0 to 3.5V
T_{CLCK}	CLKIN Low Time	25		20		15		ns	1.5V
T_{CHCK}	CLKIN High Time	25		20		15		ns	1.5V

80186 CLKOUT Timing (200 pF load)

Symbol	Parameter	80186 Min	80186 Max	80186-10 Min	80186-10 Max	80186-12 Min	80186-12 Max	Units	Test Conditions
T_{CICO}	CLKIN to CLKOUT Skew		50		25		21	ns	
T_{CLCL}	CLKOUT Peiod	125	500	100	500	80	500	ns	
T_{CLCH}	CLKOUT Low Time	$\frac{1}{2}T_{CLCL}-7.5$		$\frac{1}{2}T_{CLCL}-6.0$		$\frac{1}{2}T_{CLCL}-6.0$		ns	1.5V
T_{CHCL}	CLKOUT High Time	$\frac{1}{2}T_{CLCL}-7.5$		$\frac{1}{2}T_{CLCL}-6.0$		$\frac{1}{2}T_{CLCL}-6.0$		ns	1.5V
T_{CH1CH2}	CLKOUT Rise Time		15		12		10	ns	1.0 to 3.5V
T_{CL2CL1}	CLKOUT Fall Time		15		12		10	ns	3.5 to 1.0V

WAVEFORMS

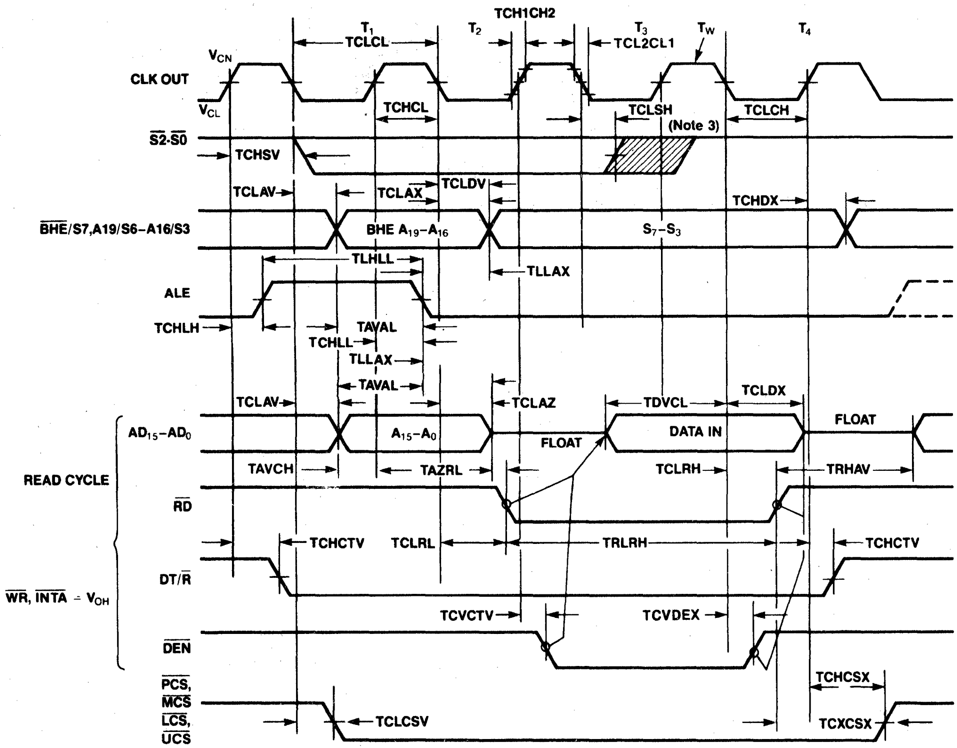
MAJOR CYCLE TIMING



210451-23

WAVEFORMS (Continued)

MAJOR CYCLE TIMING (Continued)

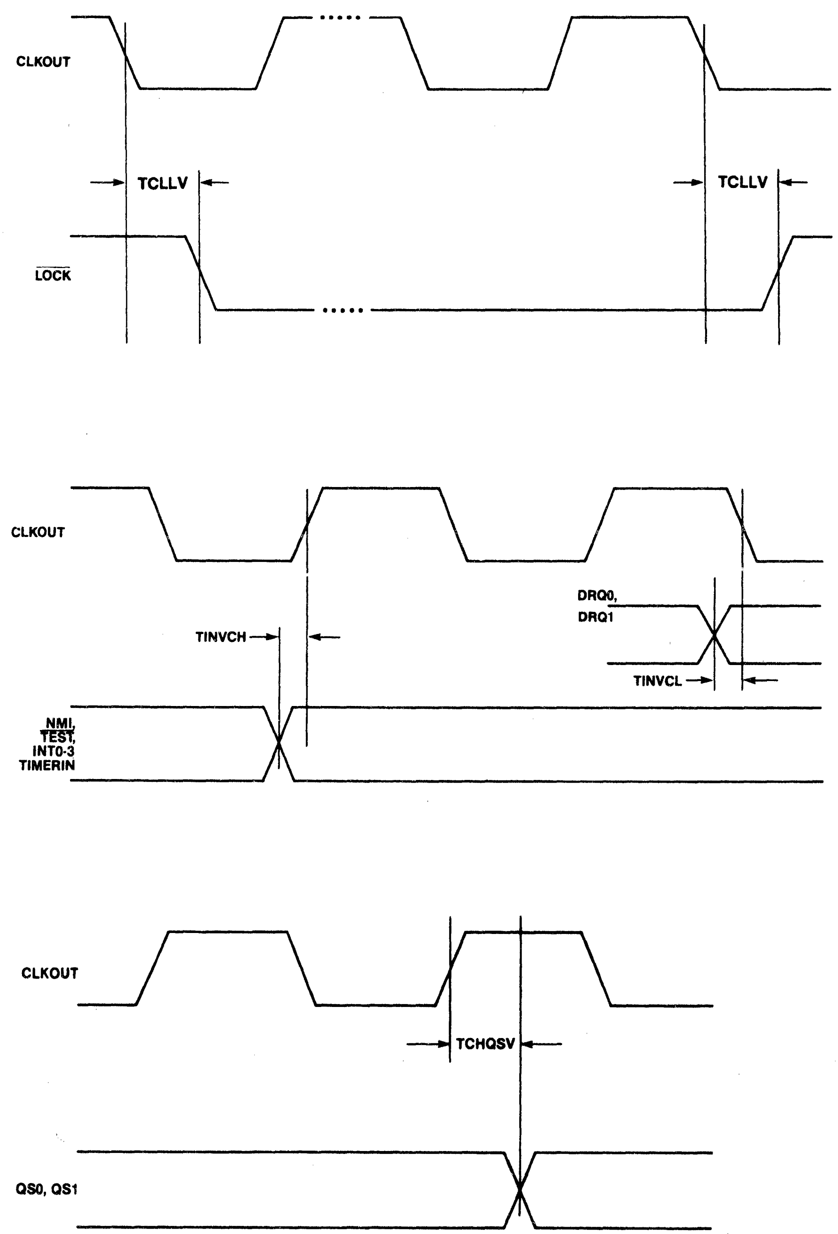


210451-24

NOTES:

1. Following a Write cycle, the Local Bus is floated by the 80186 only when the 80186 enters a "Hold Acknowledge" state.
2. INTA occurs one clock later in RMX-mode.
3. Status inactive just prior to T_4.

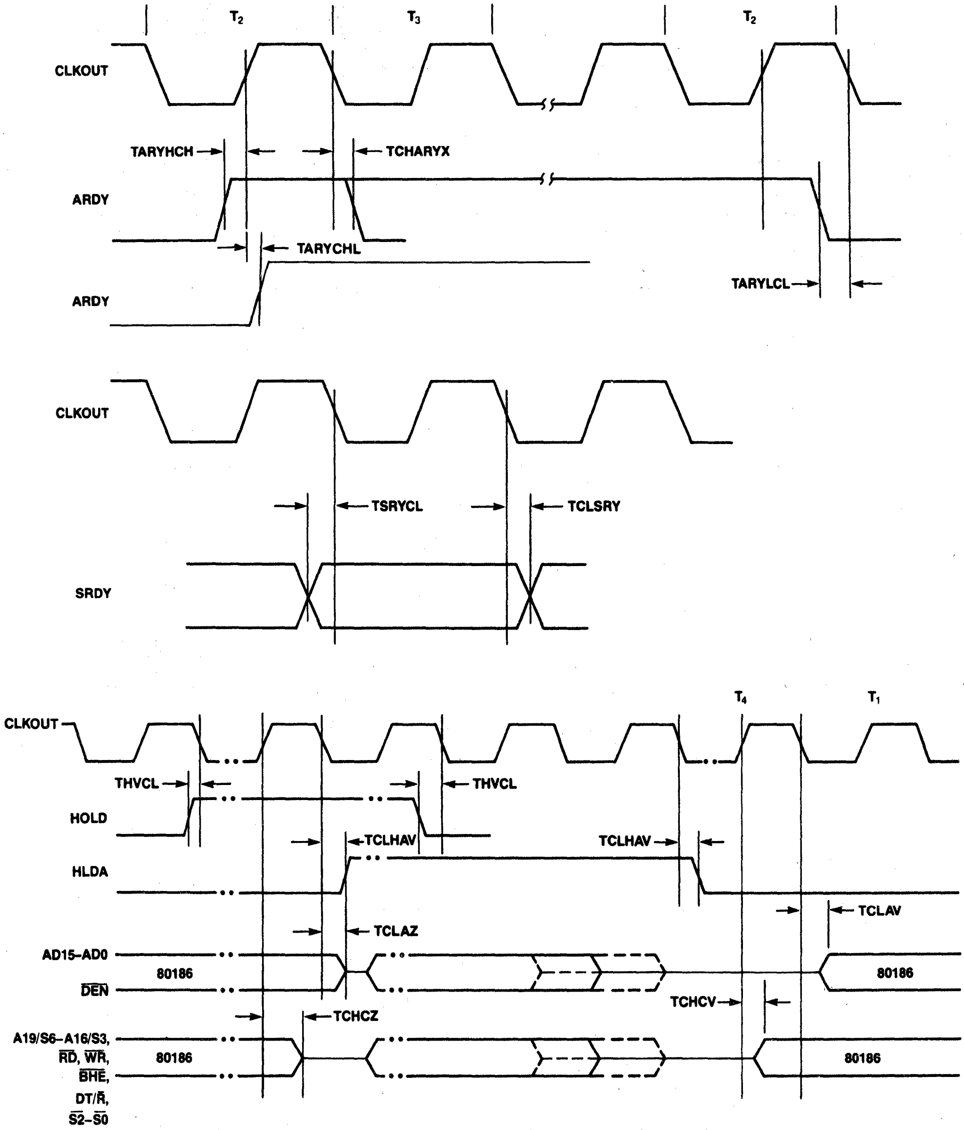
WAVEFORMS (Continued)



210451–25

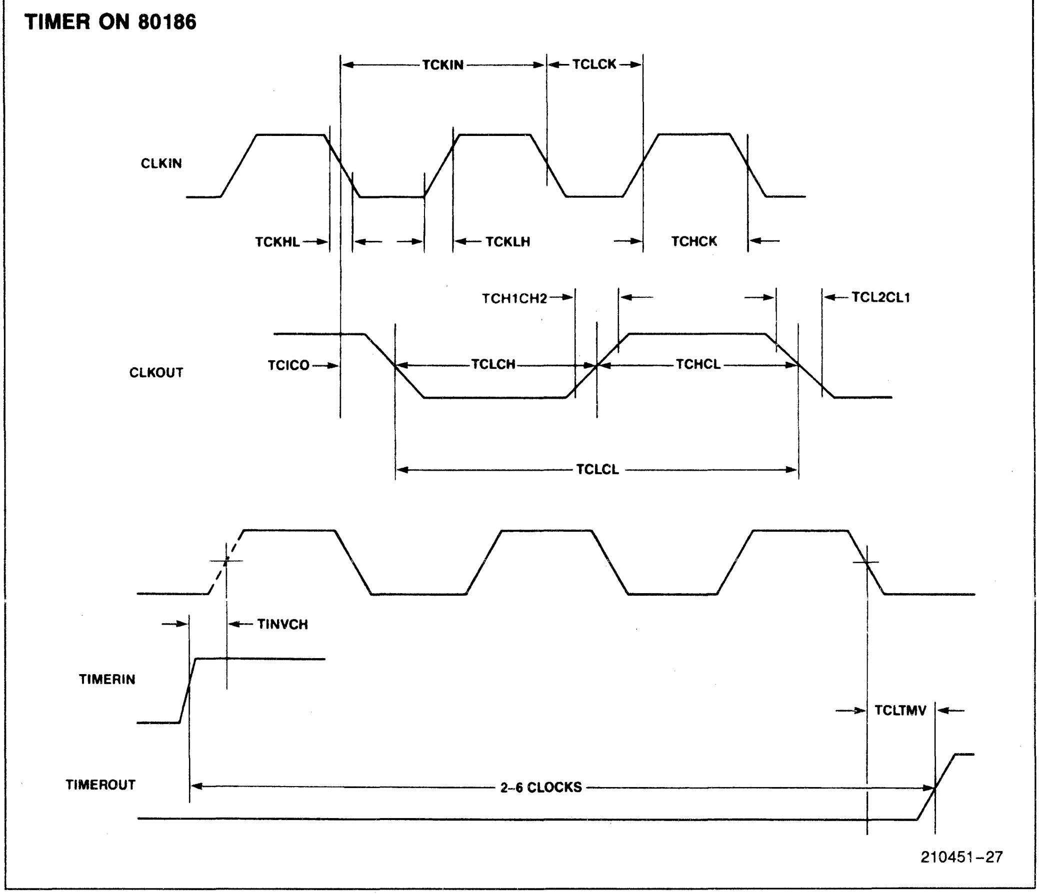
WAVEFORMS (Continued)

HOLD-HLDA TIMING



210451–26

WAVEFORMS (Continued)



80186 INSTRUCTION TIMINGS

The following instruction timings represent the minimum execution time in clock cycles for each instruction. The timings given are based on the following assumptions:

- The opcode, along with any data or displacement required for execution of a particular instruction, has been prefetched and resides in the queue at the time it is needed.
- No wait states or bus HOLDS occur.
- All word-data is located on even-address boundaries.

All jumps and calls include the time required to fetch the opcode of the next instruction at the destination address.

All instructions which involve memory reference can require one (and in some cases, two) additional clocks above the minimum timings shown. This is due to the asynchronous nature of the handshake between the BIU and the Execution unit.

INSTRUCTION SET SUMMARY

Function	Format				Clock Cycles	Comments
DATA TRANSFER						
MOV = Move:						
Register to Register/Memory	1 0 0 0 1 0 0 w	mod reg r/m			2/12	
Register/memory to register	1 0 0 0 1 0 1 w	mod reg r/m			2/9	
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 000 r/m	data	data if w = 1	12–13	8/16-bit
Immediate to register	1 0 1 1 w reg	data	data if w = 1		3–4	8/16-bit
Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high		9	
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high		8	
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 reg r/m			2/9	
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 reg r/m			2/11	
PUSH = Push:						
Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m			16	
Register	0 1 0 1 0 reg				10	
Segment register	0 0 0 reg 1 1 0				9	
Immediate	0 1 1 0 1 0 s 0	data	data if s = 0		10	
PUSHA = Push All	0 1 1 0 0 0 0 0				36	
POP = Pop:						
Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m			20	
Register	0 1 0 1 1 reg				10	
Segment register	0 0 0 reg 1 1 1	(reg≠01)			8	
POPA = Pop All	0 1 1 0 0 0 0 1				51	
XCHG = Exchange:						
Register/memory with register	1 0 0 0 0 1 1 w	mod reg r/m			4/17	
Register with accumulator	1 0 0 1 0 reg				3	
IN = Input from:						
Fixed port	1 1 1 0 0 1 0 w	port			10	
Variable port	1 1 1 0 1 1 0 w				8	
OUT = Output to:						
Fixed port	1 1 1 0 0 1 1 w	port			9	
Variable port	1 1 1 0 1 1 1 w				7	
XLAT = Translate byte to AL	1 1 0 1 0 1 1 1				11	
LEA = Load EA to register	1 0 0 0 1 1 0 1	mod reg r/m			6	
LDS = Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m	(mod≠11)		18	
LES = Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m	(mod≠11)		18	
LAHF = Load AH with flags	1 0 0 1 1 1 1 1				2	
SAHF = Store AH into flags	1 0 0 1 1 1 1 0				3	
PUSHF = Push flags	1 0 0 1 1 1 0 0				9	
POPF = Pop flags	1 0 0 1 1 1 0 1				8	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
DATA TRANSFER (Continued)						
SEGMENT = Segment Override:						
CS	00101110				2	
SS	00110110				2	
DS	00111110				2	
ES	00100110				2	
ARITHMETIC						
ADD = Add:						
Reg/memory with register to either	000000dw	mod reg r/m			3/10	
Immediate to register/memory	100000sw	mod 000 r/m	data	data if s w = 01	4/16	
Immediate to accumulator	0000010w	data	data if w = 1		3/4	8/16-bit
ADC = Add with carry:						
Reg/memory with register to either	000100dw	mod reg r/m			3/10	
Immediate to register/memory	100000sw	mod 010 r/m	data	data if s w = 01	4/16	
Immediate to accumulator	0001010w	data	data if w = 1		3/4	8/16-bit
INC = Increment:						
Register/memory	1111111w	mod 000 r/m			3/15	
Register	01000 reg				3	
SUB = Subtract:						
Reg/memory and register to either	001010dw	mod reg r/m			3/10	
Immediate from register/memory	100000sw	mod 101 r/m	data	data if s w = 01	4/16	
Immediate from accumulator	0010110w	data	data if w = 1		3/4	8/16-bit
SBB = Subtract with borrow:						
Reg/memory and register to either	000110dw	mod reg r/m			3/10	
Immediate from register/memory	100000sw	mod 011 r/m	data	data if s w = 01	4/16	
Immediate from accumulator	0001110w	data	data if w = 1		3/4	8/16-bit
DEC = Decrement						
Register/memory	1111111w	mod 001 r/m			3/15	
Register	01001 reg				3	
CMP = Compare:						
Register/memory with register	0011101w	mod reg r/m			3/10	
Register with register/memory	0011100w	mod reg r/m			3/10	
Immediate with register/memory	100000sw	mod 111 r/m	data	data if s w = 01	3/10	
Immediate with accumulator	0011110w	data	data if w = 1		3/4	8/16-bit
NEG = Change sign	1111011w	mod 011 r/m			3	
AAA = ASCII adjust for add	00110111				8	
DAA = Decimal adjust for add	00100111				4	
AAS = ASCII adjust for subtract	00111111				7	
DAS = Decimal adjust for subtract	00101111				4	
MUL = Multiply (unsigned):	1111011w	mod 100 r/m				
Register-Byte					26–28	
Register-Word					35–37	
Memory-Byte					32–34	
Memory-Word					41–43	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
ARITHMETIC (Continued)						
IMUL = Integer multiply (signed):	1 1 1 1 0 1 1 w	mod 1 0 1 r/m				
Register-Byte					25–28	
Register-Word					34–37	
Memory-Byte					31–34	
Memory-Word					40–43	
IMUL = Integer Immediate multiply (signed)	0 1 1 0 1 0 s 1	mod reg r/m	data	data if s=0	22–25/ 29–32	
DIV = Divide (unsigned):	1 1 1 1 0 1 1 w	mod 1 1 0 r/m				
Register-Byte					29	
Register-Word					38	
Memory-Byte					35	
Memory-Word					44	
IDIV = Integer divide (signed):	1 1 1 1 0 1 1 w	mod 1 1 1 r/m				
Register-Byte					44–52	
Register-Word					53–61	
Memory-Byte					50–58	
Memory-Word					59–67	
AAM = ASCII adjust for multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0			19	
AAD = ASCII adjust for divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0			15	
CBW = Convert byte to word	1 0 0 1 1 0 0 0				2	
CWD = Convert word to double word	1 0 0 1 1 0 0 1				4	
LOGIC						
Shift/Rotate Instructions:						
Register/Memory by 1	1 1 0 1 0 0 0 w	mod TTT r/m			2/15	
Register/Memory by CL	1 1 0 1 0 0 1 w	mod TTT r/m			5+n/17+n	
Register/Memory by Count	1 1 0 0 0 0 0 w	mod TTT r/m	count		5+n/17+n	
	TTT Instruction 0 0 0 ROL 0 0 1 ROR 0 1 0 RCL 0 1 1 RCR 1 0 0 SHL/SAL 1 0 1 SHR 1 1 1 SAR					
AND = And:						
Reg/memory and register to either	0 0 1 0 0 0 d w	mod reg r/m			3/10	
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w=1	4/16	
Immediate to accumulator	0 0 1 0 0 1 0 w	data	data if w=1		3/4	8/16-bit
TEST = And function to flags, no result:						
Register/memory and register	1 0 0 0 0 1 0 w	mod reg r/m			3/10	
Immediate data and register/memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w=1	4/10	
Immediate data and accumulator	1 0 1 0 1 0 0 w	data	data if w=1		3/4	8/16-bit
OR = Or:						
Reg/memory and register to either	0 0 0 0 1 0 d w	mod reg r/m			3/10	
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w=1	4/16	
Immediate to accumulator	0 0 0 0 1 1 0 w	data	data if w=1		3/4	8/16-bit

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format					Clock Cycles	Comments
LOGIC (Continued)							
XOR = **Exclusive or:**							
Reg/memory and register to either	0 0 1 1 0 0 d w	mod reg r/m				3/10	
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w=1		4/16	
Immediate to accumulator	0 0 1 1 0 1 0 w	data	data if w=1			3/4	8/16-bit
NOT = Invert register/memory	1 1 1 1 0 1 1 w	mod 0 1 0 r/m				3	
STRING MANIPULATION							
MOVS = Move byte/word	1 0 1 0 0 1 0 w					14	
CMPS = Compare byte/word	1 0 1 0 0 1 1 w					22	
SCAS = Scan byte/word	1 0 1 0 1 1 1 w					15	
LODS = Load byte/wd to ALAX	1 0 1 0 1 1 0 w					12	
STOS = Stor byte/wd from ALA	1 0 1 0 1 0 1 w					10	
INS = Input byte/wd from DX port	0 1 1 0 1 1 0 w					14	
OUTS = Output byte/wd to DX port	0 1 1 0 1 1 1 w					14	
Repeated by count in CX							
MOVS = Move string	1 1 1 1 0 0 1 0	1 0 1 0 0 1 0 w				8+8n	
CMPS = Compare string	1 1 1 1 0 0 1 z	1 0 1 0 0 1 1 w				5+22n	
SCAS = Scan string	1 1 1 1 0 0 1 z	1 0 1 0 1 1 1 w				5+15n	
LODS = Load string	1 1 1 1 0 0 1 0	1 0 1 0 1 1 0 w				6+11n	
STOS = Store string	1 1 1 1 0 0 1 0	1 0 1 0 1 0 1 w				6+9n	
INS = Input string	1 1 1 1 0 0 1 0	0 1 1 0 1 1 0 w				8+8n	
OUTS = Output string	1 1 1 1 0 0 1 0	0 1 1 0 1 1 1 w				8+8n	
CONTROL TRANSFER							
CALL = **Call:**							
Direct within segment	1 1 1 0 1 0 0 0	disp-low	disp-high			15	
Register/memory indirect within segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m				13/19	
Direct intersegment	1 0 0 1 1 0 1 0	segment offset				23	
		segment selector					
Indirect intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m	(mod ≠ 11)			38	
JMP = **Unconditional jump:**							
Short/long	1 1 1 0 1 0 1 1	disp-low				14	
Direct within segment	1 1 1 0 1 0 0 1	disp-low	disp-high			14	
Register/memory indirect within segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m				11/17	
Direct intersegment	1 1 1 0 1 0 1 0	segment offset				14	
		segment selector					
Indirect intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m	(mod ≠ 11)			26	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
CONTROL TRANSFER (Continued)			
RET = **Return from CALL:**			
Within segment	11000011	16	
Within seg adding immed to SP	11000010 · data-low · data-high	18	
Intersegment	11001011	22	
Intersegment adding immediate to SP	11001010 · data-low · data-high	25	
JE/JZ = Jump on equal/zero	01110100 · disp	4/13	JMP not taken/JMP taken
JL/JNGE = Jump on less/not greater or equal	01111100 · disp	4/13	
JLE/JNG = Jump on less or equal/not greater	01111110 · disp	4/13	
JB/JNAE = Jump on below/not above or equal	01110010 · disp	4/13	
JBE/JNA = Jump on below or equal/not above	01110110 · disp	4/13	
JP/JPE = Jump on parity/parity even	01111010 · disp	4/13	
JO = Jump on overflow	01110000 · disp	4/13	
JS = Jump on sign	01111000 · disp	4/13	
JNE/JNZ = Jump on not equal/not zero	01110101 · disp	4/13	
JNL/JGE = Jump on not less/greater or equal	01111101 · disp	4/13	
JNLE/JG = Jump on not less or equal/greater	01111111 · disp	4/13	
JNB/JAE = Jump on not below/above or equal	01110011 · disp	4/13	
JNBE/JA = Jump on not below or equal/above	01110111 · disp	4/13	
JNP/JPO = Jump on not par/par odd	01111011 · disp	4/13	
JNO = Jump on not overflow	01110001 · disp	4/13	
JNS = Jump on not sign	01111001 · disp	4/13	
JCXZ = Jump on CX zero	11100011 · disp	5/15	
LOOP = Loop CX times	11100010 · disp	6/16	LOOP not taken/LOOP taken
LOOPZ/LOOPE = Loop while zero/equal	11100001 · disp	6/16	
LOOPNZ/LOOPNE = Loop while not zero/equal	11100000 · disp	6/16	
ENTER = Enter Procedure	11001000 · data-low · data-high · L		
L = 0		15	
L = 1		25	
L > 1		22+16(n−1)	
LEAVE = Leave Procedure	11001001	8	
INT = **Interrupt:**			
Type specified	11001101 · type	47	
Type 3	11001100	45	if INT. taken/ if INT. not taken
INTO = Interrupt on overflow	11001110	48/4	
IRET = Interrupt return	11001111	28	
BOUND = Detect value out of range	01100010 · mod reg r/m	33–35	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format		Clock Cycles	Comments
PROCESSOR CONTROL				
CLC = Clear carry	1 1 1 1 1 0 0 0		2	
CMC = Complement carry	1 1 1 1 0 1 0 1		2	
STC = Set carry	1 1 1 1 1 0 0 1		2	
CLD = Clear direction	1 1 1 1 1 1 0 0		2	
STD = Set direction	1 1 1 1 1 1 0 1		2	
CLI = Clear interrupt	1 1 1 1 1 0 1 0		2	
STI = Set interrupt	1 1 1 1 1 0 1 1		2	
HLT = Halt	1 1 1 1 0 1 0 0		2	
WAIT = Wait	1 0 0 1 1 0 1 1		6	if $\overline{test}$ = 0
LOCK = Bus lock prefix	1 1 1 1 0 0 0 0		2	
ESC = Processor Extension Escape	1 1 0 1 1 T T T	mod LLL r/m	6	
	(TTT LLL are opcode to processor extension)			

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

PRELIMINARY

iAPX 88/10
8-BIT HMOS MICROPROCESSOR
8088/8088-2

- 8-Bit Data Bus Interface
- 16-Bit Internal Architecture
- Direct Addressing Capability to 1 Mbyte of Memory
- Direct Software Compatibility with iAPX 86/10 (8086 CPU)
- 14-Word by 16-Bit Register Set with Symmetrical Operations
- 24 Operand Addressing Modes
- Byte, Word, and Block Operations
- 8-Bit and 16-Bit Signed and Unsigned Arithmetic in Binary or Decimal, Including Multiply and Divide
- Compatible with 8155-2, 8755A-2 and 8185-2 Multiplexed Peripherals
- Two Clock Rates:
  5 MHz for 8088
  8 MHz for 8088-2
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range

The Intel® iAPX 88/10 is a new generation, high performance microprocessor implemented in N-channel, depletion load, silicon gate technology (HMOS), and packaged in a 40-pin CerDIP package. The processor has attributes of both 8- and 16-bit microprocessors. It is directly compatible with iAPX 86/10 software and 8080/8085 hardware and peripherals.

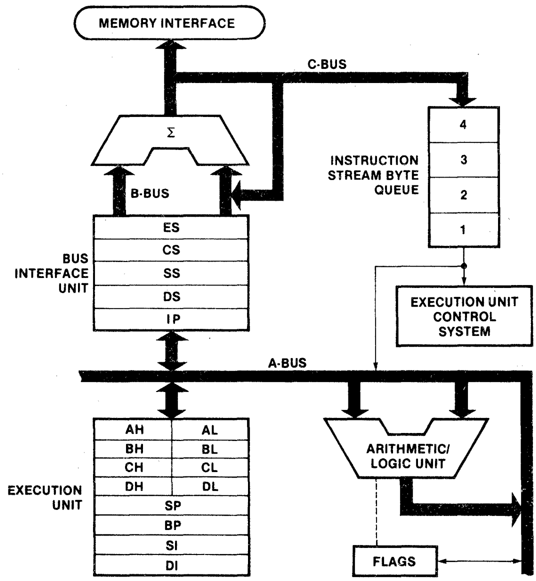


Figure 1. iAPX 88/10 CPU Functional Block Diagram

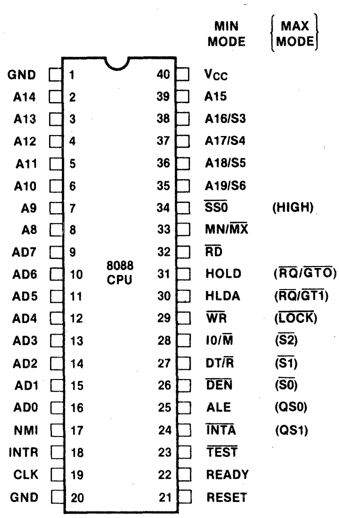


Figure 2. iAPX 88/10 Pin Configuration

Table 1. Pin Description

The following pin function descriptions are for 8088 systems in either minimum or maximum mode. The "local bus" in these descriptions is the direct multiplexed bus interface connection to the 8088 (without regard to additional bus buffers).

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>AD7–AD0</td><td>9-16</td><td>I/O</td><td>Address Data Bus: These lines constitute the time multiplexed memory/IO address (T1) and data (T2, T3, Tw, and T4) bus. These lines are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge".</td></tr>
<tr><td>A15–A8</td><td>2-8, 39</td><td>O</td><td>Address Bus: These lines provide address bits 8 through 15 for the entire bus cycle (T1–T4). These lines do not have to be latched by ALE to remain valid. A15–A8 are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge".</td></tr>
<tr><td>A19/S6, A18/S5, A17/S4, A16/S3</td><td>35-38</td><td>O</td><td>Address/Status: During T1, these are the four most significant address lines for memory operations. During I/O operations, these lines are LOW. During memory and I/O operations, status information is available on these lines during T2, T3, Tw, and T4. S6 is always low. The status of the interrupt enable flag bit (S5) is updated at the beginning of each clock cycle. S4 and S3 are encoded as shown.
<table>
<tr><th>S4</th><th>S3</th><th>CHARACTERISTICS</th></tr>
<tr><td>0 (LOW)</td><td>0</td><td>Alternate Data</td></tr>
<tr><td>0</td><td>1</td><td>Stack</td></tr>
<tr><td>1 (HIGH)</td><td>0</td><td>Code or None</td></tr>
<tr><td>1</td><td>1</td><td>Data</td></tr>
<tr><td>S6 is 0 (LOW)</td><td></td><td></td></tr>
</table>
This information indicates which segment register is presently being used for data accessing.
These lines float to 3-state OFF during local bus "hold acknowledge".</td></tr>
<tr><td>$\overline{RD}$</td><td>32</td><td>O</td><td>Read: Read strobe indicates that the processor is performing a memory or I/O read cycle, depending on the state of the IO/$\overline{M}$ pin or S2. This signal is used to read devices which reside on the 8088 local bus. $\overline{RD}$ is active LOW during T2, T3 and Tw of any read cycle, and is guaranteed to remain HIGH in T2 until the 8088 local bus has floated.
This signal floats to 3-state OFF in "hold acknowledge".</td></tr>
<tr><td>READY</td><td>22</td><td>I</td><td>READY: is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The RDY signal from memory or I/O is synchronized by the 8284 clock generator to form READY. This signal is active HIGH. The 8088 READY input is not synchronized. Correct operation is not guaranteed if the set up and hold times are not met.</td></tr>
<tr><td>INTR</td><td>18</td><td>I</td><td>Interrupt Request: is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.</td></tr>
<tr><td>$\overline{TEST}$</td><td>23</td><td>I</td><td>$\overline{TEST}$: input is examined by the "wait for test" instruction. If the $\overline{TEST}$ input is LOW, execution continues, otherwise the processor waits in an "idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.</td></tr>
<tr><td>NMI</td><td>17</td><td>I</td><td>Non-Maskable Interrupt: is an edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software. A transition from a LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.</td></tr>
</table>

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
RESET	21	I	**RESET:** causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. It restarts execution, as described in the instruction set description, when RESET returns LOW. RESET is internally synchronized.
CLK	19	I	**Clock:** provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.
V_{CC}	40		**V_{CC}:** is the +5V ±10% power supply pin.
GND	1, 20		**GND:** are the ground pins.
MN/$\overline{MX}$	33	I	**Minimum/Maximum:** indicates what mode the processor is to operate in. The two modes are discussed in the following sections.

The following pin function descriptions are for the 8088 minimum mode (i.e., MN/MX = V_{CC}). Only the pin functions which are unique to minimum mode are described; all other pin functions are as described above.

IO/$\overline{M}$	28	O	**Status Line:** is an inverted maximum mode $\overline{S2}$. It is used to distinguish a memory access from an I/O access. IO/$\overline{M}$ becomes valid in the T4 preceding a bus cycle and remains valid until the final T4 of the cycle (I/O=HIGH, M=LOW). IO/$\overline{M}$ floats to 3-state OFF in local bus "hold acknowledge".
$\overline{WR}$	29	O	**Write:** strobe indicates that the processor is performing a write memory or write I/O cycle, depending on the state of the IO/$\overline{M}$ signal. WR is active for T2, T3, and Tw of any write cycle. It is active LOW, and floats to 3-state OFF in local bus "hold acknowledge".
$\overline{INTA}$	24	O	**INTA:** is used as a read strobe for interrupt acknowledge cycles. It is active LOW during T2, T3, and Tw of each interrupt acknowledge cycle.
ALE	25	O	**Address Latch Enable:** is provided by the processor to latch the address into the 8282/8283 address latch. It is a HIGH pulse active during clock low of T1 of any bus cycle. Note that ALE is never floated.
DT/$\overline{R}$	27	O	**Data Transmit/Receive:** is needed in a minimum system that desires to use an 8286/8287 data bus transceiver. It is used to control the direction of data flow through the transceiver. Logically, DT/$\overline{R}$ is equivalent to $\overline{S1}$ in the maximum mode, and its timing is the same as for IO/$\overline{M}$ (T=HIGH, R=LOW). This signal floats to 3-state OFF in local "hold acknowledge".
$\overline{DEN}$	26	O	**Data Enable:** is provided as an output enable for the 8286/8287 in a minimum system which uses the transceiver. $\overline{DEN}$ is active LOW during each memory and I/O access, and for $\overline{INTA}$ cycles. For a read or $\overline{INTA}$ cycle, it is active from the middle of T2 until the middle of T4, while for a write cycle, it is active from the beginning of T2 until the middle of T4. $\overline{DEN}$ floats to 3-state OFF during local bus "hold acknowledge".
HOLD, HLDA	30,31	I, O	**HOLD:** indicates that another master is requesting a local bus "hold". To be acknowledged, HOLD must be active HIGH. The processor receiving the "hold" request will issue HLDA (HIGH) as an acknowledgement, in the middle of a T4 or TI clock cycle. Simultaneous with the issuance of HLDA the processor will float the local bus and control lines. After HOLD is detected as being LOW, the processor lowers HLDA, and when the processor needs to run another cycle, it will again drive the local bus and control lines. Hold is not an asynchronous input. External synchronization should be provided if the system cannot otherwise guarantee the set up time.
$\overline{SSO}$	34	O	**Status line:** is logically equivalent to $\overline{SO}$ in the maximum mode. The combination of $\overline{SSO}$, IO/$\overline{M}$ and DT/$\overline{R}$ allows the system to completely decode the current bus cycle status.

IO/$\overline{M}$	DT/$\overline{R}$	$\overline{SSO}$	CHARACTERISTICS
1 (HIGH)	0	0	Interrupt Acknowledge
1	0	1	Read I/O port
1	1	0	Write I/O port
1	1	1	Halt
0 (LOW)	0	0	Code access
0	0	1	Read memory
0	1	0	Write memory
0	1	1	Passive

Table 1. Pin Description (Continued)

The following pin function descriptions are for the 8088, 8228 system in maximum mode (i.e., MN/MX=GND.) Only the pin functions which are unique to maximum mode are described; all other pin functions are as described above.

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>$\overline{S2}$, $\overline{S1}$, $\overline{S0}$</td><td>26-28</td><td>O</td><td> Status: is active during clock high of T4, T1, and T2, and is returned to the passive state (1,1,1) during T3 or during Tw when READY is HIGH. This status is used by the 8288 bus controller to generate all memory and I/O access control signals. Any change by $\overline{S2}$, $\overline{S1}$, or $\overline{S0}$ during T4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T3 or Tw is used to indicate the end of a bus cycle.
<table>
<tr><th>$\overline{S2}$</th><th>$\overline{S1}$</th><th>$\overline{S0}$</th><th>CHARACTERISTICS</th></tr>
<tr><td>0 (LOW)</td><td>0</td><td>0</td><td>Interrupt Acknowledge</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Read I/O port</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Write I/O port</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Halt</td></tr>
<tr><td>1 (HIGH)</td><td>0</td><td>0</td><td>Code access</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Read memory</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Write memory</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Passive</td></tr>
</table>
These signals float to 3-state OFF during "hold acknowledge". During the first clock cycle after RESET becomes active, these signals are active HIGH. After this first clock, they float to 3-state OFF. </td></tr>
<tr><td>$\overline{RQ}/\overline{GT0}$,
$\overline{RQ}/\overline{GT1}$</td><td>30, 31</td><td>I/O</td><td> Request/Grant: pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with $\overline{RQ}/\overline{GT0}$ having higher priority than $\overline{RQ}/\overline{GT1}$. $\overline{RQ}/\overline{GT}$ has an internal pull-up resistor, so may be left unconnected. The request/grant sequence is as follows (See Figure 8):
1. A pulse of one CLK wide from another local bus master indicates a local bus request ("hold") to the 8088 (pulse 1).
2. During a T4 or TI clock cycle, a pulse one clock wide from the 8088 to the requesting master (pulse 2), indicates that the 8088 has allowed the local bus to float and that it will enter the "hold acknowledge" state at the next CLK. The CPU's bus interface unit is disconnected logically from the local bus during "hold acknowledge". The same rules as for HOLD/HOLDA apply as for when the bus is released.
3. A pulse one CLK wide from the requesting master indicates to the 8088 (pulse 3) that the "hold" request is about to end and that the 8088 can reclaim the local bus at the next CLK. The CPU then enters T4.
Each master-master exchange of the local bus is a sequence of three pulses. There must be one idle CLK cycle after each bus exchange. Pulses are active LOW.
If the request is made while the CPU is performing a memory cycle, it will release the local bus during T4 of the cycle when all the following conditions are met:
1. Request occurs on or before T2.
2. Current cycle is not the low bit of a word.
3. Current cycle is not the first acknowledge of an interrupt acknowledge sequence.
4. A locked instruction is not currently executing.
If the local bus is idle when the request is made the two possible events will follow:
1. Local bus will be released during the next clock.
2. A memory cycle will start within 3 clocks. Now the four rules for a currently active memory cycle apply with condition number 1 already satisfied. </td></tr>
</table>

Table 1. Pin Description (Continued)

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>$\overline{\text{LOCK}}$</td><td>29</td><td>O</td><td>$\overline{\text{LOCK}}$: indicates that other system bus masters are not to gain control of the system bus while $\overline{\text{LOCK}}$ is active (LOW). The $\overline{\text{LOCK}}$ signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW, and floats to 3-state off in "hold acknowledge".</td></tr>
<tr><td>QS1, QS0</td><td>24, 25</td><td>O</td><td>Queue Status: provide status to allow external tracking of the internal 8088 instruction queue.
The queue status is valid during the CLK cycle after which the queue operation is performed.
<table>
<tr><th>QS1</th><th>QS0</th><th>CHARACTERISTICS</th></tr>
<tr><td>0 (LOW)</td><td>0</td><td>No operation</td></tr>
<tr><td>0</td><td>1</td><td>First byte of opcode from queue</td></tr>
<tr><td>1 (HIGH)</td><td>0</td><td>Empty the queue</td></tr>
<tr><td>1</td><td>1</td><td>Subsequent byte from queue</td></tr>
</table></td></tr>
<tr><td>—</td><td>34</td><td>O</td><td>Pin 34 is always high in the maximum mode.</td></tr>
</table>

FUNCTIONAL DESCRIPTION

Memory Organization

The processor provides a 20-bit address to memory which locates the byte being referenced. The memory is organized as a linear array of up to 1 million bytes, addressed as 00000(H) to FFFFF(H). The memory is logically divided into code, data, extra data, and stack segments of up to 64K bytes each, with each segment falling on 16-byte boundaries. (See Figure 3.)

All memory references are made relative to base addresses contained in high speed segment registers. The segment types were chosen based on the addressing needs of programs. The segment register to be selected is automatically chosen according to the rules of the following table. All information in one segment type share the same logical attributes (e.g. code or data). By structuring memory into relocatable areas of similar characteristics and by automatically selecting segment registers, programs are shorter, faster, and more structured.

Word (16-bit) operands can be located on even or odd address boundaries. For address and data operands, the least significant byte of the word is stored in the lower valued address location and the most significant byte in the next higher address location. The BIU will automatically execute two fetch or write cycles for 16-bit operands.

Certain locations in memory are reserved for specific CPU operations. (See Figure 4.) Locations from addresses FFFF0H through FFFFFH are reserved for operations including a jump to the initial system initialization routine. Following RESET, the CPU will always begin execution at location FFFF0H where the jump must be located. Locations 00000H through 003FFH are reserved for interrupt operations. Four-byte pointers consisting of a 16-bit segment address and a 16-bit offset address direct program flow to one of the 256 possible interrupt service routines. The pointer elements are assumed to have been stored at their respective places in reserved memory prior to the occurrence of interrupts.

Minimum and Maximum Modes

The requirements for supporting minimum and maximum 8088 systems are sufficiently different that they cannot be done efficiently with 40 uniquely defined pins. Consequently, the 8088 is equipped with a strap pin (MN/$\overline{MX}$) which defines the system configuration. The definition of a certain subset of the pins changes, dependent on the condition of the strap pin. When the MN/$\overline{MX}$ pin is strapped to GND, the 8088 defines pins 24 through 31 and 34 in maximum mode. When the MN/$\overline{MX}$ pin is strapped to V_{CC}, the 8088 generates bus control signals itself on pins 24 through 31 and 34.

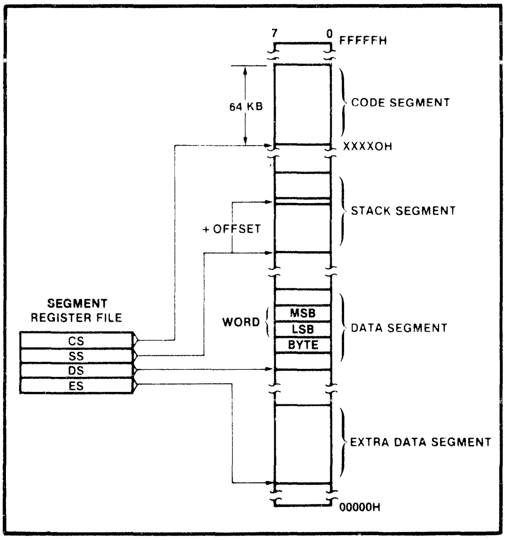


Figure 3. Memory Organization

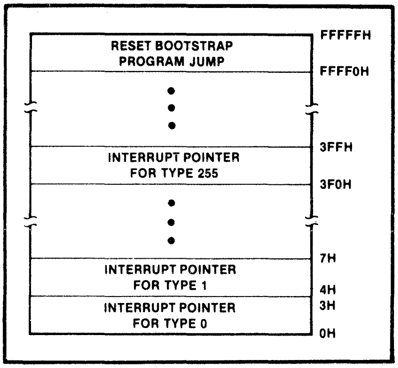


Figure 4. Reserved Memory Locations

Memory Reference Need	Segment Register Used	Segment Selection Rule
Instructions	CODE (CS)	Automatic with all instruction prefetch.
Stack	STACK (SS)	All stack pushes and pops. Memory references relative to BP base register except data references.
Local Data	DATA (DS)	Data references when: relative to stack, destination of string operation, or explicitly overridden.
External (Global) Data	EXTRA (ES)	Destination of string operations: Explicitly selected using a segment override.

The minimum mode 8088 can be used with either a multiplexed or demultiplexed bus. The multiplexed bus configuration is compatible with the MCS-85™ multiplexed bus peripherals (8155, 8156, 8355, 8755A, and 8185). This configuration (See Figure 5) provides the user with a minimum chip count system. This architecture provides the 8088 processing power in a highly integrated form.

The demultiplexed mode requires one latch (for 64K addressability) or two latches (for a full megabyte of addressing). A third latch can be used for buffering if the address bus loading requires it. An 8286 or 8287 transceiver can also be used if data bus buffering is required. (See Figure 6.) The 8088 provides $\overline{DEN}$ and DT/$\overline{R}$ to control the transceiver, and ALE to latch the addresses. This configuration of the minimum mode provides the standard demultiplexed bus structure with heavy bus buffering and relaxed bus timing requirements.

The maximum mode employs the 8288 bus controller. (See Figure 7.) The 8288 decodes status lines $\overline{S0}$, $\overline{S1}$, and $\overline{S2}$, and provides the system with all bus control signals. Moving the bus control to the 8288 provides better source and sink current capability to the control lines, and frees the 8088 pins for extended large system features. Hardware lock, queue status, and two request/grant interfaces are provided by the 8088 in maximum mode. These features allow co-processors in local bus and remote bus configurations.

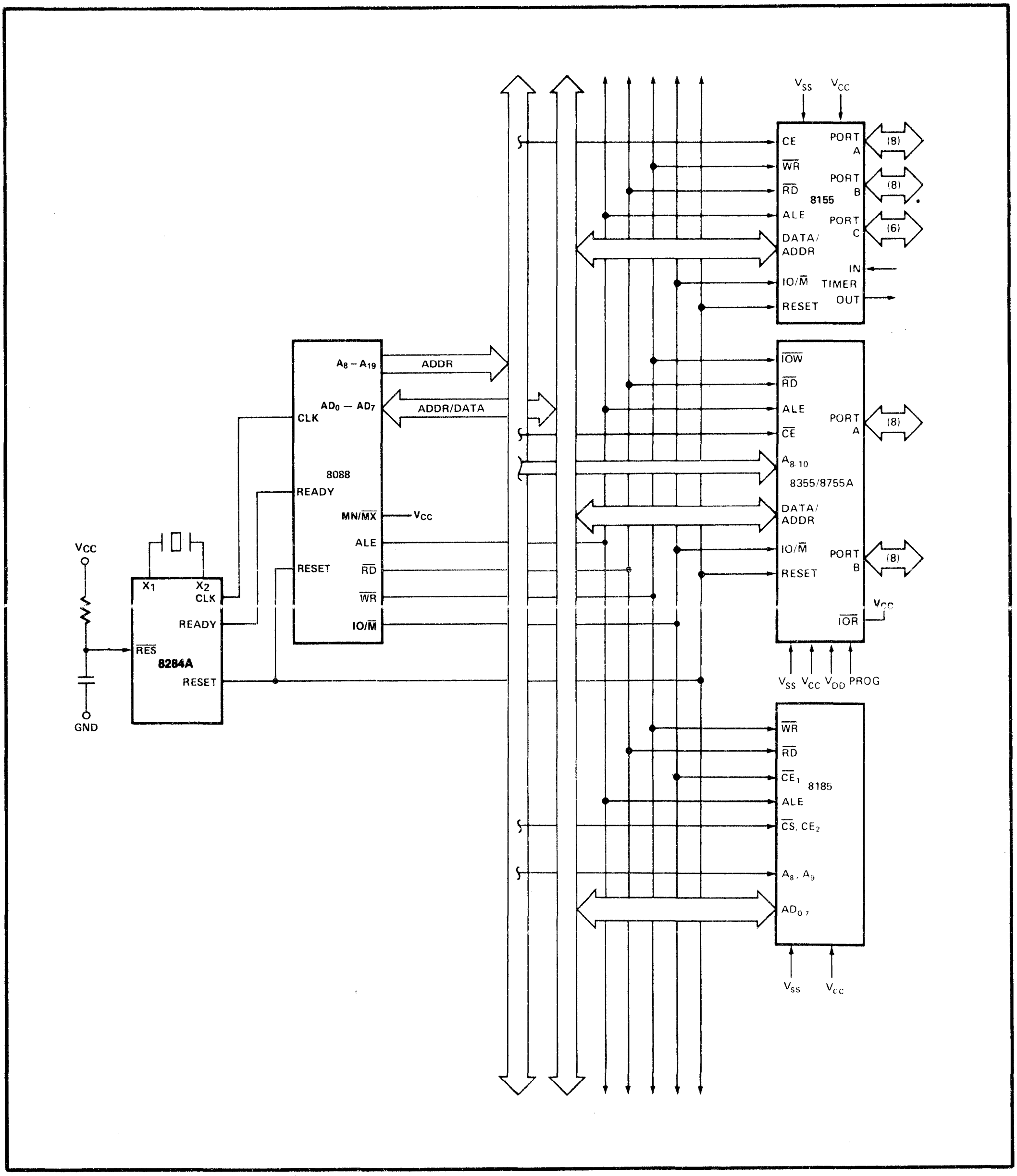


Figure 5. Multiplexed Bus Configuration

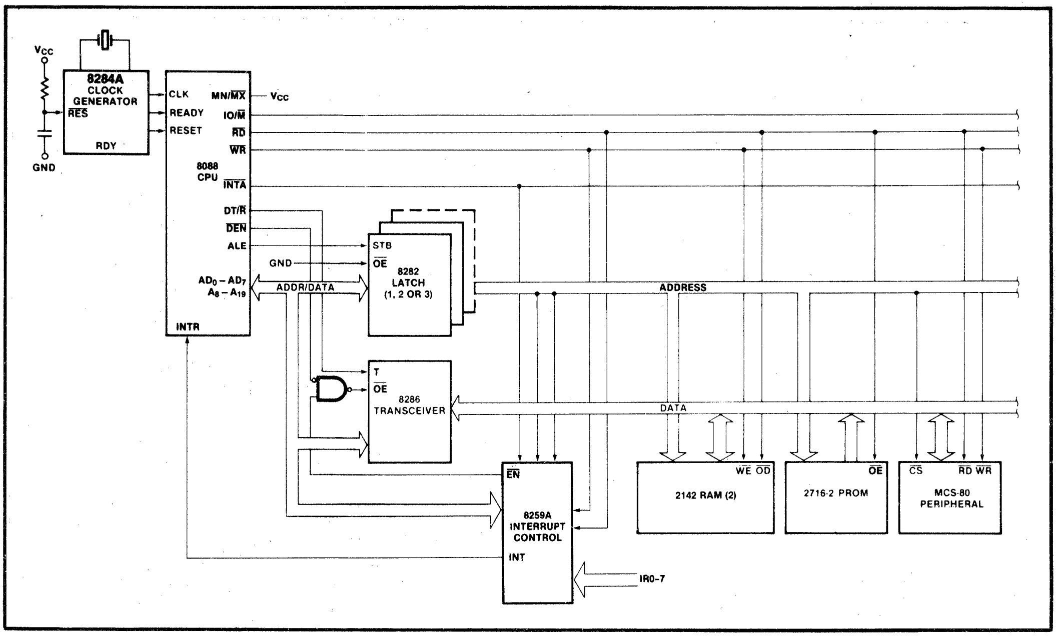


Figure 6. Demultiplexed Bus Configuration

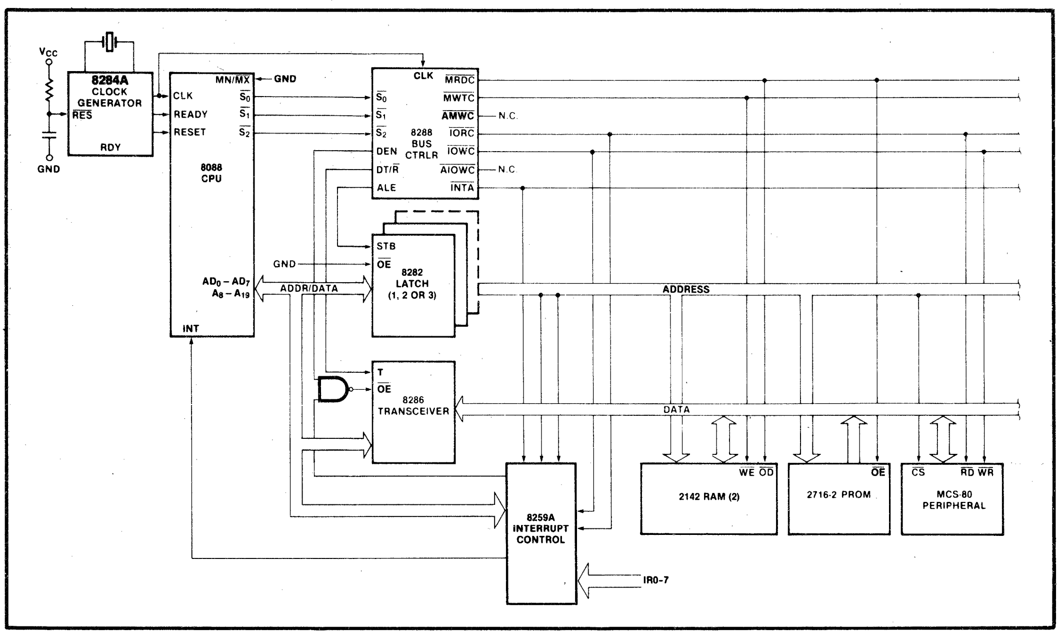


Figure 7. Fully Buffered System Using Bus Controller

Bus Operation

The 8088 address/data bus is broken into three parts — the lower eight address/data bits (AD0–AD7), the middle eight address bits (A8–A15), and the upper four address bits (A16–A19). The address/data bits and the highest four address bits are time multiplexed. This technique provides the most efficient use of pins on the processor, permitting the use of a standard 40 lead package. The middle eight address bits are not multiplexed, i.e. they remain valid throughout each bus cycle. In addition, the bus can be demultiplexed at the processor with a single address latch if a standard, non-multiplexed bus is desired for the system.

Each processor bus cycle consists of at least four CLK cycles. These are referred to as T1, T2, T3, and T4. (See Figure 8). The address is emitted from the processor during T1 and data transfer occurs on the bus during T3 and T4. T2 is used primarily for changing the direction of the bus during read operations. In the event that a "NOT READY" indication is given by the addressed device,

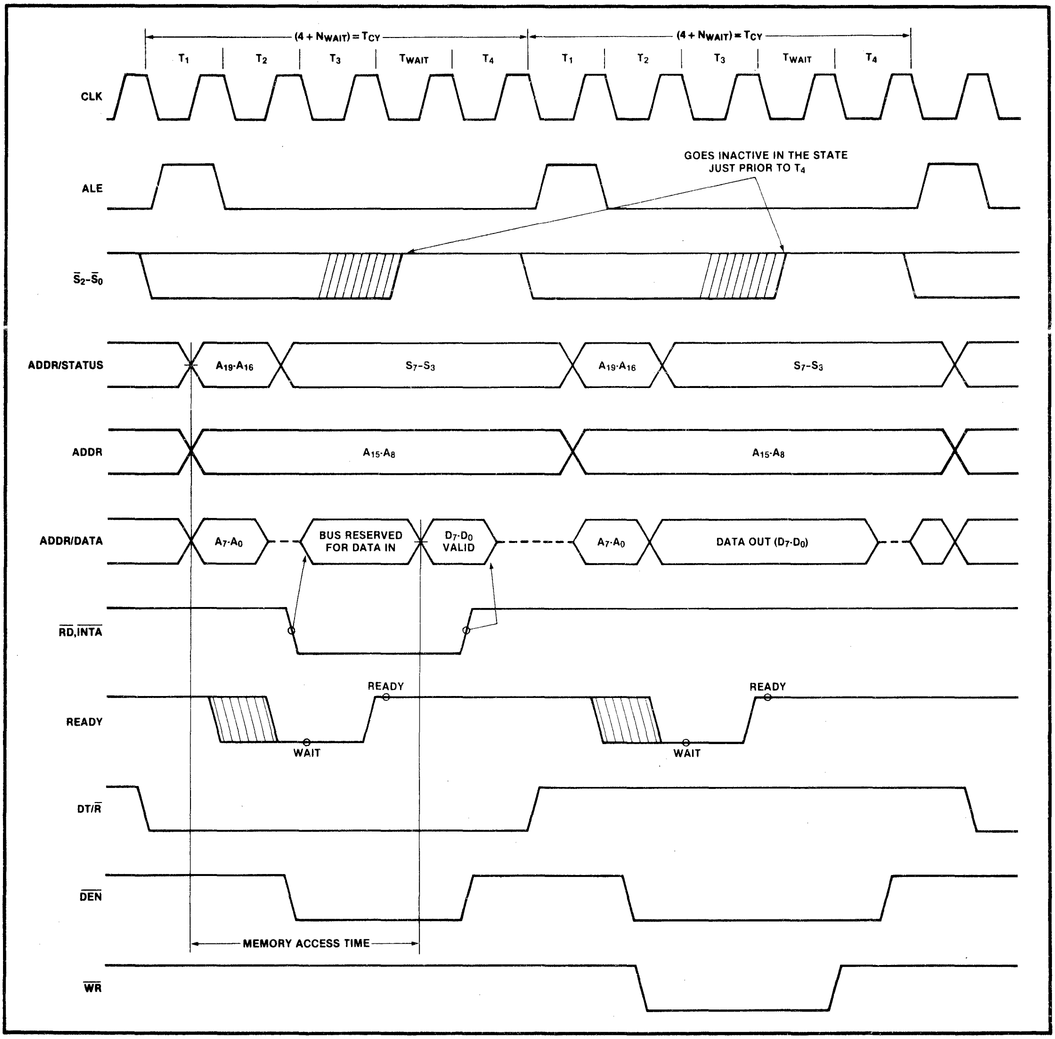


Figure 8. Basic System Timing

"wait" states (Tw) are inserted between T3 and T4. Each inserted "wait" state is of the same duration as a CLK cycle. Periods can occur between 8088 driven bus cycles. These are referred to as "idle" states (Ti), or inactive CLK cycles. The processor uses these cycles for internal housekeeping.

During T1 of any bus cycle, the ALE (address latch enable) signal is emitted (by either the processor or the 8288 bus controller, depending on the MN/$\overline{MX}$ strap). At the trailing edge of this pulse, a valid address and certain status information for the cycle may be latched.

Status bits $\overline{S0}$, $\overline{S1}$, and $\overline{S2}$ are used by the bus controller, in maximum mode, to identify the type of bus transaction according to the following table:

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	CHARACTERISTICS
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1 (HIGH)	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

Status bits S3 through S6 are multiplexed with high order address bits and are therefore valid during T2 through T4. S3 and S4 indicate which segment register was used for this bus cycle in forming the address according to the following table:

S_4	S_3	CHARACTERISTICS
0 (LOW)	0	Alternate Data (extra segment)
0	1	Stack
1 (HIGH)	0	Code or None
1	1	Data

S5 is a reflection of the PSW interrupt enable bit. S6 is always equal to 0.

I/O Addressing

In the 8088, I/O operations can address up to a maximum of 64K I/O registers. The I/O address appears in the same format as the memory address on bus lines A15-A0. The address lines A19-A16 are zero in I/O operations. The variable I/O instructions, which use register DX as a pointer, have full address capability, while the direct I/O instructions directly address one or two of the 256 I/O byte locations in page 0 of the I/O address space. I/O ports are addressed in the same manner as memory locations.

Designers familiar with the 8085 or upgrading an 8085 design should note that the 8085 addresses I/O with an 8-bit address on both halves of the 16-bit address bus. The 8088 uses a full 16-bit address on its lower 16 address lines.

EXTERNAL INTERFACE

Processor Reset and Initialization

Processor initialization or start up is accomplished with activation (HIGH) of the RESET pin. The 8088 RESET is required to be HIGH for greater than four clock cycles. The 8088 will terminate operations on the high-going edge of RESET and will remain dormant as long as RESET is HIGH. The low-going transition of RESET triggers an internal reset sequence for approximately 7 clock cycles. After this interval the 8088 operates normally, beginning with the instruction in absolute location FFFF0H. (See Figure 4.) The RESET input is internally synchronized to the processor clock. At initialization, the HIGH to LOW transition of RESET must occur no sooner than 50 μs after power up, to allow complete initialization of the 8088.

If INTR is asserted sooner than nine clock cycles after the end of RESET, the processor may execute one instruction before responding to the interrupt.

All 3-state outputs float to 3-state OFF during RESET. Status is active in the idle state for the first clock after RESET becomes active and then floats to 3-state OFF.

Interrupt Operations

Interrupt operations fall into two classes: software or hardware initiated. The software initiated interrupts and software aspects of hardware interrupts are specified in the instruction set description in the iAPX 88 book or the iAPX 86,88 User's Manual. Hardware interrupts can be classified as nonmaskable or maskable.

Interrupts result in a transfer of control to a new program location. A 256 element table containing address pointers to the interrupt service program locations resides in absolute locations 0 through 3FFH (see Figure 4), which are reserved for this purpose. Each element in the table is 4 bytes in size and corresponds to an interrupt "type." An interrupting device supplies an 8-bit type number, during the interrupt acknowledge sequence, which is used to vector through the appropriate element to the new interrupt service program location.

Non-Maskable Interrupt (NMI)

The processor provides a single non-maskable interrupt (NMI) pin which has higher priority than the maskable interrupt request (INTR) pin. A typical use would be to activate a power failure routine. The NMI is edge-triggered on a LOW to HIGH transition. The activation of this pin causes a type 2 interrupt.

NMI is required to have a duration in the HIGH state of greater than two clock cycles, but is not required to be synchronized to the clock. Any higher going transition of NMI is latched on-chip and will be serviced at the end of the current instruction or between whole moves (2 bytes in the case of word moves) of a block type instruction. Worst case response to NMI would be for multiply, divide, and variable shift instructions. There is no specification on the occurrence of the low-going edge; it may occur before, during, or after the servicing of NMI. Another high-going edge triggers another response if it occurs after the start of the NMI procedure. The signal must be free of logical spikes in general and be free of bounces on the low-going edge to avoid triggering extraneous responses.

Maskable Interrupt (INTR)

The 8088 provides a single interrupt request input (INTR) which can be masked internally by software with the resetting of the interrupt enable (IF) flag bit. The interrupt request signal is level triggered. It is internally synchronized during each clock cycle on the high-going edge of CLK. To be responded to, INTR must be present (HIGH) during the clock period preceding the end of the current instruction or the end of a whole move for a block type instruction. During interrupt response sequence, further interrupts are disabled. The enable bit is reset as part of the response to any interrupt (INTR, NMI, software interrupt, or single step), although the FLAGS register which is automatically pushed onto the stack reflects the state of the processor prior to the interrupt. Until the old FLAGS register is restored, the enable bit will be zero unless specifically set by an instruction.

During the response sequence (See Figure 9), the processor executes two successive (back to back) interrupt acknowledge cycles. The 8088 emits the LOCK signal (maximum mode only) from T2 of the first bus cycle until T2 of the second. A local bus "hold" request will not be honored until the end of the second bus cycle. In the second bus cycle, a byte is fetched from the external interrupt system (e.g., 8259A PIC) which identifies the source (type) of the interrupt. This byte is multiplied by four and used as a pointer into the interrupt vector lookup table. An INTR signal left HIGH will be continually responded to within the limitations of the enable bit and sample period. The interrupt return instruction includes a flags pop which returns the status of the original interrupt enable bit when it restores the flags.

HALT

When a software HALT instruction is executed, the processor indicates that it is entering the HALT state in one of two ways, depending upon which mode is strapped. In minimum mode, the processor issues ALE, delayed by one clock cycle, to allow the system to latch the halt status. Halt status is available on IO/$\overline{M}$, DT/$\overline{R}$, and $\overline{SSO}$. In maximum mode, the processor issues appropriate HALT status on $\overline{S2}$, $\overline{S1}$, and $\overline{S0}$, and the 8288 bus controller issues one ALE. The 8088 will not leave the HALT state when a local bus hold is entered while in HALT. In this case, the processor reissues the HALT indicator at the end of the local bus hold. An interrupt request or RESET will force the 8088 out of the HALT state.

Read/Modify/Write (Semaphore) Operations via LOCK

The LOCK status information is provided by the processor when consecutive bus cycles are required during the execution of an instruction. This allows the processor to perform read/modify/write operations on memory (via the "exchange register with memory" instruction), without another system bus master receiving intervening memory cycles. This is useful in multiprocessor system configurations to accomplish "test and set lock" operations. The $\overline{LOCK}$ signal is activated (LOW) in the clock cycle following decoding of the LOCK prefix instruction. It is deactivated at the end of the last bus cycle of the instruction following the LOCK prefix. While $\overline{LOCK}$ is active, a request on a $\overline{RQ}/\overline{GT}$ pin will be recorded, and then honored at the end of the LOCK.

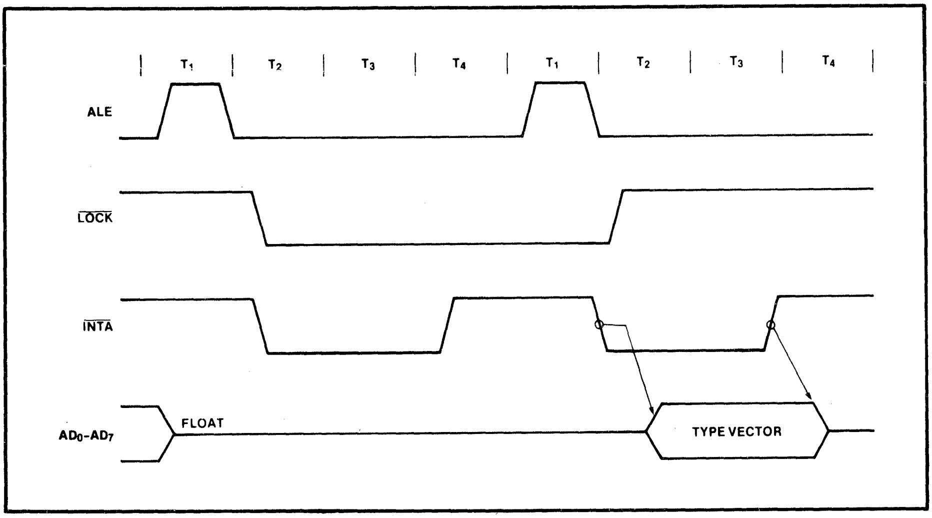


Figure 9. Interrupt Acknowledge Sequence

External Synchronization via TEST

As an alternative to interrupts, the 8088 provides a single software-testable input pin (TEST). This input is utilized by executing a WAIT instruction. The single WAIT instruction is repeatedly executed until the TEST input goes active (LOW). The execution of WAIT does not consume bus cycles once the queue is full.

If a local bus request occurs during WAIT execution, the 8088 3-states all output drivers. If interrupts are enabled, the 8088 will recognize interrupts and process them. The WAIT instruction is then refetched, and reexecuted.

Basic System Timing

In minimum mode, the MN/MX pin is strapped to V_{CC} and the processor emits bus control signals compatible with the 8085 bus structure. In maximum mode, the MN/MX pin is strapped to GND and the processor emits coded status information which the 8288 bus controller uses to generate MULTIBUS compatible bus control signals.

System Timing — Minimum System

(See Figure 8.)

The read cycle begins in T1 with the assertion of the address latch enable (ALE) signal. The trailing (low going) edge of this signal is used to latch the address information, which is valid on the address/data bus (AD0–AD7) at this time, into the 8282/8283 latch. Address lines A8 through A15 do not need to be latched because they remain valid throughout the bus cycle. From T1 to T4 the IO/M signal indicates a memory or I/O operation. At T2 the address is removed from the address/data bus and the bus goes to a high impedance state. The read control signal is also asserted at T2. The read (RD) signal causes the addressed device to enable its data bus drivers to the local bus. Some time later, valid data will be available on the bus and the addressed device will drive the READY line HIGH. When the processor returns the read signal to a HIGH level, the addressed device will again 3-state its bus drivers. If a transceiver (8286/8287) is required to buffer the 8088 local bus, signals DT/R and DEN are provided by the 8088.

A write cycle also begins with the assertion of ALE and the emission of the address. The IO/M signal is again asserted to indicate a memory or I/O write operation. In T2, immediately following the address emission, the processor emits the data to be written into the addressed location. This data remains valid until at least the middle of T4. During T2, T3, and T_W, the processor asserts the write control signal. The write (WR) signal becomes active at the beginning of T2, as opposed to the read, which is delayed somewhat into T2 to provide time for the bus to float.

The basic difference between the interrupt acknowledge cycle and a read cycle is that the interrupt acknowledge (INTA) signal is asserted in place of the read (RD) signal and the address bus is floated. (See Figure 9.) In the second of two successive INTA cycles, a byte of information is read from the data bus, as supplied by the interrupt system logic (i.e. 8259A priority interrupt controller). This byte identifies the source (type) of the interrupt. It is multiplied by four and used as a pointer into the interrupt vector lookup table, as described earlier.

Bus Timing — Medium Complexity Systems

(See Figure 10.)

For medium complexity systems, the MN/MX pin is connected to GND and the 8288 bus controller is added to the system, as well as an 8282/8283 latch for latching the system address, and an 8286/8287 transceiver to allow for bus loading greater than the 8088 is capable of handling. Signals ALE, DEN, and DT/R are generated by the 8288 instead of the processor in this configuration, although their timing remains relatively the same. The 8088 status outputs (S2, S1, and S0) provide type of cycle information and become 8288 inputs. This bus cycle information specifies read (code, data, or I/O), write (data or I/O), interrupt acknowledge, or software halt. The 8288 thus issues control signals specifying memory read or write, I/O read or write, or interrupt acknowledge. The 8288 provides two types of write strobes, normal and advanced, to be applied as required. The normal write strobes have data valid at the leading edge of write. The advanced write strobes have the same timing as read strobes, and hence, data is not valid at the leading edge of write. The 8286/8287 transceiver receives the usual T and OE inputs from the 8288's DT/R and DEN outputs.

The pointer into the interrupt vector table, which is passed during the second INTA cycle, can derive from an 8259A located on either the local bus or the system bus. If the master 8289A priority interrupt controller is positioned on the local bus, a TTL gate is required to disable the 8286/8287 transceiver when reading from the master 8259A during the interrupt acknowledge sequence and software "poll"

The 8088 Compared to the 8086

The 8088 CPU is an 8-bit processor designed around the 8086 internal structure. Most internal functions of the 8088 are identical to the equivalent 8086 functions. The 8088 handles the external bus the same way the 8086 does with the distinction of handling only 8 bits at a time. Sixteen-bit operands are fetched or written in two consecutive bus cycles. Both processors will appear identical to the software engineer, with the exception of execution time. The internal register structure is identical and all instructions have the same end result. The differences between the 8088 and 8086 are outlined below. The engineer who is unfamiliar with the 8086 is referred to the iAPX 86, 88 User's Manual, Chapters 2 and 4, for function description and instruction set information. Internally, there are three differences between the 8088 and the 8086. All changes are related to the 8-bit bus interface.

- The queue length is 4 bytes in the 8088, whereas the 8086 queue contains 6 bytes, or three words. The queue was shortened to prevent overuse of the bus by the BIU when prefetching instructions. This was required because of the additional time necessary to fetch instructions 8 bits at a time.
- To further optimize the queue, the prefetching algorithm was changed. The 8088 BIU will fetch a new instruction to load into the queue each time there is a 1 byte hole (space available) in the queue. The 8086 waits until a 2-byte space is available.
- The internal execution time of the instruction set is affected by the 8-bit interface. All 16-bit fetches and writes from/to memory take an additional four clock cycles. The CPU is also limited by the speed of instruction fetches. This latter problem only occurs when a series of simple operations occur. When the more sophisticated instructions of the 8088 are being used, the queue has time to fill and the execution proceeds as fast as the execution unit will allow.

The 8088 and 8086 are completely software compatible by virture of their identical execution units. Software that is system dependent may not be completely transferable, but software that is not system dependent will operate equally as well on an 8088 or an 8086.

The hardware interface of the 8088 contains the major differences between the two CPUs. The pin assignments are nearly identical, however, with the following functional changes:

- A8–A15 — These pins are only address outputs on the 8088. These address lines are latched internally and remain valid throughout a bus cycle in a manner similar to the 8085 upper address lines.
- $\overline{BHE}$ has no meaning on the 8088 and has been eliminated.
- $\overline{SSO}$ provides the $\overline{SO}$ status information in the minimum mode. This output occurs on pin 34 in minimum mode only. DT/$\overline{R}$, IO/$\overline{M}$, and $\overline{SSO}$ provide the complete bus status in minimum mode.
- IO/$\overline{M}$ has been inverted to be compatible with the MCS-85 bus structure.
- ALE is delayed by one clock cycle in the minimum mode when entering HALT, to allow the status to be latched with ALE.

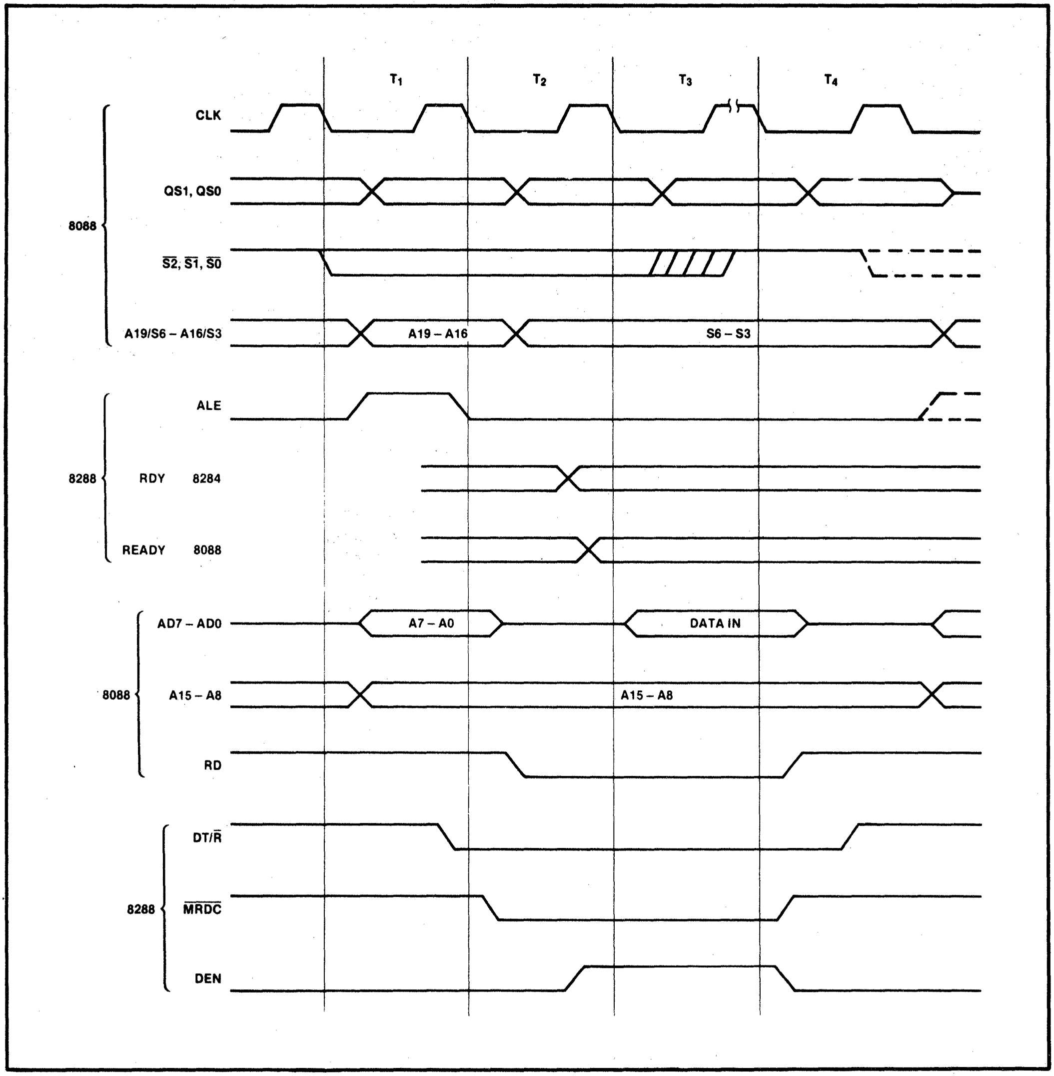


Figure 10. Medium Complexity System Timing

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias.........0°C to 70°C
Storage Temperature............. −65°C to +150°C
Voltage on Any Pin with
  Respect to Ground.................. −1.0 to +7V
Power Dissipation .... 2.5 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

(8088: T_A = 0°C to 70°C, V_{CC} = 5V ±10%)*
(8088-2: T_A = 0°C to 70°C, V_{CC} = 5V ±5%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	(See note 1)
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	(See note 1,2)
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.0 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{CC}	Power Supply Current: 8088 8088-2 P8088		340 350 250	mA	T_A = 25°C
I_{LI}	Input Leakage Current		±10	μA	0V ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}
V_{CL}	Clock Input Low Voltage	−0.5	+0.6	V	
V_{CH}	Clock Input High Voltage	3.9	V_{CC}+1.0	V	
C_{IN}	Capacitance if Input Buffer (All input except AD_0-AD_7, RQ/GT		15	pF	fc = 1 MHz
C_{IO}	Capacitance of I/O Buffer (AD_0-AD_7, RQ/GT		15	pF	fc = 1 MHz

*Note: For Extended Temperature EXPRESS V_{CC} = 5V ± 5%

Note 1: V_{IL} tested with MN/$\overline{MX}$ Pin = 0V
V_{IH} tested with MN/$\overline{MX}$ Pin = 5V
MN/$\overline{MX}$ Pin is a strap Pin
Note 2: Not applicable to $\overline{RQ}/\overline{GT0}$ and $\overline{RQ}/\overline{GT1}$ Pins (Pin 30 and 31)

A.C. CHARACTERISTICS (8088: T_A = 0°C to 70°C, V_{CC} = 5V ±10%)* (8088-2: T_A = 0°C to 70°C, V_{CC} = 5V ±5%)

MINIMUM COMPLEXITY SYSTEM TIMING REQUIREMENTS

		8088		8088-2			
Symbol	Parameter	Min.	Max.	Min.	Max.	Units	Test Conditions
TCLCL	CLK Cycle Period	200	500	125	500	ns	
TCLCH	CLK Low Time	118		68		ns	
TCHCL	CLK High Time	69		44		ns	
TCH1CH2	CLK Rise Time		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		20		ns	
TCLDX	Data in Hold Time	10		10		ns	
TR1VCL	RDY Setup Time into 8284 (See Notes 1, 2)	35		35		ns	
TCLR1X	RDY Hold Time into 8284 (See Notes 1, 2)	0		0		ns	
TRYHCH	READY Setup Time into 8088	118		68		ns	
TCHRYX	READY Hold Time into 8088	30		20		ns	
TRYLCL	READY Inactive to CLK (See Note 3)	−8		−8		ns	
THVCH	HOLD Setup Time	35		20		ns	
TINVCH	INTR, NMI, $\overline{TEST}$ Setup Time (See Note 2)	30		15		ns	
TILIH	Input Rise Time (Except CLK)		20		20	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		12		12	ns	From 2.0V to 0.8V

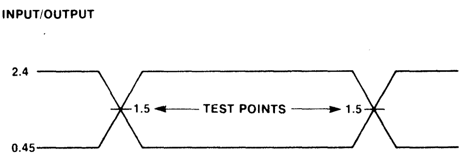
*Note: For Extended Temperature EXPRESS V_{CC} = 5V ± 5%

A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES

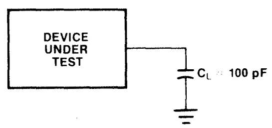
		8088		8088-2			
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
TCLAV	Address Valid Delay	10	110	10	60	ns	
TCLAX	Address Hold Time	10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	TCLAX	50	ns	
TLHLL	ALE Width	TCLCH−20		TCLCH−10		ns	
TCLLH	ALE Active Delay		80		50	ns	
TCHLL	ALE Inactive Delay		85		55	ns	
TLLAX	Address Hold Time to ALE Inactive	TCHCL−10		TCHCL−10		ns	
TCLDV	Data Valid Delay	10	110	10	60	ns	C_L = 20-100 pF for all 8088 Outputs in addition to internal loads
TCHDX	Data Hold Time	10		10		ns	
TWHDX	Data Hold Time After $\overline{WR}$	TCLCH−30		TCLCH−30		ns	
TCVCTV	Control Active Delay 1	10	110	10	70	ns	
TCHCTV	Control Active Delay 2	10	110	10	60	ns	
TCVCTX	Control Inactive Delay	10	110	10	70	ns	
TAZRL	Address Float to READ Active	0		0		ns	
TCLRL	$\overline{RD}$ Active Delay	10	165	10	100	ns	
TCLRH	$\overline{RD}$ Inactive Delay	10	150	10	80	ns	
TRHAV	$\overline{RD}$ Inactive to Next Address Active	TCLCL−45		TCLCL−40		ns	
TCLHAV	HLDA Valid Delay	10	160	10	100	ns	
TRLRH	$\overline{RD}$ Width	2TCLCL−75		2TCLCL−50		ns	
TWLWH	$\overline{WR}$ Width	2TCLCL−60		2TCLCL−40		ns	
TAVAL	Address Valid to ALE Low	TCLCH−60		TCLCH−40		ns	
TOLOH	Output Rise Time		20		20	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		12		12	ns	From 2.0V to 0.8V

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." THE CLOCK IS DRIVEN AT 4.3V AND 0.25V. TIMING MEASUREMENTS ARE MADE AT 1.5V FOR BOTH A LOGIC "1" AND "0."

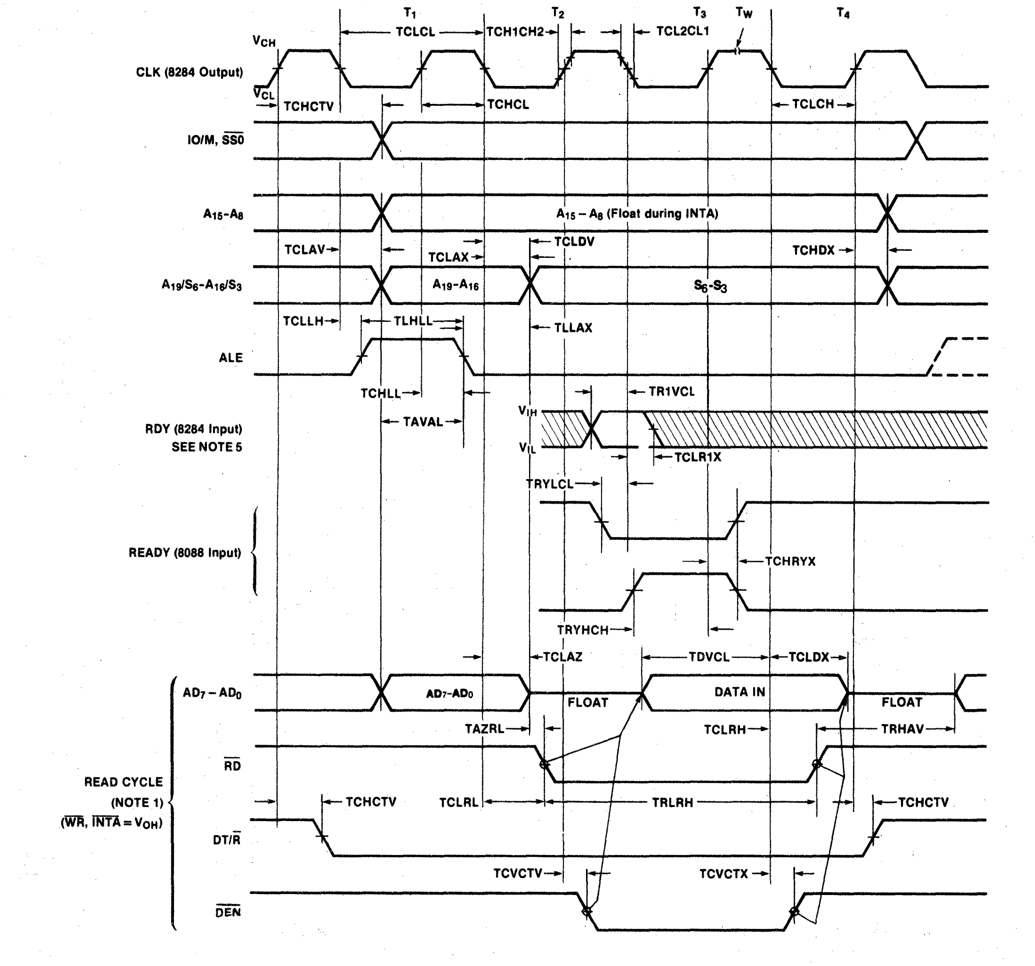
A.C. TESTING LOAD CIRCUIT



C_L INCLUDES JIG CAPACITANCE

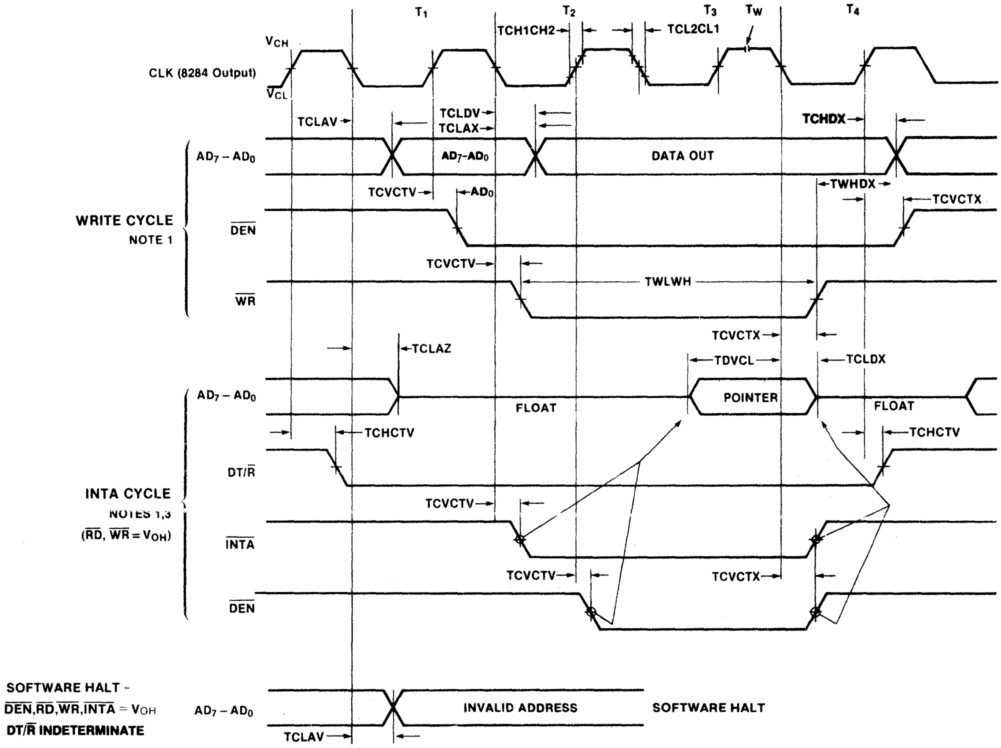
WAVEFORMS

BUS TIMING—MINIMUM MODE SYSTEM



WAVEFORMS (Continued)

BUS TIMING—MINIMUM MODE SYSTEM (Continued)



NOTES:
1. ALL SIGNALS SWITCH BETWEEN V_{OH} AND V_{OL} UNLESS OTHERWISE SPECIFIED.
2. RDY IS SAMPLED NEAR THE END OF T_2, T_3, T_W TO DETERMINE IF T_W MACHINES STATES ARE TO BE INSERTED.
3. TWO INTA CYCLES RUN BACK-TO-BACK. THE 8088 LOCAL ADDR/DATA BUS IS FLOATING DURING BOTH INTA CYCLES. CONTROL SIGNALS ARE SHOWN FOR THE SECOND INTA CYCLE.
4. SIGNALS AT 8284 ARE SHOWN FOR REFERENCE ONLY.
5. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS OTHERWISE NOTED.

A.C. CHARACTERISTICS

MAX MODE SYSTEM (USING 8288 BUS CONTROLLER)
TIMING REQUIREMENTS

		8088		8088-2			
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
TCLCL	CLK Cycle Period	200	500	125	500	ns	
TCLCH	CLK Low Time	118		68		ns	
TCHCL	CLK High Time	69		44		ns	
TCH1CH2	CLK Rise Time		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10	ns	From 3.5V to 1.0V
TDVCL	Data In Setup Time	30		20		ns	
TCLDX	Data In Hold Time	10		10		ns	
TR1VCL	RDY Setup Time into 8284 (See Notes 1, 2)	35		35		ns	
TCLR1X	RDY Hold Time into 8284 (See Notes 1, 2)	0		0		ns	
TRYHCH	READY Setup Time into 8088	118		68		ns	
TCHRYX	READY Hold Time into 8088	30		20		ns	
TRYLCL	READY Inactive to CLK (See Note 4)	−8		−8		ns	
TINVCH	Setup Time for Recognition (INTR, NMI, TEST) (See Note 2)	30		15		ns	
TGVCH	RQ/GT Setup Time	30		15		ns	
TCHGX	RQ Hold Time into 8086	40		30		ns	
TILIH	Input Rise Time (Except CLK)		20		20	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		12		12	ns	From 2.0V to 0.8V

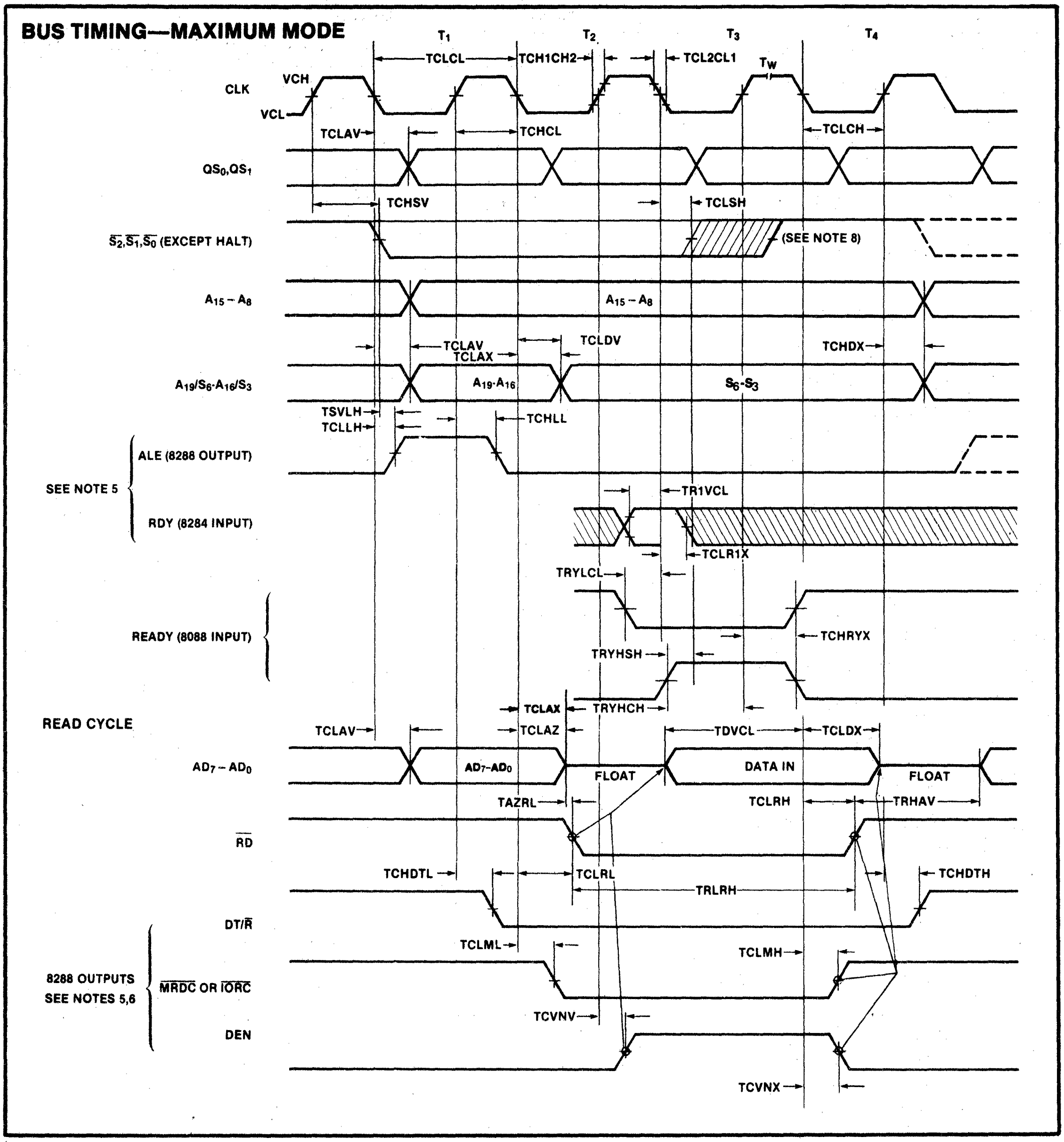
NOTES:
1. Signal at 8284 or 8288 shown for reference only.
2. Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
3. Applies only to T2 state (8 ns into T3 state).
4. Applies only to T2 state (8 ns into T3 state).

A.C. CHARACTERISTICS

TIMING RESPONSES

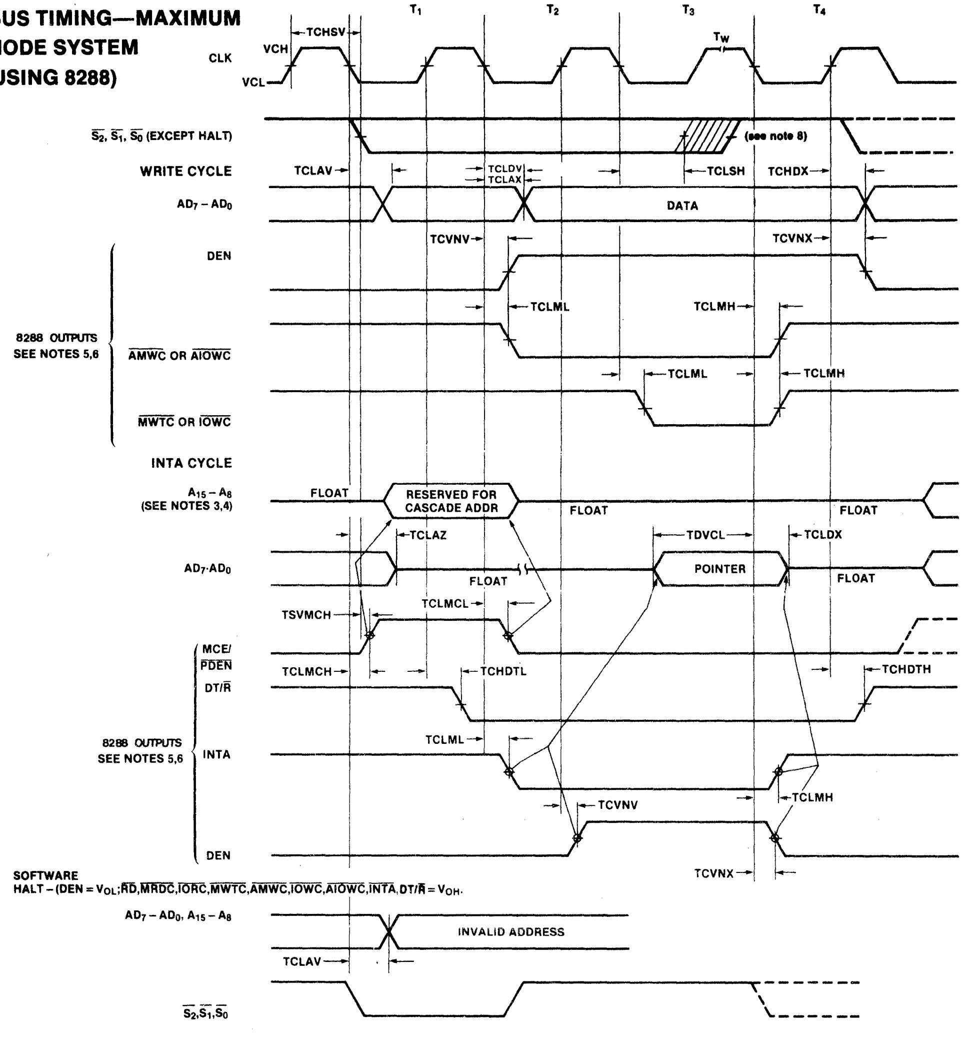
Symbol	Parameter	8088 Min.	8088 Max.	8088-2 Min.	8088-2 Max.	Units	Test Conditions
TCLML	Command Active Delay (See Note 1)	10	35	10	35	ns	
TCLMH	Command Inactive Delay (See Note 1)	10	35	10	35	ns	
TRYHSH	READY Active to Status Passive (See Note 3)		110		65	ns	
TCHSV	Status Active Delay	10	110	10	60	ns	
TCLSH	Status Inactive Delay	10	130	10	70	ns	
TCLAV	Address Valid Delay	10	110	10	60	ns	
TCLAX	Address Hold Time	10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	TCLAX	50	ns	
TSVLH	Status Valid to ALE High (See Note 1)		15		15	ns	
TSVMCH	Status Valid to MCE High (See Note 1)		15		15	ns	
TCLLH	CLK Low to ALE Valid (See Note 1)		15		15	ns	
TCLMCH	CLK Low to MCE High (See Note 1)		15		15	ns	
TCHLL	ALE Inactive Delay (See Note 1)		15		15	ns	
TCLMCL	MCE Inactive Delay (See Note 1)		15		15	ns	C_L = 20-100 pF for all 8088 Outputs in addition to internal loads
TCLDV	Data Valid Delay	10	110	10	60	ns	
TCHDX	Data Hold Time	10		10		ns	
TCVNV	Control Active Delay (See Note 1)	5	45	5	45	ns	
TCVNX	Control Inactive Delay (See Note 1)	10	45	10	45	ns	
TAZRL	Address Float to Read Active	0		0		ns	
TCLRL	RD Active Delay	10	165	10	100	ns	
TCLRH	RD Inactive Delay	10	150	10	80	ns	
TRHAV	RD Inactive to Next Address Active	TCLCL−45		TCLCL−40		ns	
TCHDTL	Direction Control Active Delay (See Note 1)		50		50	ns	
TCHDTH	Direction Control Inactive Delay (See Note 1)		30		30	ns	
TCLGL	GT Active Delay		85		50	ns	
TCLGH	GT Inactive Delay		85		50	ns	
TRLRH	RD Width	2TCLCL−75		2TCLCL−50		ns	
TOLOH	Output Rise Time		20		20	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		12		12	ns	From 2.0V to 0.8V

WAVEFORMS


BUS TIMING—MAXIMUM MODE
T1
T2
T3
T4
TCLCL
TCH1CH2
TCL2CL1
TW
CLK
VCH
VCL
TCLAV
TCHCL
TCLCH
QS0,QS1
TCHSV
TCLSH
S2,S1,S0 (EXCEPT HALT)
(SEE NOTE 8)
A15 – A8
A15 – A8
TCLAV
TCLAX
TCLDV
TCHDX
A19/S6-A16/S3
A19-A16
S6-S3
TSVLH
TCLLH
TCHLL
SEE NOTE 5
ALE (8288 OUTPUT)
RDY (8284 INPUT)
TR1VCL
TCLR1X
TRYLCL
READY (8088 INPUT)
TCHRYX
TRYHSH
TCLAX
TRYHCH
READ CYCLE
TCLAV
TCLAZ
TDVCL
TCLDX
AD7 – AD0
AD7-AD0
FLOAT
DATA IN
FLOAT
TAZRL
TCLRH
TRHAV
RD
TCHDTL
TCLRL
TCHDTH
TRLRH
DT/R
TCLML
TCLMH
8288 OUTPUTS
SEE NOTES 5,6
MRDC OR IORC
TCVNV
DEN
TCVNX

WAVEFORMS (Continued)

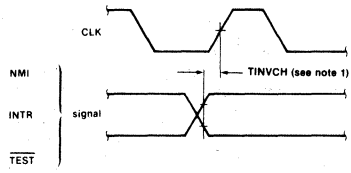
BUS TIMING—MAXIMUM MODE SYSTEM (USING 8288)



NOTES:
1. ALL SIGNALS SWITCH BETWEEN V_{OH} AND V_{OL} UNLESS OTHERWISE SPECIFIED.
2. RDY IS SAMPLED NEAR THE END OF T_2, T_3, T_W TO DETERMINE IF T_W MACHINES STATES ARE TO BE INSERTED.
3. CASCADE ADDRESS IS VALID BETWEEN FIRST AND SECOND INTA CYCLES.
4. TWO INTA CYCLES RUN BACK-TO-BACK. THE 8088 LOCAL ADDR/DATA BUS IS FLOATING DURING BOTH INTA CYCLES. CONTROL FOR POINTER ADDRESS IS SHOWN FOR SECOND INTA CYCLE.
5. SIGNALS AT 8284 OR 8288 ARE SHOWN FOR REFERENCE ONLY.
6. THE ISSUANCE OF THE 8288 COMMAND AND CONTROL SIGNALS ($\overline{MRDC}$, $\overline{MWTC}$, $\overline{AMWC}$, $\overline{IORC}$, $\overline{IOWC}$, $\overline{AIOWC}$, $\overline{INTA}$ AND DEN) LAGS THE ACTIVE HIGH 8288 CEN.
7. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS OTHERWISE NOTED.
8. STATUS INACTIVE IN STATE JUST PRIOR TO T_4.

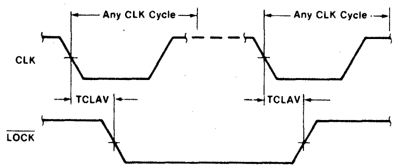
WAVEFORMS (Continued)

ASYNCHRONOUS SIGNAL RECOGNITION

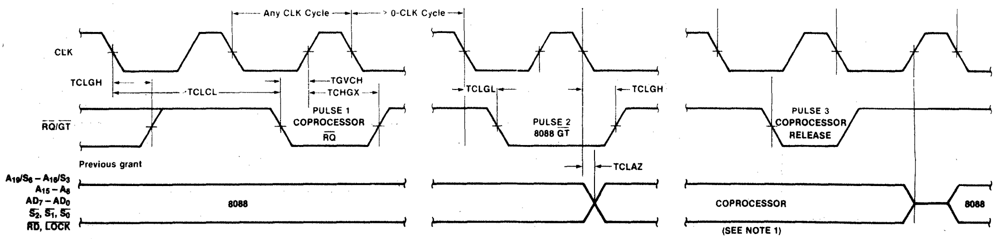


NOTE: 1. SETUP REQUIREMENTS FOR ASYNCHRONOUS SIGNALS ONLY TO GUARANTEE RECOGNITION AT NEXT CLK

BUS LOCK SIGNAL TIMING (MAXIMUM MODE ONLY)

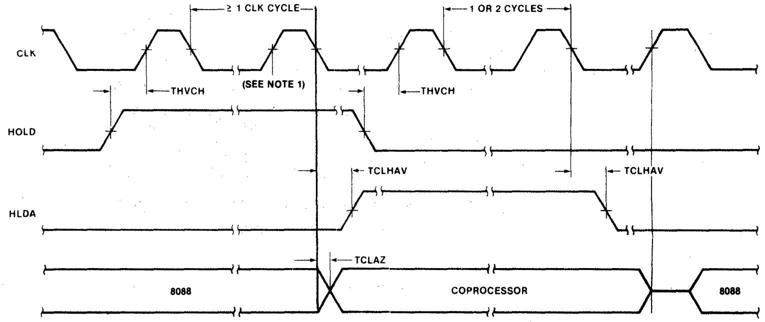


REQUEST/GRANT SEQUENCE TIMING (MAXIMUM MODE ONLY)



NOTE: 1. THE COPROCESSOR MAY NOT DRIVE THE BUSSES OUTSIDE THE REGION SHOWN WITHOUT RISKING CONTENTION.

HOLD/HOLD ACKNOWLEDGE TIMING (MINIMUM MODE ONLY)



iAPX 86/10, 88/10 INSTRUCTION SET SUMMARY

DATA TRANSFER

MOV **Move**:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Register/memory to/from register	1 0 0 0 1 0 d w	mod reg r/m		
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w 1
Immediate to register	1 0 1 1 w reg	data	data if w 1	
Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high	
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high	
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 reg r/m		
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 reg r/m		

PUSH **Push**:		
Register/memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m
Register	0 1 0 1 0 reg	
Segment register	0 0 0 reg 1 1 0	

POP **Pop**:		
Register/memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m
Register	0 1 0 1 1 reg	
Segment register	0 0 0 reg 1 1 1	

XCHG **Exchange**:		
Register/memory with register	1 0 0 0 0 1 1 w	mod reg r/m
Register with accumulator	1 0 0 1 0 reg	

IN=Input from:		
Fixed port	1 1 1 0 0 1 0 w	port
Variable port	1 1 1 0 1 1 0 w	

OUT = Output to:		
Fixed port	1 1 1 0 0 1 1 w	port
Variable port	1 1 1 0 1 1 1 w	
XLAT=Translate byte to AL	1 1 0 1 0 1 1 1	
LEA=Load EA to register	1 0 0 0 1 1 0 1	mod reg r/m
LDS=Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m
LES=Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m
LAHF=Load AH with flags	1 0 0 1 1 1 1 1	
SAHF=Store AH into flags	1 0 0 1 1 1 1 0	
PUSHF=Push flags	1 0 0 1 1 1 0 0	
POPF=Pop flags	1 0 0 1 1 1 0 1	

ARITHMETIC

ADD **Add**:				
Reg./memory with register to either	0 0 0 0 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s w 01
Immediate to accumulator	0 0 0 0 0 1 0 w	data	data if w 1	

ADC **Add with carry**:				
Reg./memory with register to either	0 0 0 1 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s w 01
Immediate to accumulator	0 0 0 1 0 1 0 w	data	data if w 1	

INC = **Increment**:		
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m
Register	0 1 0 0 0 reg	
AAA=ASCII adjust for add	0 0 1 1 0 1 1 1	
DAA=Decimal adjust for add	0 0 1 0 0 1 1 1	

SUB = **Subtract**:				
Reg./memory and register to either	0 0 1 0 1 0 d w	mod reg r/m		
Immediate from register/memory	1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s w 01
Immediate from accumulator	0 0 1 0 1 1 0 w	data	data if w 1	

SBB = **Subtract with borrow**				
Reg./memory and register to either	0 0 0 1 1 0 d w	mod reg r/m		
Immediate from register/memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s w 01
Immediate from accumulator	0 0 0 1 1 1 0 w	data	data if w 1	

DEC **Decrement**:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m		
Register	0 1 0 0 1 reg			
NEG Change sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m		

CMP **Compare**:				
Register/memory and register	0 0 1 1 1 0 d w	mod reg r/m		
Immediate with register/memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s w 01
Immediate with accumulator	0 0 1 1 1 1 0 w	data	data if w 1	
AAS ASCII adjust for subtract	0 0 1 1 1 1 1 1			
DAS Decimal adjust for subtract	0 0 1 0 1 1 1 1			
MUL Multiply (unsigned)	1 1 1 1 0 1 1 w	mod 1 0 0 r/m		
IMUL Integer multiply (signed)	1 1 1 1 0 1 1 w	mod 1 0 1 r/m		
AAM ASCII adjust for multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0		
DIV Divide (unsigned)	1 1 1 1 0 1 1 w	mod 1 1 0 r/m		
IDIV Integer divide (signed)	1 1 1 1 0 1 1 w	mod 1 1 1 r/m		
AAD ASCII adjust for divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0		
CBW Convert byte to word	1 0 0 1 1 0 0 0			
CWD Convert word to double word	1 0 0 1 1 0 0 1			

LOGIC

NOT Invert	1 1 1 1 0 1 1 w	mod 0 1 0 r/m
SHL/SAL Shift logical/arithmetic left	1 1 0 1 0 0 v w	mod 1 0 0 r/m
SHR Shift logical right	1 1 0 1 0 0 v w	mod 1 0 1 r/m
SAR Shift arithmetic right	1 1 0 1 0 0 v w	mod 1 1 1 r/m
ROL Rotate left	1 1 0 1 0 0 v w	mod 0 0 0 r/m
ROR Rotate right	1 1 0 1 0 0 v w	mod 0 0 1 r/m
RCL Rotate through carry flag left	1 1 0 1 0 0 v w	mod 0 1 0 r/m
RCR Rotate through carry right	1 1 0 1 0 0 v w	mod 0 1 1 r/m

AND **And**:				
Reg./memory and register to either	0 0 1 0 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w 1
Immediate to accumulator	0 0 1 0 0 1 0 w	data	data if w 1	

TEST **And function to flags, no result**:				
Register/memory and register	1 0 0 0 0 1 0 w	mod reg r/m		
Immediate data and register/memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w 1
Immediate data and accumulator	1 0 1 0 1 0 0 w	data	data if w 1	

OR **Or**:				
Reg./memory and register to either	0 0 0 0 1 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w 1
Immediate to accumulator	0 0 0 0 1 1 0 w	data	data if w 1	

XOR **Exclusive or**:				
Reg./memory and register to either	0 0 1 1 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w 1
Immediate to accumulator	0 0 1 1 0 1 0 w	data	data if w 1	

STRING MANIPULATION

REP=Repeat	1 1 1 1 0 0 1 z
MOVS=Move byte/word	1 0 1 0 0 1 0 w
CMPS=Compare byte/word	1 0 1 0 0 1 1 w
SCAS=Scan byte/word	1 0 1 0 1 1 1 w
LODS=Load byte/wd to AL/AX	1 0 1 0 1 1 0 w
STOS=Stor byte/wd from AL/A	1 0 1 0 1 0 1 w

INSTRUCTION SET SUMMARY (Continued)

CONTROL TRANSFER

CALL = **Call:**	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Direct within segment	1 1 1 0 1 0 0 0	disp-low	disp-high
Indirect within segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m	
Direct intersegment	1 0 0 1 1 0 1 0	offset-low	offset-high
		seg-low	seg-high
Indirect intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m	
JMP = **Unconditional Jump:**			
Direct within segment	1 1 1 0 1 0 0 1	disp-low	disp-high
Direct within segment-short	1 1 1 0 1 0 1 1	disp	
Indirect within segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m	
Direct intersegment	1 1 1 0 1 0 1 0	offset-low	offset-high
		seg-low	seg-high
Indirect intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m	
RET = **Return from CALL:**			
Within segment	1 1 0 0 0 0 1 1		
Within seg. adding immed to SP	1 1 0 0 0 0 1 0	data-low	data-high
Intersegment	1 1 0 0 1 0 1 1		
Intersegment, adding immediate to SP	1 1 0 0 1 0 1 0	data-low	data-high
JE/JZ=Jump on equal/zero	0 1 1 1 0 1 0 0	disp	
JL/JNGE=Jump on less/not greater or equal	0 1 1 1 1 1 0 0	disp	
JLE/JNG=Jump on less or equal/not greater	0 1 1 1 1 1 1 0	disp	
JB/JNAE=Jump on below/not above or equal	0 1 1 1 0 0 1 0	disp	
JBE/JNA=Jump on below or equal/not above	0 1 1 1 0 1 1 0	disp	
JP/JPE=Jump on parity/parity even	0 1 1 1 1 0 1 0	disp	
JO=Jump on overflow	0 1 1 1 0 0 0 0	disp	
JS=Jump on sign	0 1 1 1 1 0 0 0	disp	
JNE/JNZ=Jump on not equal/not zero	0 1 1 1 0 1 0 1	disp	
JNL/JGE=Jump on not less/greater or equal	0 1 1 1 1 1 0 1	disp	
JNLE/JG=Jump on not less or equal/greater	0 1 1 1 1 1 1 1	disp	

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
JNB/JAE Jump on not below/above or equal	0 1 1 1 0 0 1 1	disp
JNBE/JA Jump on not below or equal/above	0 1 1 1 0 1 1 1	disp
JNP/JPO Jump on not par/par odd	0 1 1 1 1 0 1 1	disp
JNO Jump on not overflow	0 1 1 1 0 0 0 1	disp
JNS Jump on not sign	0 1 1 1 1 0 0 1	disp
LOOP Loop CX times	1 1 1 0 0 0 1 0	disp
LOOPZ/LOOPE Loop while zero/equal	1 1 1 0 0 0 0 1	disp
LOOPNZ/LOOPNE Loop while not zero/equal	1 1 1 0 0 0 0 0	disp
JCXZ Jump on CX zero	1 1 1 0 0 0 1 1	disp
INT Interrupt		
Type specified	1 1 0 0 1 1 0 1	type
Type 3	1 1 0 0 1 1 0 0	
INTO Interrupt on overflow	1 1 0 0 1 1 1 0	
IRET Interrupt return	1 1 0 0 1 1 1 1	

PROCESSOR CONTROL

CLC Clear carry	1 1 1 1 1 0 0 0	
CMC Complement carry	1 1 1 1 0 1 0 1	
STC Set carry	1 1 1 1 1 0 0 1	
CLD Clear direction	1 1 1 1 1 1 0 0	
STD Set direction	1 1 1 1 1 1 0 1	
CLI Clear interrupt	1 1 1 1 1 0 1 0	
STI Set interrupt	1 1 1 1 1 0 1 1	
HLT Halt	1 1 1 1 0 1 0 0	
WAIT Wait	1 0 0 1 1 0 1 1	
ESC Escape (to external device)	1 1 0 1 1 x x x	mod x x x r/m
LOCK Bus lock prefix	1 1 1 1 0 0 0 0	

Footnotes:

AL = 8-bit accumulator
AX = 16-bit accumulator
CX = Count register
DS = Data segment
ES = Extra segment
Above/below refers to unsigned value.
Greater = more positive;
Less = less positive (more negative) signed values
if d = 1 then "to" reg; if d = 0 then "from" reg
if w = 1 then word instruction; if w = 0 then byte instruction

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low

if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP
DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

if s:w = 01 then 16 bits of immediate data form the operand
if s:w = 11 then an immediate data byte is sign extended to form the 16-bit operand.
if v = 0 then "count" = 1; if v = 1 then "count" in (CL)
x = don't care
z is used for string primitives for comparison with Z.F FLAG.

SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:

FLAGS = X:X:X:X:(OF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF)

intel

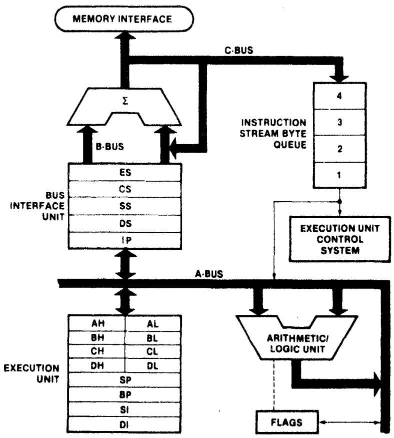
PRELIMINARY

80C88/80C88-2
8-BIT CHMOS MICROPROCESSOR

- **Pin-for-Pin and Functionally Compatible to Industry Standard HMOS 8088**
- **Direct Software Compatibility with 80C86, 8086, 8088**
- **Fully Static Design with Frequency Range from D.C. to:**
  - **—5 MHz for 80C88**
  - **—8 MHz for 80C88-2**
- **Low Power Operation**
  - **—Operating I_{CC} = 10 mA/MHz**
  - **—Standby I_{CCs} = 500 μA max**
- **Bus-Hold Circuitry Eliminates Pull-Up Resistors**
- **Direct Addressing Capability of 1 MByte of Memory**
- **Architecture Designed for Powerful Assembly Language and Efficient High Level Languages**
- **24 Operand Addressing Modes**
- **Byte, Word and Block Operations**
- **8 and 16-Bit Signed and Unsigned Arithmetic**
  - **—Binary or Decimal**
  - **—Multiply and Divide**
- **Will be Available in 40-Lead Plastic DIP and 44-Lead PLCC Packages**

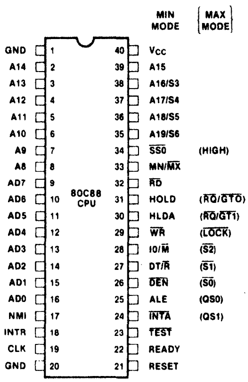
(See Packaging Spec., Order #231369)

The Intel 80C88 is a high performance, CHMOS version of the industry standard HMOS 8088 8-bit CPU. The processor has attributes of both 8 and 16-bit microprocessors. It is available in 5 MHz clock rate and will be available in 8 MHz clock rate in 1st half of 1986. The 80C88 offers two modes of operation: MINimum for small systems and MAXimum for larger applications such as multi-processing. It is available in 40-pin DIP and will be available in 44-pin plastic leaded chip carrier (PLCC) package in 1st quarter of 1986.



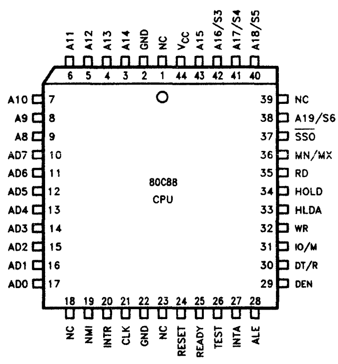
231197-1

Figure 1. iAPX 80C88 CPU Functional Block Diagram



231197-2

Figure 2a. 80C88 40-Lead DIP Configuration



231197-3

Figure 2b. 80C88 44-Lead PLCC Configuration

Order Number: 231197-002

Table 1. Pin Description

The following pin function descriptions are for 80C88 systems in either minimum or maximum mode. The "local bus" in these descriptions is the direct multiplexed bus interface connection to the 80C88 (without regard to additional bus buffers).

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>AD7–AD0</td><td>9–16</td><td>I/O</td><td>ADDRESS DATA BUS: These lines constitute the time multiplexed memory/IO address (T1) and data (T2, T3, Tw, and T4) bus. These lines are active HIGH and float to 3-state OFF(1) during interrupt acknowledge and local bus "hold acknowledge".</td></tr>
<tr><td>A15–A8</td><td>2–8, 39</td><td>O</td><td>ADDRESS BUS: These lines provide address bits 8 through 15 for the entire bus cycle (T1–T4). These lines do not have to be latched by ALE to remain valid. A15–A8 are active HIGH and float to 3-state OFF(1) during interrupt acknowledge and local bus "hold acknowledge".</td></tr>
<tr><td>A19/S6, A18/S5,
A17/S4, A16/S3</td><td>35–38</td><td>O</td><td>ADDRESS/STATUS: During T1, these are the four most significant address lines for memory operations. During I/O operations, these lines are LOW. During memory and I/O operations, status information is available on these lines during T2, T3, Tw, and T4. S6 is always low. The status of the interrupt enable flag bit (S5) is updated at the beginning of each clock cycle. S4 and S3 are encoded as shown.

This information indicates which segment register is presently being used for data accessing.

These lines float to 3-state OFF(1) during local bus "hold acknowledge".
<table>
<tr><th>S4</th><th>S3</th><th>CHARACTERISTICS</th></tr>
<tr><td>0(LOW)</td><td>0</td><td>Alternate Data</td></tr>
<tr><td>0</td><td>1</td><td>Stack</td></tr>
<tr><td>1(HIGH)</td><td>0</td><td>Code or None</td></tr>
<tr><td>1</td><td>1</td><td>Data</td></tr>
<tr><td>S6 is 0(LOW)</td><td></td><td></td></tr>
</table></td></tr>
<tr><td>$\overline{RD}$</td><td>32</td><td>O</td><td>READ: Read strobe indicates that the processor is performing a memory or I/O read cycle, depending on the state of the IO/$\overline{M}$ pin or S2. This signal is used to read devices which reside on the 80C88 local bus. $\overline{RD}$ is active LOW during T2, T3 and Tw of any read cycle, and is guaranteed to remain HIGH in T2 until the 80C88 local bus has floated.

This signal floats to 3-state OFF(1) in "hold acknowledge".</td></tr>
<tr><td>READY</td><td>22</td><td>I</td><td>READY: is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The RDY signal from memory or I/O is synchronized by the 82C84A clock generator to form READY. This signal is active HIGH. The 80C88 READY input is not synchronized. Correct operation is not guaranteed if the set up and hold times are not met.</td></tr>
</table>

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
INTR	18	I	**INTERRUPT REQUEST:** is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.
$\overline{\text{TEST}}$	23	I	**$\overline{\text{TEST}}$:** input is examined by the "wait for test" instruction. If the $\overline{\text{TEST}}$ input is LOW, execution continues, otherwise the processor waits in an "idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.
NMI	17	I	**NON-MASKABLE INTERRUPT:** is an edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software. A transition from a LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.
RESET	21	I	**RESET:** causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. It restarts execution, as described in the instruction set description, when RESET returns LOW. RESET is internally synchronized.
CLK	19	I	**CLOCK:** provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.
V_{CC}	40		**V_{CC}:** is the +5V ±10% power supply pin.
GND	1, 20		**GND:** are the ground pins. Both must be connected.
MN/$\overline{\text{MX}}$	33	I	**MINIMUM/MAXIMUM:** indicates what mode the processor is to operate in. The two modes are discussed in the following sections.

The following pin function descriptions are for the 80C88 minimum mode (i.e., MN/MX = V_{CC}). Only the pin functions which are unique to minimum mode are described; all other pin functions are as described above.

IO/$\overline{\text{M}}$	28	O	**STATUS LINE:** is an inverted maximum mode $\overline{\text{S2}}$. It is used to distinguish a memory access from an I/O access. IO/$\overline{\text{M}}$ becomes valid in the T4 preceding a bus cycle and remains valid until the final T4 of the cycle (I/O = HIGH, M = LOW). IO/$\overline{\text{M}}$ floats to 3-state OFF(1) in local bus "hold acknowledge".
$\overline{\text{WR}}$	29	O	**WRITE:** strobe indicates that the processor is performing a write memory or write I/O cycle, depending on the state of the IO/$\overline{\text{M}}$ signal. WR is active for T2, T3, and Tw of any write cycle. It is active LOW, and floats to 3-state OFF(1) in local bus "hold acknowledge".
$\overline{\text{INTA}}$	24	O	**INTA:** is used as a read strobe for interrupt acknowledge cycles. It is active LOW during T2, T3, and Tw of each interrupt acknowledge cycle.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
ALE	25	O	**ADDRESS LATCH ENABLE:** is provided by the processor to latch the address into an address latch. It is a HIGH pulse active during clock low of T1 of any bus cycle. Note that ALE is never floated.
DT/$\overline{\text{R}}$	27	O	**DATA TRANSMIT/RECEIVE:** is needed in a minimum system that desires to use a data bus transceiver. It is used to control the direction of data flow through the transceiver. Logically, DT/$\overline{\text{R}}$ is equivalent to $\overline{\text{S1}}$ in the maximum mode, and its timing is the same as for IO/$\overline{\text{M}}$ (T = HIGH, R = LOW). This signal floats to 3-state OFF(1) in local "hold acknowledge".
$\overline{\text{DEN}}$	26	O	**DATA ENABLE:** is provided as an output enable for the transceiver in a minimum system which uses the transceiver. $\overline{\text{DEN}}$ is active LOW during each memory and I/O access, and for $\overline{\text{INTA}}$ cycles. For a read or $\overline{\text{INTA}}$ cycle, it is active from the middle of T2 until the middle of T4, while for a write cycle, it is active from the beginning of T2 until the middle of T4. $\overline{\text{DEN}}$ floats to 3-state OFF(1) during local bus "hold acknowledge".
HOLD, HLDA	30, 31	I, O	**HOLD:** indicates that another master is requesting a local bus "hold". To be acknowledged, HOLD must be active HIGH. The processor receiving the "hold" request will issue HLDA (HIGH) as an acknowledgement, in the middle of a T4 or T1 clock cycle. Simultaneous with the issuance of HLDA the processor will float the local bus and control lines. After HOLD is detected as being LOW, the processor lowers HLDA, and when the processor needs to run another cycle, it will again drive the local bus and control lines. Hold is not an asynchronous input. External synchronization should be provided if the system cannot otherwise guarantee the set up time.
$\overline{\text{SSO}}$	34	O	**STATUS LINE:** is logically equivalent to $\overline{\text{SO}}$ in the maximum mode. The combination of $\overline{\text{SSO}}$, IO/$\overline{\text{M}}$ and DT/$\overline{\text{R}}$ allows the system to completely decode the current bus cycle status.

IO/$\overline{\text{M}}$	DT/$\overline{\text{R}}$	$\overline{\text{SSO}}$	CHARACTERISTICS
1(HIGH)	0	0	Interrupt Acknowledge
1	0	1	Read I/O port
1	1	0	Write I/O port
1	1	1	Halt
0(LOW)	0	0	Code access
0	0	1	Read memory
0	1	0	Write memory
0	1	1	Passive

Table 1. Pin Description (Continued)

The following pin function descriptions are for the 80C88/82C88 system in maximum mode (i.e., MN/MX = GND.) Only the pin functions which are unique to maximum mode are described; all other pin functions are as described above.

Symbol	Pin No.	Type	Name and Function
$\overline{S2}$, $\overline{S1}$, $\overline{S0}$	26–28	O	**STATUS:** is active during clock high of T4, T1, and T2, and is returned to the passive state (1,1,1) during T3 or during Tw when READY is HIGH. This status is used by the 82C88 bus controller to generate all memory and I/O access control signals. Any change by $\overline{S2}$, $\overline{S1}$, or $\overline{S0}$ during T4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T3 or Tw is used to indicate the end of a bus cycle. These signals float to 3-state OFF(1) during "hold acknowledge". During the first clock cycle after RESET becomes active, these signals are active HIGH. After this first clock, they float to 3-state OFF.

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	CHARACTERISTICS
0(LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1(HIGH)	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

Symbol	Pin No.	Type	Name and Function
$\overline{RQ}/\overline{GT0}$, $\overline{RQ}/\overline{GT1}$	30, 31	I/O	**REQUEST/GRANT:** pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with $\overline{RQ}/\overline{GT0}$ having higher priority than $\overline{RQ}/\overline{GT1}$. $\overline{RQ}/\overline{GT}$ has an internal pull-up resistor, so may be left unconnected. The request/grant sequence is as follows (see timing diagram): 1. A pulse of one CLK wide from another local bus master indicates a local bus request ("hold") to the 80C88 (pulse 1). 2. During a T4 or T1 clock cycle, a pulse one clock wide from the 80C88 to the requesting master (pulse 2), indicates that the 80C88 has allowed the local bus to float and that it will enter the "hold acknowledge" state at the next CLK. The CPU's bus interface unit is disconnected logically from the local bus during "hold acknowledge". The same rules as for HOLD/HOLDA apply as for when the bus is released. 3. A pulse one CLK wide from the requesting master indicates to the 80C88 (pulse 3) that the "hold" request is about to end and that the 80C88 can reclaim the local bus at the next CLK. The CPU then enters T4.

Table 1. Pin Descriptions (Continued)

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th colspan="3">Name and Function</th></tr>
<tr><td>$\overline{RQ}/\overline{GT0}$,
$\overline{RQ}/\overline{GT1}$</td><td>30, 31</td><td>I/O</td><td colspan="3">Each master-master exchange of the local bus is a sequence of three pulses. There must be one idle CLK cycle after each bus exchange. Pulses are active LOW.

If the request is made while the CPU is performing a memory cycle, it will release the local bus during T4 of the cycle when all the following conditions are met:

1. Request occurs on or before T2.
2. Current cycle is not the low bit of a word.
3. Current cycle is not the first acknowledge of an interrupt acknowledge sequence.
4. A locked instruction is not currently executing.

If the local bus is idle when the request is made the two possible events will follow:

1. Local bus will be released during the next clock.
2. A memory cycle will start within 3 clocks. Now the four rules for a currently active memory cycle apply with condition number 1 already satisfied.</td></tr>
<tr><td>$\overline{LOCK}$</td><td>29</td><td>O</td><td colspan="3">**$\overline{LOCK}$:** indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active (LOW). The $\overline{LOCK}$ signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW, and floats to 3-state OFF(1) in "hold acknowledge".</td></tr>
<tr><td rowspan="3">QS1, QS0</td><td rowspan="3">24, 25</td><td rowspan="3">O</td><td colspan="3">**QUEUE STATUS:** provide status to allow external tracking of the internal 80C88 instruction queue.

The queue status is valid during the CLK cycle after which the queue operation is performed.</td></tr>
<tr><td>**QS1**</td><td>**QS0**</td><td>**CHARACTERISTICS**</td></tr>
<tr><td>0(LOW)
0
1(HIGH)
1</td><td>0
1
0
1</td><td>No operation
First byte of opcode from queue
Empty the queue
Subsequent byte from queue</td></tr>
<tr><td>—</td><td>34</td><td>O</td><td colspan="3">Pin 34 is always high in the maximum mode.</td></tr>
</table>

NOTE:
1. See the section on Bus Hold Circuitry.

FUNCTIONAL DESCRIPTION

STATIC OPERATION

All 80C88 circuitry is of static design. Internal registers, counters and latches are static and require no refresh as with dynamic circuit design. This eliminates the minimum operating frequency restriction placed on other microprocessors. The CMOS 80C88 can operate from DC to the appropriate upper frequency limit. The processor clock may be stopped in either state (high/low) and held there indefinitely. This type of operation is especially useful for system debug or power critical applications.

The 80C88 can be single stepped using only the CPU clock. This state can be maintained as long as is necessary. Single step clock operation allows simple interface circuitry to provide critical information for bringing up your system.

Static design also allows very low frequency operation (down to DC). In a power critical situation, this can provide extremely low power operation since 80C88 power dissipation is directly related to operating frequency. As the system frequency is reduced, so is the operating power until ultimately, at a DC input frequency, the 80C88 power requirement is the standby current (500 μA maximum).

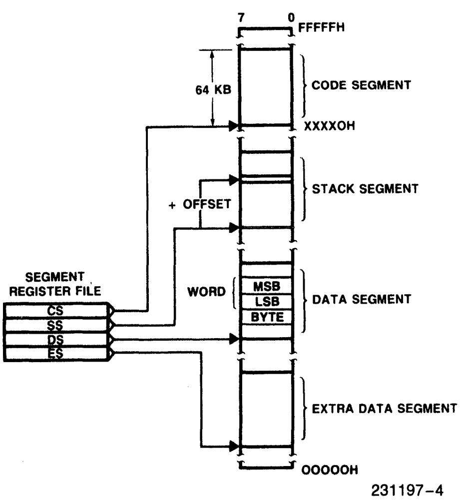


Figure 3. Memory Organization

MEMORY ORGANIZATION

The processor provides a 20-bit address to memory which locates the byte being referenced. The memory is organized as a linear array of up to 1 million bytes, addressed as 00000(H) to FFFFF(H). The memory is logically divided into code, data, extra data, and stack segments of up to 64K bytes each, with each segment falling on 16-byte boundaries. (See Figure 3.)

All memory references are made relative to base addresses contained in high speed segment registers. The segment types were chosen based on the addressing needs of programs. The segment register to be selected is automatically chosen according to the rules of the following table. All information in one segment type share the same logical attributes (e.g. code or data). By structuring memory into relocatable areas of similar characteristics and by automatically selecting segment registers, programs are shorter, faster, and more structured.

Word (16-bit) operands can be located on even or odd address boundaries. For address and data operands, the least significant byte of the word is stored in the lower valued address location and the most significant byte in the next higher address location. The BIU will automatically execute two fetch or write cycles for 16-bit operands.

Certain locations in memory are reserved for specific CPU operations. (See Figure 4.) Locations from addresses FFFF0H through FFFFFH are reserved for operations including a jump to the initial system

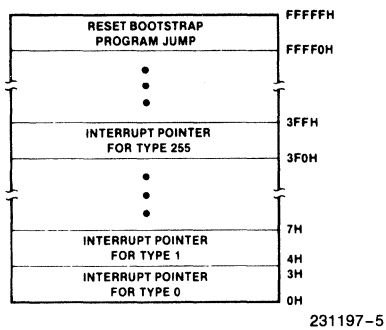


Figure 4. Reserved Memory Locations

Memory Reference Need	Segment Register Used	Segment Selection Rule
Instructions	CODE (CS)	Automatic with all instruction prefetch.
Stack	STACK (SS)	All stack pushes and pops. Memory references relative to BP base register except data references.
Local Data	DATA (DS)	Data references when: relative to stack, destination of string operation, or explicitly overridden.
External (Global) Data	EXTRA (ES)	Destination of string operations: Explicitly selected using a segment override.

initialization routine. Following RESET, the CPU will always begin execution at location FFFF0H where the jump must be located. Locations 00000H through 003FFH are reserved for interrupt operations. Four-byte pointers consisting of a 16-bit segment address and a 16-bit offset address direct program flow to one of the 256 possible interrupt service routines. The pointer elements are assumed to have been stored at their respective places in reserved memory prior to the occurrence of interrupts.

MINIMUM AND MAXIMUM MODES

The requirements for supporting minimum and maximum 80C88 systems are sufficiently different that they cannot be done efficiently with 40 uniquely defined pins. Consequently, the 80C88 is equipped with a strap pin (MN/$\overline{\text{MX}}$) which defines the system configuration. The definition of a certain subset of the pins changes, dependent on the condition of the strap pin. When the MN/$\overline{\text{MX}}$ pin is strapped to GND, the 80C88 defines pins 24 through 31 and 34 in maximum mode. When the MN/$\overline{\text{MX}}$ pin is strapped to V_{CC}, the 80C88 generates bus control signals itself on pins 24 through 31 and 34.

The minimum mode 80C88 can be used with either a multiplexed or demultiplexed bus. The multiplexed bus configuration is compatible with the MCS®-85 multiplexed bus peripherals (8155, 8156, 8355, 8755A, and 8185). This configuration (See Figure 5) provides the user with a minimum chip count system. This architecture provides the 80C88 processing power in a highly integrated form.

The demultiplexed mode requires one latch (for 64k addressability) or two latches (for a full megabyte of addressing). A third latch can be used for buffering if the address bus loading requires it. A transceiver can also be used if data bus buffering is required. (See Figure 6.) The 80C88 provides $\overline{\text{DEN}}$ and DT/$\overline{\text{R}}$ to control the transceiver, and ALE to latch the addresses. This configuration of the minimum mode provides the standard demultiplexed bus structure with heavy bus buffering and relaxed bus timing requirements.

The maximum mode employs the 82C88 bus controller. (See Figure 7.) The 82C88 decodes status lines $\overline{\text{S0}}$, $\overline{\text{S1}}$, and $\overline{\text{S2}}$, and provides the system with all bus control signals. Moving the bus control to the 82C88 provides better source and sink current capability to the control lines, and frees the 80C88 pins for extended large system features. Hardware lock, queue status, and two request/grant interfaces are provided by the 80C88 in maximum mode. These features allow co-processors in local bus and remote bus configurations.

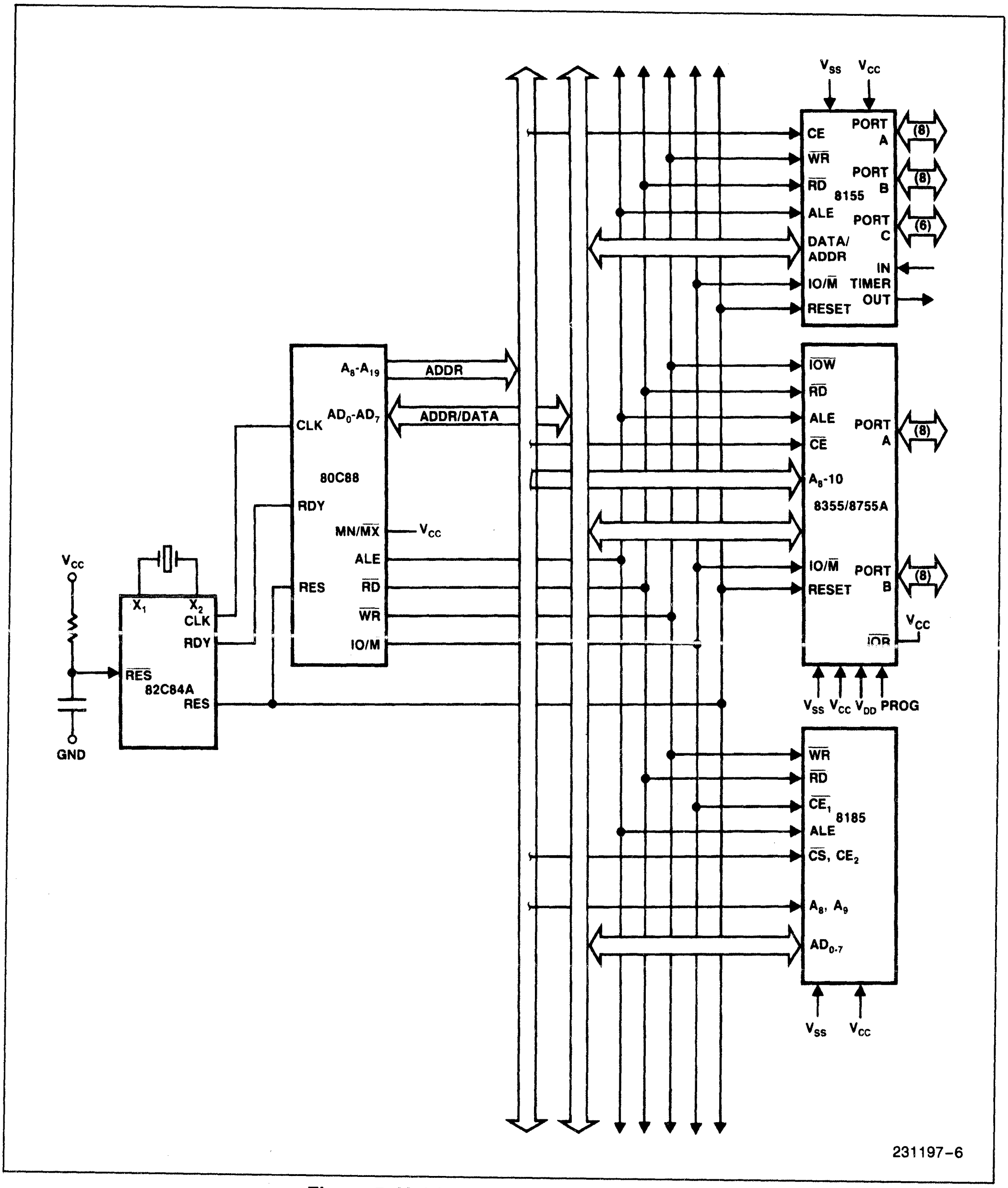


Figure 5. Multiplexed Bus Configuration

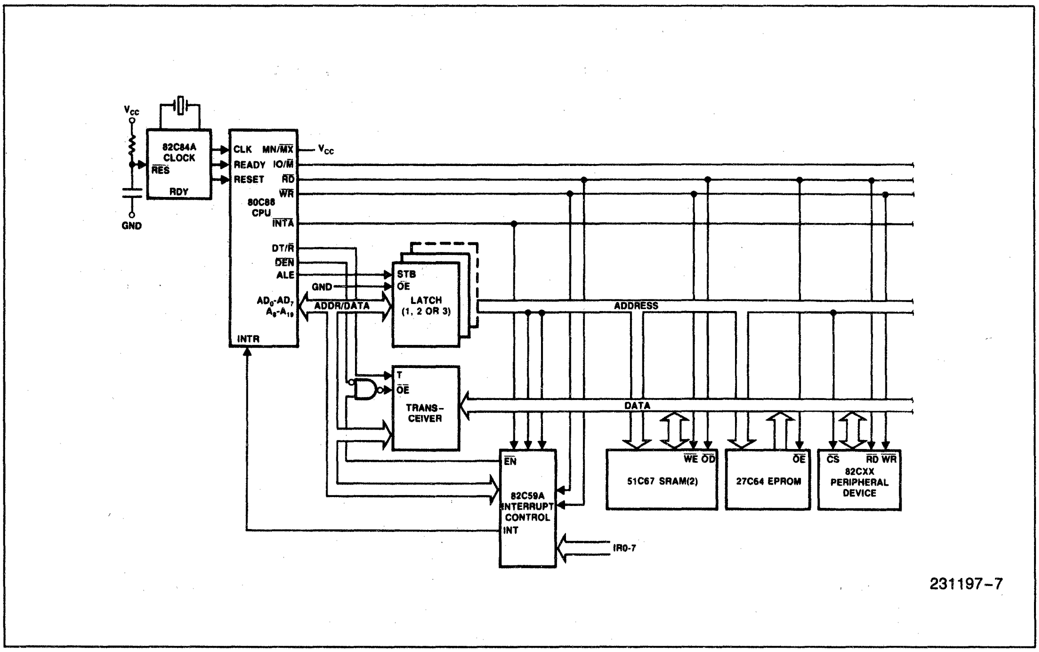


Figure 6. Demultiplexed Bus Configuration

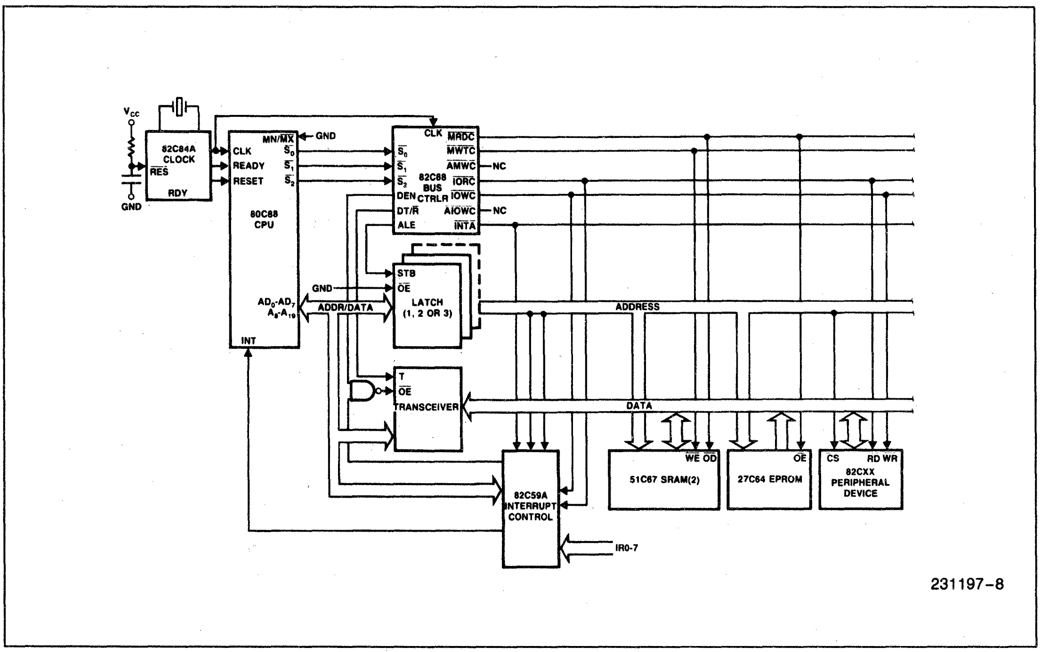


Figure 7. Fully Buffered System Using Bus Controller

Bus Operation

The 80C88 address/data bus is broken into three parts—the lower eight address/data bits (AD0–AD7), the middle eight address bits (A8–A15), and the upper four address bits (A16–A19). The address/data bits and the highest four address bits are time multiplexed. This technique provides the most efficient use of pins on the processor, permitting the use of a standard 40 lead package. The middle eight address bits are not multiplexed, i.e. they remain valid throughout each bus cycle. In addition, the bus can be demultiplexed at the processor with a single address latch if a standard, non-multiplexed bus is desired for the system.

Each processor bus cycle consists of at least four CLK cycles. These are referred to as T1, T2, T3, and T4. (See Figure 8). The address is emitted from the processor during T1 and data transfer occurs on the bus during T3 and T4. T2 is used primarily for changing the direction of the bus during read operations. In the event that a "NOT READY" indication is given by the addressed device, "wait" states (Tw) are inserted between T3 and T4. Each inserted "wait" state is of the same duration as a CLK cycle. Periods can occur between 80C88 driven bus cycles. These are referred to as "idle" states (Ti), or inactive CLK cycles. The processor uses these cycles for internal housekeeping.

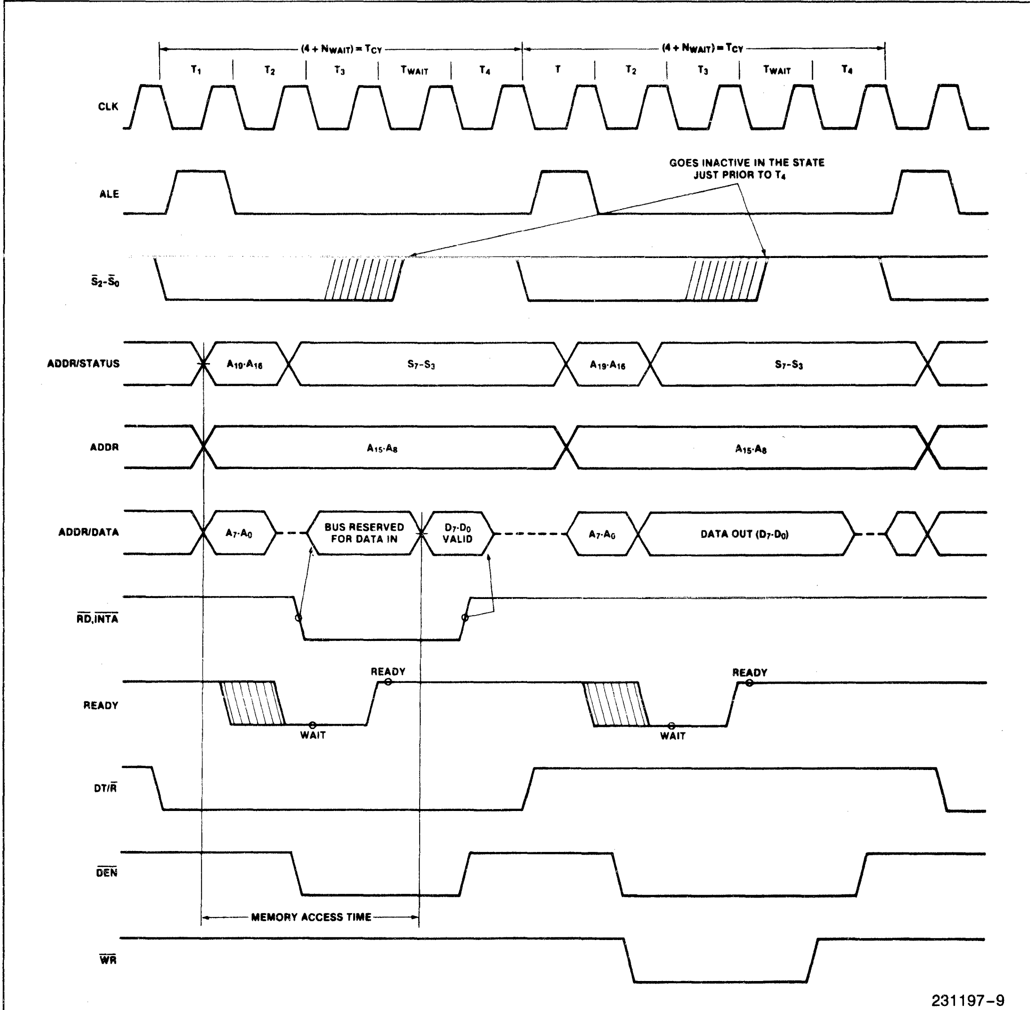


Figure 8. Basic System Timing

During T1 of any bus cycle, the ALE (address latch enable) signal is emitted (by either the processor or the 82C88 bus controller, depending on the MN/$\overline{MX}$ strap). At the trailing edge of this pulse, a valid address and certain status information for the cycle may be latched.

Status bits $\overline{S0}$, $\overline{S1}$, and $\overline{S2}$ are used by the bus controller, in maximum mode, to identify the type of bus transaction according to the following table:

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	CHARACTERISTICS
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1 (HIGH)	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

Status bits S3 through S6 are multiplexed with high order address bits and are therefore valid during T2 through T4. S3 and S4 indicate which segment register was used for this bus cycle in forming the address according to the following table:

S_4	S_3	CHARACTERISTICS
0 (LOW)	0	Alternate Data (extra segment)
0	1	Stack
1 (HIGH)	0	Code or None
1	1	Data

S5 is a reflection of the PSW interrupt enable bit. S6 is always equal to 0.

I/O ADDRESSING

In the 80C88, I/O operations can address up to a maximum of 64k I/O registers. The I/O address appears in the same format as the memory address on bus lines A15–A0. The address lines A19–A16 are zero in I/O operations. The variable I/O instructions, which use register DX as a pointer, have full address capability, while the direct I/O instructions directly address one or two of the 256 I/O byte locations in page 0 of the I/O address space. I/O ports are addressed in the same manner as memory locations.

Designers familiar with the 8085 or upgrading an 8085 design should note that the 8085 addresses I/O with an 8-bit address on both halves of the 16-bit address bus. The 80C88 uses a full 16-bit address on its lower 16 address lines.

EXTERNAL INTERFACE

PROCESSOR RESET AND INITIALIZATION

Processor initialization or start up is accomplished with activation (HIGH) of the RESET pin. The 80C88 RESET is required to be HIGH for greater than four clock cycles. The 80C88 will terminate operations on the high-going edge of RESET and will remain dormant as long as RESET is HIGH. The low-going transition of RESET triggers an internal reset sequence for approximately 7 clock cycles. After this interval the 80C88 operates normally, beginning with the instruction in absolute location FFFF0H. (See Figure 4.) The RESET input is internally synchronized to the processor clock. At initialization, the HIGH to LOW transition of RESET must occur no sooner than 50 μs after power up, to allow complete initialization of the 80C88.

If INTR is asserted sooner than nine clock cycles after the end of RESET, the processor may execute one instruction before responding to the interrupt.

All 3-state outputs float to 3-state OFF during RESET. Status is active in the idle state for the first clock after RESET becomes active and then floats to 3-state OFF.

BUS HOLD CIRCUITRY

To avoid high current conditions caused by floating inputs to CMOS devices and to eliminate the need for pull-up/down resistors, "bus-hold" circuitry has been used on the 80C88 pins 2–16, 26–32, and 34–39 (Figure 9a, 9b). These circuits will maintain the last valid logic state if no driving source is present (i.e. an unconnected pin or a driving source which goes to a high impedance state). To overdrive the "bus hold" circuits, an external driver must be capable of supplying 350 μA minimum sink or source current at valid input voltage levels. Since this "bus hold" circuitry is active and not a "resistive" type element, the associated power supply current is negligible and power dissipation is significantly reduced when compared to the use of passive pull-up resistors.

INTERRUPT OPERATIONS

Interrupt operations fall into two classes: software or hardware initiated. The software initiated interrupts and software aspects of hardware interrupts are specified in the instruction set description in the iAPX 88 book or the iAPX 86,88 User's Manual. Hardware interrupts can be classified as nonmaskable or maskable.

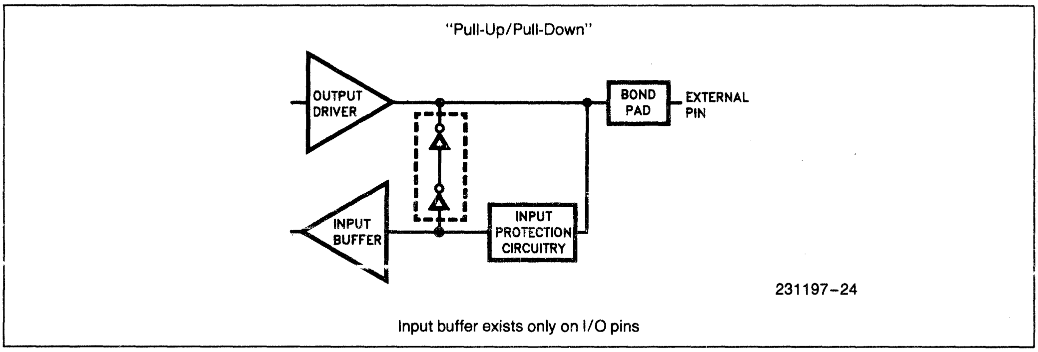


Figure 9a. Bus hold circuitry pin 2–16, 35–39.

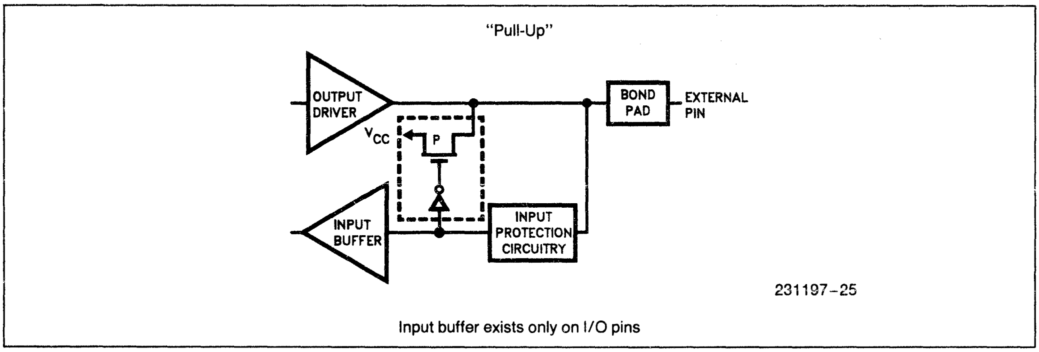


Figure 9b. Bus hold circuitry pin 26–32, 34.

Interrupts result in a transfer of control to a new program location. A 256 element table containing address pointers to the interrupt service program locations resides in absolute locations 0 through 3FFH (See Figure 4), which are reserved for this purpose. Each element in the table is 4 bytes in size and corresponds to an interrupt "type." An interrupting device supplies an 8-bit type number, during the interrupt acknowledge sequence, which is used to vector through the appropriate element to the new interrupt service program location.

NON-MASKABLE INTERRUPT (NMI)

The processor provides a single non-maskable interrupt (NMI) pin which has higher priority than the maskable interrupt request (INTR) pin. A typical use would be to activate a power failure routine. The NMI is edge-triggered on a LOW to HIGH transition. The activation of this pin causes a type 2 interrupt.

NMI is required to have a duration in the HIGH state of greater than two clock cycles, but is not required to be synchronized to the clock. Any higher going transition of NMI is latched on-chip and will be serviced at the end of the current instruction or between whole moves (2 bytes in the case of word moves) of a block type instruction. Worst case response to NMI would be for multiply, divide, and variable shift instructions. There is no specification on the occurrence of the low-going edge; it may occur before, during, or after the servicing of NMI. Another high-going edge triggers another response if it occurs after the start of the NMI procedure. The signal must be free of logical spikes in general and be free of bounces on the low-going edge to avoid triggering extraneous responses.

MASKABLE INTERRUPT (INTR)

The 80C88 provides a single interrupt request input (INTR) which can be masked internally by software with the resetting of the interrupt enable (IF) flag bit. The interrupt request signal is level triggered. It is internally synchronized during each clock cycle on the high-going edge of CLK. To be responded to, INTR must be present (HIGH) during the clock period preceding the end of the current instruction or the end of a whole move for a block type instruction. During interrupt response sequence, further interrupts are disabled. The enable bit is reset as part of the response to any interrupt (INTR, NMI, software interrupt, or single step), although the FLAGS register which is automatically pushed onto the stack reflects the state of the processor prior to the interrupt. Until the old FLAGS register is restored, the enable bit will be zero unless specifically set by an instruction.

During the response sequence (See Figure 10), the processor executes two successive (back to back) interrupt acknowledge cycles. The 80C88 emits the LOCK signal (maximum mode only) from T2 of the first bus cycle until T2 of the second. A local bus "hold" request will not be honored until the end of the second bus cycle. In the second bus cycle, a

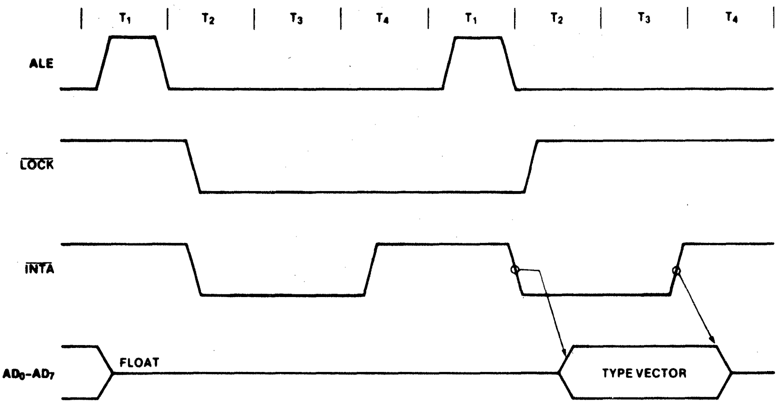


Figure 10. Interrupt Acknowledge Sequence

byte is fetched from the external interrupt system (e.g., 82C59A PIC) which identifies the source (type) of the interrupt. This byte is multiplied by four and used as a pointer into the interrupt vector lookup table. An INTR signal left HIGH will be continually responded to within the limitations of the enable bit and sample period. The interrupt return instruction includes a flags pop which returns the status of the original interrupt enable bit when it restores the flags.

HALT

When a software HALT instruction is executed, the processor indicates that it is entering the HALT state in one of two ways, depending upon which mode is strapped. In minimum mode, the processor issues ALE, delayed by one clock cycle, to allow the system to latch the halt status. Halt status is available on IO/$\overline{M}$, DT/$\overline{R}$, and $\overline{SSO}$. In maximum mode, the processor issues appropriate HALT status on $\overline{S2}$, $\overline{S1}$, and $\overline{S0}$, and the 82C88 bus controller issues one ALE. The 80C88 will not leave the HALT state when a local bus hold is entered while in HALT. In this case, the processor reissues the HALT indicator at the end of the local bus hold. An interrupt request or RESET will force the 80C88 out of the HALT state.

READ/MODIFY/WRITE (SEMAPHORE) OPERATIONS VIA LOCK

The LOCK status information is provided by the processor when consecutive bus cycles are required during the execution of an instruction. This allows the processor to perform read/modify/write operations on memory (via the "exchange register with memory" instruction), without another system bus master receiving intervening memory cycles. This is useful in multiprocessor system configurations to accomplish "test and set lock" operations. The $\overline{LOCK}$ signal is activated (LOW) in the clock cycle following decoding of the LOCK prefix instruction. It is deactivated at the end of the last bus cycle of the instruction following the LOCK prefix. While $\overline{LOCK}$ is active, a request on a $\overline{RQ}/\overline{GT}$ pin will be recorded, and then honored at the end of the LOCK.

EXTERNAL SYNCHRONIZATION VIA $\overline{TEST}$

As an alternative to interrupts, the 80C88 provides a single software-testable input pin ($\overline{TEST}$). This input is utilized by executing a WAIT instruction. The single WAIT instruction is repeatedly executed until the $\overline{TEST}$ input goes active (LOW). The execution of WAIT does not consume bus cycles once the queue is full.

If a local bus request occurs during WAIT execution, the 80C88 3-states all output drivers. If interrupts are enabled, the 80C88 will recognize interrupts and process them. The WAIT instruction is then refetched, and reexecuted.

BASIC SYSTEM TIMING

In minimum mode, the MN/$\overline{MX}$ pin is strapped to V_{CC} and the processor emits bus control signals compatible with the 8085 bus structure. In maximum mode, the MN/$\overline{MX}$ pin is strapped to GND and the processor emits coded status information which the 82C88 bus controller uses to generate MULTIBUS compatible bus control signals.

System Timing — Minimum System

(See Figure 8.)

The read cycle begins in T1 with the assertion of the address latch enable (ALE) signal. The trailing (low going) edge of this signal is used to latch the address information, which is valid on the address/data bus (AD0–AD7) at this time, into a latch. Address lines A8 through A15 do not need to be latched because they remain valid throughout the bus cycle. From T1 to T4 the IO/$\overline{M}$ signal indicates a memory or I/O operation. At T2 the address is removed from the address/data bus and the bus goes to a high impedance state. The read control signal is also asserted at T2. The read ($\overline{RD}$) signal causes the addressed device to enable its data bus drivers to the local bus. Some time later, valid data will be available on the bus and the addressed device will drive the READY line HIGH. When the processor returns the read signal to a HIGH level, the addressed device will again 3-state its bus drivers. If a transceiver is required to buffer the 80C88 local bus, signals DT/$\overline{R}$ and $\overline{DEN}$ are provided by the 80C88.

A write cycle also begins with the assertion of ALE and the emission of the address. The IO/$\overline{M}$ signal is again asserted to indicate a memory or I/O write operation. In T2, immediately following the address emission, the processor emits the data to be written into the addressed location. This data remains valid until at least the middle of T4. During T2, T3, and T_W, the processor asserts the write control signal. The write ($\overline{WR}$) signal becomes active at the beginning of T2, as opposed to the read, which is delayed somewhat into T2 to provide time for the bus to float.

The basic difference between the interrupt acknowledge cycle and a read cycle is that the interrupt acknowledge ($\overline{INTA}$) signal is asserted in place of the read ($\overline{RD}$) signal and the address bus is floated. (See Figure 10.) In the second of two successive $\overline{INTA}$ cycles, a byte of information is read from the data bus, as supplied by the interrupt system logic (i.e. 82C59A priority interrupt controller). This byte identifies the source (type) of the interrupt. It is multiplied by four and used as a pointer into the interrupt vector lookup table, as described earlier.

BUS TIMING — MEDIUM COMPLEXITY SYSTEMS

(See Figure 11.)

For medium complexity systems, the MN/$\overline{MX}$ pin is connected to GND and the 82C88 bus controller is added to the system, as well as a latch for latching the system address, and a transceiver to allow for bus loading greater than the 80C88 is capable of handling. Signals ALE, $\overline{DEN}$, and DT/$\overline{R}$ are generated by the 82C88 instead of the processor in this configuration, although their timing remains relatively the same. The 80C88 status outputs ($\overline{S2}$, $\overline{S1}$, and $\overline{S0}$) provide type of cycle information and become 82C88 inputs. This bus cycle information specifies read (code, data, or I/O), write (data or I/O), interrupt acknowledge, or software halt. The 82C88 thus issues control signals specifying memory read or write, I/O read or write, or interrupt acknowledge. The 82C88 provides two types of write strobes, normal and advanced, to be applied as required. The normal write strobes have data valid at the leading edge of write. The advanced write strobes have the same timing as read strobes, and hence, data is not valid at the leading edge of write. The transceiver receives the usual T and $\overline{OE}$ inputs from the 82C88's DT/$\overline{R}$ and $\overline{DEN}$ outputs.

The pointer into the interrupt vector table, which is passed during the second $\overline{INTA}$ cycle, can derive from an 82C59A located on either the local bus or the system bus. If the master 82C59A priority interrupt controller is positioned on the local bus, a TTL gate is required to disable the transceiver when reading from the master 82C59A during the interrupt acknowledge sequence and software "poll".

THE 80C88 COMPARED TO THE 80C86

The 80C88 CPU is an 8-bit processor designed around the 80C86 internal structure. Most internal functions of the 80C88 are identical to the equivalent 80C86 functions. The 80C88 handles the external bus the same way the 80C86 does with the distinction of handling only 8 bits at a time. Sixteen-bit operands are fetched or written in two consecutive bus cycles. Both processors will appear identical to the software engineer, with the exception of execution time. The internal register structure is identical and all instructions have the same end result. The differences between the 80C88 and 80C86 are outlined below. The engineer who is unfamiliar with the 80C86 is referred to the iAPX 86, 88 User's Manual, Chapters 2 and 4, for function description and instruction set information. Internally, there are three differences between the 80C88 and the 80C86. All changes are related to the 8-bit bus interface.

- The queue length is 4 bytes in the 80C88, whereas the 80C86 queue contains 6 bytes, or three words. The queue was shortened to prevent overuse of the bus by the BIU when prefetching instructions. This was required because of the additional time necessary to fetch instructions 8 bits at a time.
- To further optimize the queue, the prefetching algorithm was changed. The 80C88 BIU will fetch a new instruction to load into the queue each time there is a 1 byte hole (space available) in the queue. The 80C86 waits until a 2-byte space is available.
- The internal execution time of the instruction set is affected by the 8-bit interface. All 16-bit fetches and writes from/to memory take an additional four clock cycles. The CPU is also limited by the speed of instruction fetches. This latter problem only occurs when a series of simple operations occur. When the more sophisticated instructions of the 80C88 are being used, the queue has time to fill and the execution proceeds as fast as the execution unit will allow.

The 80C88 and 80C86 are completely software compatible by virture of their identical execution units. Software that is system dependent may not be completely transferable, but software that is not system dependent will operate equally as well on an 80C88 or an 80C86.

The hardware interface of the 80C88 contains the major differences between the two CPUs. The pin assignments are nearly identical, however with the following functional changes:

- A8–A15 — These pins are only address outputs on the 80C88. These address lines are latched internally and remain valid throughout a bus cycle in a manner similar to the 8085 upper address lines.

- $\overline{BHE}$ has no meaning on the 80C88 and has been eliminated.
- $\overline{SSO}$ provides the $\overline{SO}$ status information in the minimum mode. This output occurs on pin 34 in minimum mode only. DT/$\overline{R}$, IO/$\overline{M}$, and $\overline{SSO}$ provide the complete bus status in minimum mode.
- IO/$\overline{M}$ has been inverted to be compatible with the MCS-85 bus structure.
- ALE is delayed by one clock cycle in the minimum mode when entering HALT, to allow the status to be latched with ALE.

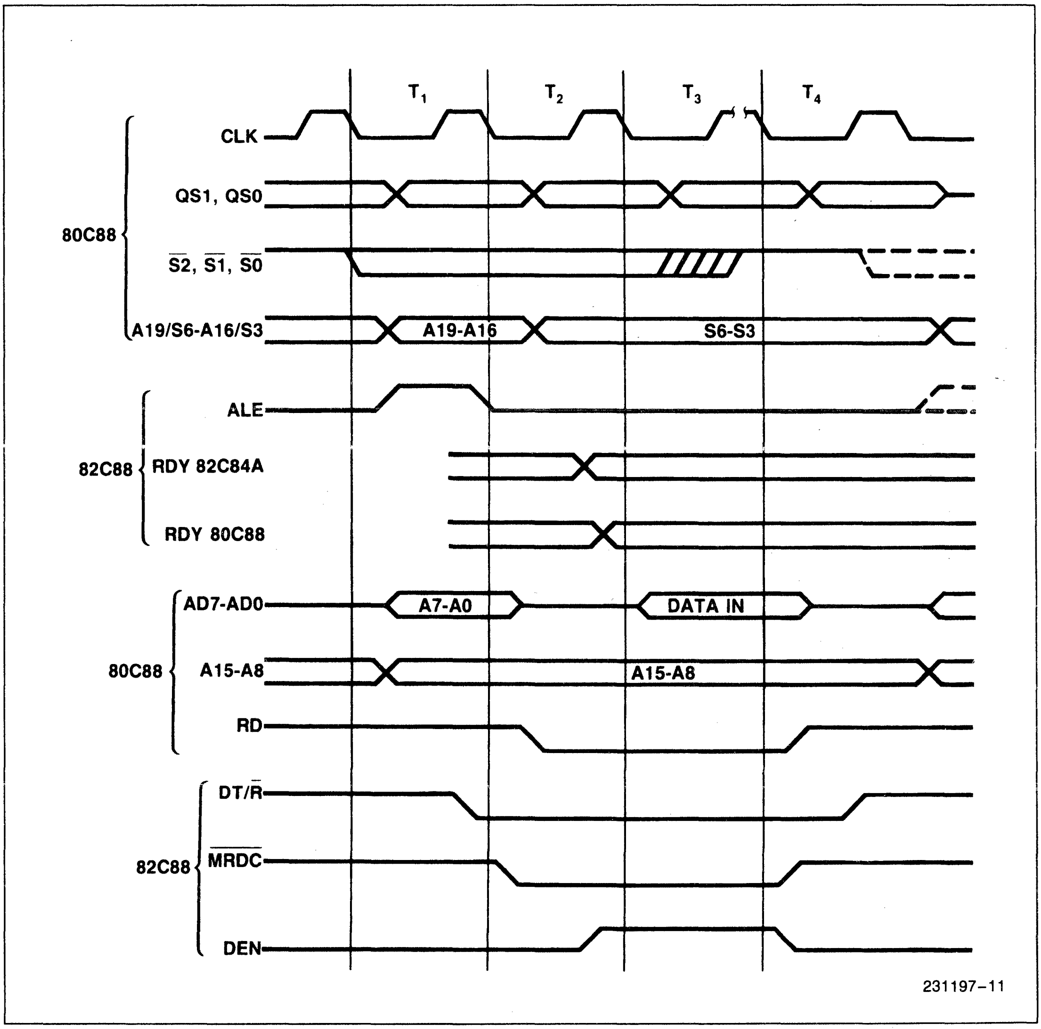


Figure 11. Medium Complexity System Timing

ABSOLUTE MAXIMUM RATINGS*

Supply Voltage
(With respect to ground) .... −0.5 to 8.0V
Input Voltage Applied
(w.r.t. ground) .... −2.0 to V_{CC} + 0.5V
Output Voltage Applied
(w.r.t. ground) .... −0.5 to V_{CC} + 0.5V
Power Dissipation 1.0W
Storage Temperature .... −65°C to +150°C
Ambient Temperature Under Bias .... 0°C to +70°C

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS (80C88: T_A = 0°C to 70°C, V_{CC} = 5V ±10%) (80C88-2: T_A = 0°C to 70°C, V_{CC} = 5V ±5%)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage (All input except $\overline{RQ}/\overline{GT0}$, $\overline{RQ}/\overline{GT1}$ MN/MX)	2.0	V_{CC}+0.5	V	(Note 6)
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 2.5 mA
V_{OH}	Output High Voltage	3.0 V_{CC}−0.4		V	I_{OH} = −2.5 mA I_{OH} = −100 μA
I_{CC}	Power Supply Current		10 mA/MHz		T_A = 25°C, V_{CC} = 5.5V V_{IL} = GND, V_{IH} = V_{CC}
I_{CCS}	Standby Supply Current		750	μA	V_{IN}(max) = V_{CC} or GND V_{CC} = 5.5V Ready = High Outputs Unloaded CLK = GND or V_{CC} (Note 7)
I_{CCS}	Standby Supply Current		2.5	mA	V_{CC} = 5.5V Ready = Low V_{IN} (max) = V_{CC} or GND Outputs Unloaded CLK = GND or V_{CC} (Note 7)
I_{LI}	Input Leakage Current		±1.0	μA	0V≤V_{IN}≤V_{CC}
I_{BHL}	Input Leakage Current (Bus Hold Low)	50	300	μA	V_{IN} = 0.8V (Note 1)
I_{BHH}	Input Leakage Current (Bus Hold High)	−50	−300	μA	V_{IN} = 3.0V (Note 2)
I_{BHLO}	Bus Hold Low Overdrive		350	μA	(Note 4)
I_{BHHO}	Bus Hold High Overdrive		−350	μA	(Note 5)
I_{LO}	Output Leakage Current		±10	μA	0≤V_{OUT}≤V_{CC}
V_{CL}	Clock Input Low Voltage	−0.5	+0.8	V	
V_{CH}	Clock Input High Voltage	V_{CC}−0.8	V_{CC}+0.5	V	
C_{IN}	Capacitance of Input Buffer (All input except AD_0–AD_7, RQ/GT)		5	pF	(Note 3)
C_{IO}	Capacitance of I/O Buffer (AD_0–AD_7, RQ/GT)		20	pF	(Note 3)
C_{OUT}	Output Capacitance		15	pF	(Note 3)

NOTES:
1. Test condition is to lower V_{IN} to GND and then raise V_{IN} to 0.8V on pins 2-16, and 35-39.
2. Test condition is to raise V_{IN} to V_{CC} and then lower V_{IN} to 3.0V on pins 2-16, 26-32, and 34-39.
3. Characterization conditions are a) Frequency = 1 MHz, b) Unmeasured pins at GND c) V_{IN} at +5.0V or GND.
4. An external driver must source at least I_{BHLO} to switch this node from LOW to HIGH.
5. An external driver must sink at least I_{BHHO} to switch this node from HIGH to LOW.
6. V_{IH} for MN/MX is 2.5V.
7. This spec may improve to 500 μA during 1986.

A.C. CHARACTERISTICS (80C88: T_A = 0°C to 70°C, V_{CC} = 5V ±10%) (80C88-2: T_A = 0°C to 70°C, V_{CC} = 5V ±5%)

MINIMUM COMPLEXITY SYSTEM TIMING REQUIREMENTS

Symbol	Parameter	80C88 Min	80C88 Max	80C88-2 Min	80C88-2 Max	Units	Test Conditions
TCLCL	CLK Cycle Period	200	D.C.	125	D.C.	ns	
TCLCH	CLK Low Time	118		68		ns	
TCHCL	CLK High Time	69		44		ns	
TCH1CH2	CLK Rise Time		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		20		ns	CL = 20−100 pF
TCLDX	Data in Hold Time	10		10		ns	
TR1VCL	RDY Setup Time into 82C84A (Notes 1, 2)	35		35		ns	
TCLR1X	RDY Hold Time into 82C84A (Notes 1, 2)	0		0		ns	
TRYHCH	READY Setup Time into 80C88	118		68		ns	
TCHRYX	READY Hold Time into 80C88	30		20		ns	
TRYLCL	READY Inactive to CLK (Note 3)	−8		−8		ns	
THVCH	HOLD Setup Time	35		20		ns	
TINVCH	INTR, NMI, $\overline{TEST}$ Setup Time (Note 2)	30		15		ns	
TILIH	Input Rise Time (Except CLK) (Note 4)		15		15	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK) (Note 4)		15		15	ns	From 2.0V to 0.8V

NOTES:
1. Signal at 82C84A or 82C88 shown for reference only.
2. Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
3. Applies only to T2 state (8 ns into T3 state).
4. Characterization only.

A.C. CHARACTERISTICS (Continued)

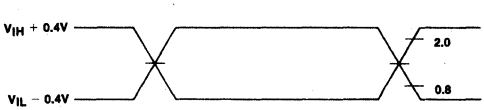
TIMING RESPONSES

		80C88		80C88-2			
Symbol	**Parameter**	**Min**	**Max**	**Min**	**Max**	**Units**	**Test Conditions**
TCLAV	Address Valid Delay	10	110	10	60	ns	C_L = 20–100 pF for all 80C88 Outputs in addition to internal loads
TCLAX	Address Hold Time	10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	TCLAX	50	ns	
TLHLL	ALE Width	TCLCH−20		TCLCH−10		ns	
TCLLH	ALE Active Delay		80		50	ns	
TCHLL	ALE Inactive Delay		85		55	ns	
TLLAX	Address Hold Time to ALE Inactive	TCHCL−25		TCHCL−25		ns	
TCLDV	Data Valid Delay	10	110	10	60	ns	
TCHDX	Data Hold Time	10		10		ns	
TWHDX	Data Hold Time After $\overline{WR}$	TCLCH−30		TCLCH−30		ns	
TCVCTV	Control Active Delay 1	10	110	10	70	ns	
TCHCTV	Control Active Delay 2	10	110	10	60	ns	
TCVCTX	Control Inactive Delay	10	110	10	70	ns	
TAZRL	Address Float to READ Active	0		0		ns	
TCLRL	$\overline{RD}$ Active Delay	10	165	10	100	ns	
TCLRH	$\overline{RD}$ Inactive Delay	10	150	10	80	ns	
TRHAV	$\overline{RD}$ Inactive to Next Address Active	TCLCL−45		TCLCL−40		ns	
TCLHAV	HLDA Valid Delay	10	160	10	100	ns	
TRLRH	$\overline{RD}$ Width	2TCLCL−75		2TCLCL−50		ns	
TWLWH	$\overline{WR}$ Width	2TCLCL−60		2TCLCL−40		ns	
TAVAL	Address Valid to ALE Low	TCLCH−60		TCLCH−40		ns	
TOLOH	Output Rise Time (Note 1)		15		15	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time (Note 1)		15		15	ns	From 2.0V to 0.8V

NOTE:
1. Characterization only.

A.C. TESTING INPUT, OUTPUT WAVEFORM

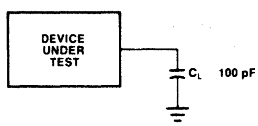
Input/Output



231197–12

A.C. Testing inputs are driven at V_{IH} + 0.4V for a logic "1" and V_{IL} − 0.4V for a logic "0". The clock is driven at V_{CH} + 0.4V and V_{CL} − 0.4V. Timing measurements are made at 2.0V and 0.8V.

A.C. TESTING LOAD CIRCUIT

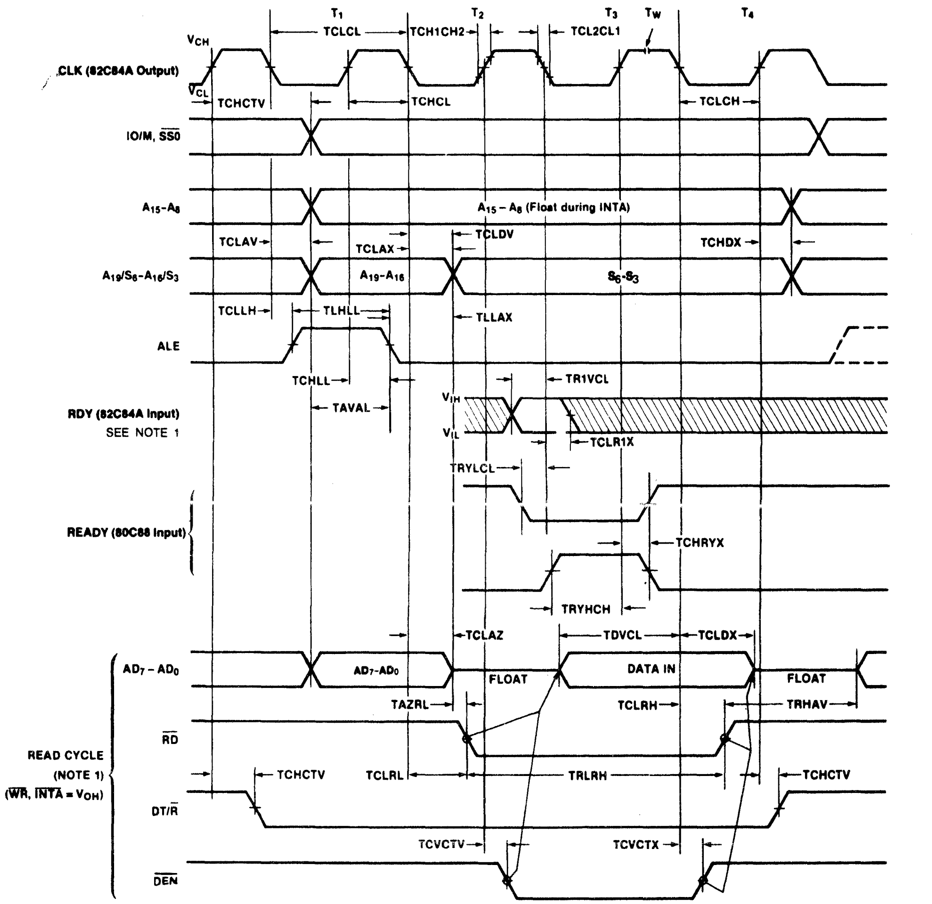


231197–13

C_L Includes Jig Capacitance

WAVEFORMS

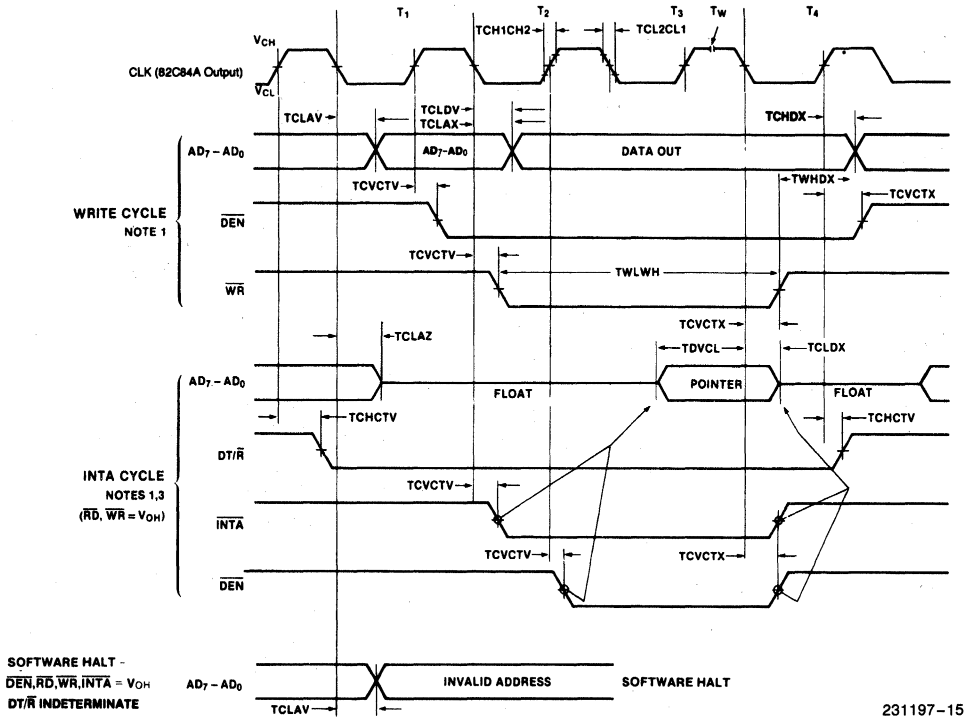
BUS TIMING — MINIMUM MODE SYSTEM



231197-14

WAVEFORMS (Continued)

BUS TIMING — MINIMUM MODE SYSTEM (Continued)



NOTES:

1. All output timing measurements are made at 0.8V and 2.0V unless otherwise noted.
2. RDY is sampled near the end of T_2, T_3, T_W to determine if T_W machines states are to be inserted.
3. Two INTA Cycles run back-to-back. The 80C88 local ADDR/Data bus is floating during both INTA Cycles. Control signals are shown for the second INTA cycle.
4. Signals at 82C84A are shown for reference only.

A.C. CHARACTERISTICS

MAX MODE SYSTEM (USING 82C88 BUS CONTROLLER)
TIMING REQUIREMENTS

		80C88		80C88-2			
Symbol	**Parameter**	**Min**	**Max**	**Min**	**Max**	**Units**	**Test Conditions**
TCLCL	CLK Cycle Period	200	D.C.	125	D.C.	ns	
TCLCH	CLK Low Time	118		68		ns	
TCHCL	CLK High Time	69		44		ns	
TCH1CH2	CLK Rise Time		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10	ns	From 3.5V to 1.0V
TDVCL	Data In Setup Time	30		20		ns	
TCLDX	Data In Hold Time	10		10		ns	
TR1VCL	RDY Setup Time into 82C84 (See Notes 1, 2)	35		35		ns	
TCLR1X	RDY Hold Time into 82C84 (See Notes 1, 2)	0		0		ns	
TRYHCH	READY Setup Time into 80C88	118		68		ns	CL = 20 −100 pF
TCHRYX	READY Hold Time into 80C88	30		20		ns	
TRYLCL	READY Inactive to CLK (See Note 4)	−8		−8		ns	
TINVCH	Setup Time for Recognition (INTR, NMI, TEST) (See Note 2)	30		15		ns	
TGVCH	RQ/GT Setup Time	30		15		ns	
TCHGX	RQ Hold Time into 80C88	40		30		ns	
TILIH	Input Rise Time (Except CLK) (Note 5)		15		15	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK) (Note 5)		15		15	ns	From 2.0V to 0.8V

NOTES:
1. Signal at 82C84A or 82C88 shown for reference only.
2. Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
3. Applies only to T3 and wait states (8 ns into T3 state).
4. Applies only to T2 state (8 ns into T3 state).
5. Characterization only.

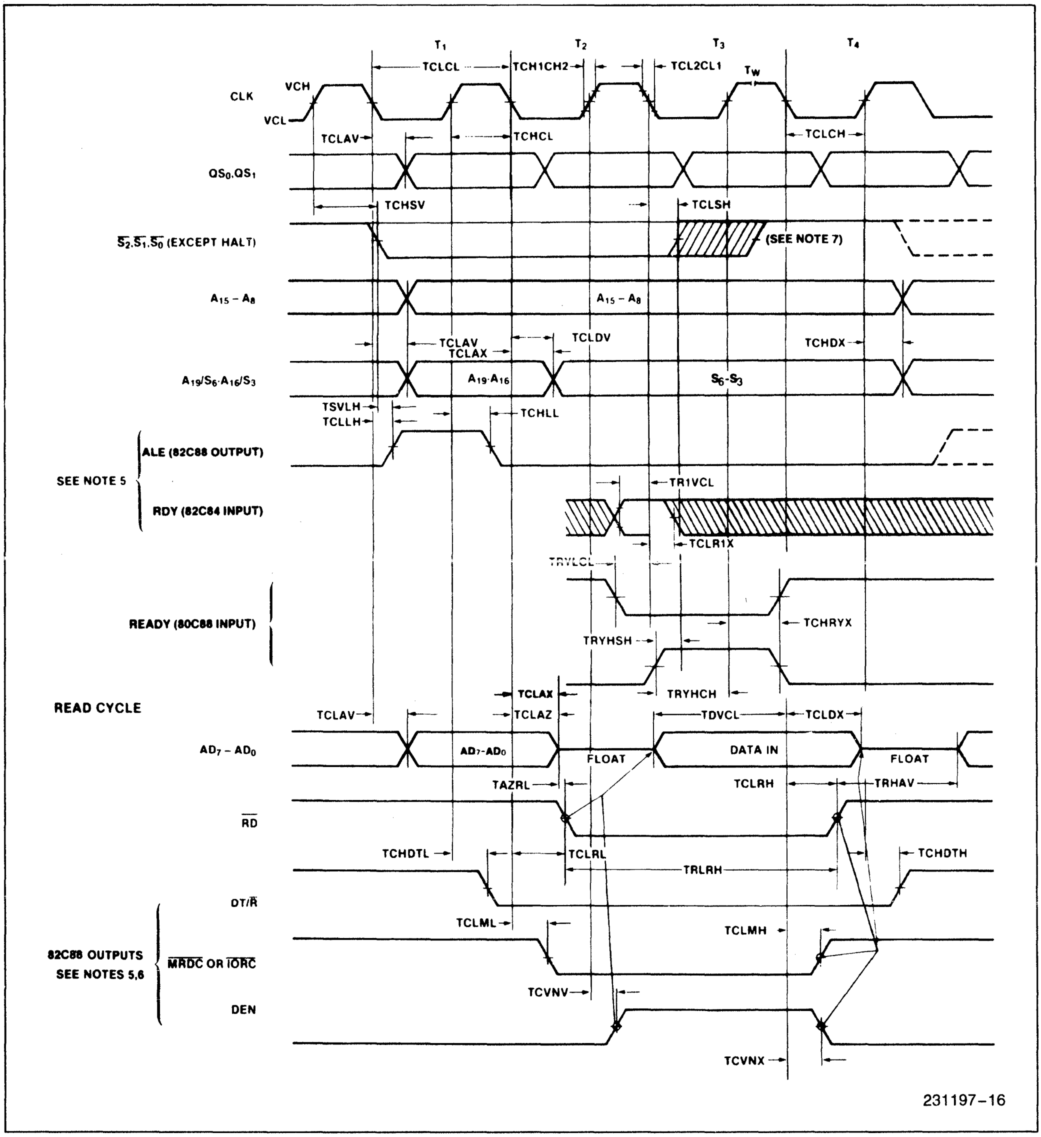
A.C. CHARACTERISTICS

TIMING RESPONSES

Symbol	Parameter	80C88 Min	80C88 Max	80C88-2 Min	80C88-2 Max	Units	Test Conditions
TCLML	Command Active Delay (Note 1)	5	35	5	35	ns	
TCLMH	Command Inactive Delay (Note 1)	5	35	5	35	ns	
TRYHSH	READY Active to Status Passive (Note 3)		110		65	ns	
TCHSV	Status Active Delay	10	110	10	60	ns	
TCLSH	Status Inactive Delay	10	130	10	70	ns	
TCLAV	Address Valid Delay	10	110	10	60	ns	
TCLAX	Address Hold Time	10		10		ns	C_L = 20–100 pF for all 80C88 Outputs in addition to internal loads
TCLAZ	Address Float Delay	TCLAX	80	TCLAX	50	ns	
TSVLH	Status Valid to ALE High (Note 1)		20		20	ns	
TSVMCH	Status Valid to MCE High (Note 1)		30		30	ns	
TCLLH	CLK Low to ALE Valid (Note 1)		20		20	ns	
TCLMCH	CLK Low to MCE High (Note 1)		25		25	ns	
TCHLL	ALE Inactive Delay (Note 1)	4	25	4	25	ns	
TCLDV	Data Valid Delay	10	110	10	60	ns	
TCHDX	Data Hold Time	10		10		ns	
TCVNV	Control Active Delay (Note 1)	5	45	5	45	ns	
TCVNX	Control Inactive Delay (Note 1)	10	45	10	45	ns	
TAZRL	Address Float to Read Active	0		0		ns	
TCLRL	RD Active Delay	10	165	10	100	ns	
TCLRH	RD Inactive Delay	10	150	10	80	ns	
TRHAV	RD Inactive to Next Address Active	TCLCL-45		TCLCL-40		ns	
TCHDTL	Direction Control Active Delay (Note 1)		50		50	ns	
TCHDTH	Direction Control Inactive Delay (Note 1)		30		30	ns	
TCLGL	GT Active Delay	0	85	0	50	ns	
TCLGH	GT Inactive Delay	0	85	0	50	ns	
TRLRH	RD Width	2TCLCL-75		2TCLCL-50		ns	
TOLOH	Output Rise Time		15		15	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		15		15	ns	From 2.0V to 0.8V

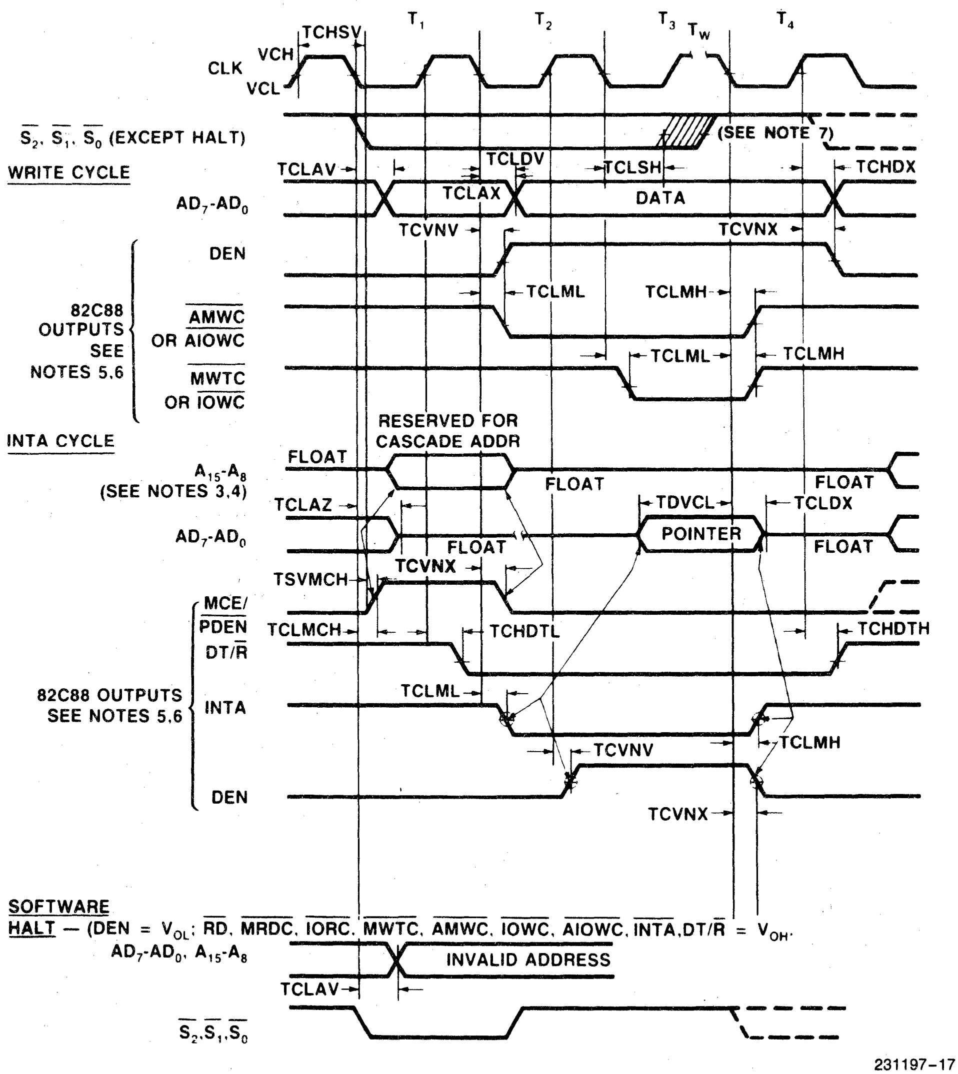
WAVEFORMS

BUS TIMING—MAXIMUM MODE



WAVEFORMS (Continued)

BUS TIMING — MAXIMUM MODE SYSTEM (USING 82C88)

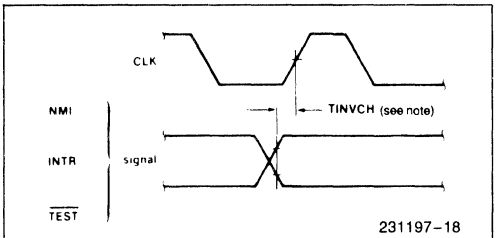


NOTES:

1. All output timing measurements are made at 0.8V and 2.0V unless otherwise noted.
2. RDY is sampled near the end of T_2, T_3, T_W to determine if T_W machines states are to be inserted.
3. Cascade address is valid between first and second INTA cycles.
4. Two INTA cycles run back-to-back. The 80C88 local ADDR/Data bus is floating during both INTA cycles. Control for pointer address is shown for second INTA cycle.
5. Signals at 82C84A or 82C88 are shown for reference only.
6. The issuance of the 82C88 command and control signals (MRDC, MWTC, AMWC, IORC, IOWC, AIOWC, INTA and DEN) lags the active high 82C88 CEN.
7. Status inactive in state just prior to T_4.

WAVEFORMS (Continued)

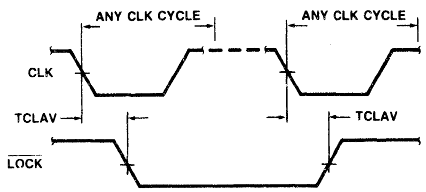
ASYNCHRONOUS SIGNAL RECOGNITION



231197-18

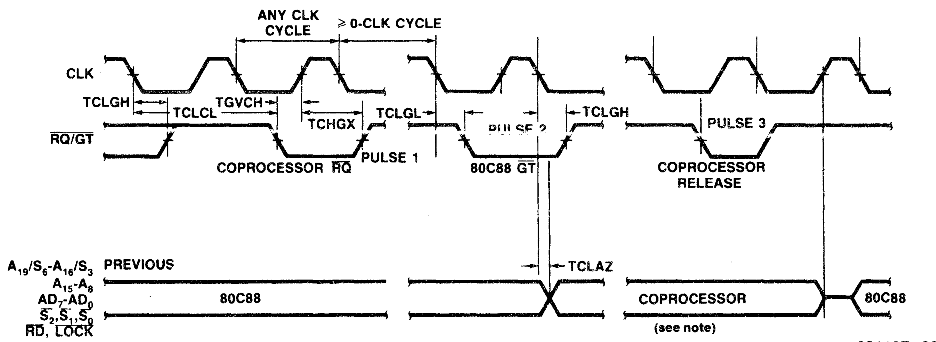
NOTE: Setup requirements for asynchronous signals only to guarantee recognition at next CLK.

BUS LOCK SIGNAL TIMING (MAXIMUM MODE ONLY)



231197-19

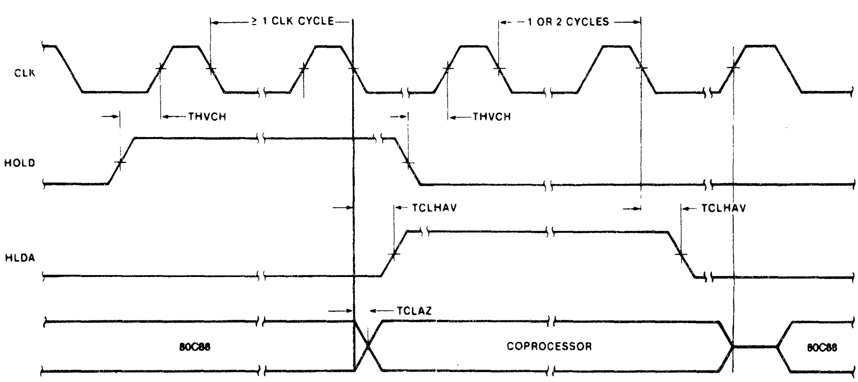
REQUEST/GRANT SEQUENCE TIMING (MAXIMUM MODE ONLY)



231197-20

NOTE: The coprocessor may not drive the busses outside the region shown without risking contention.

HOLD/HOLD ACKNOWLEDGE TIMING (MINIMUM MODE ONLY)



231197-21

80C86/80C88

INSTRUCTION SET SUMMARY

DATA TRANSFER

	76543210	76543210	76543210	76543210
MOV = Move:				
Register/memory to/from register	1 0 0 0 1 0 d w	mod reg r/m		
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w 1
Immediate to register	1 0 1 1 w reg	data	data if w 1	
Memory to accumulator	1 0 1 0 0 0 0 w	add-low	addr-high	
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high	
Register/memory to segment register**	1 0 0 0 1 1 1 0	mod 0 reg r/m		
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 reg r/m		
PUSH = Push:				
Register/memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m		
Register	0 1 0 1 0 reg			
Segment register	0 0 0 reg 1 1 0			
POP = Pop:				
Register/memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m		
Register	0 1 0 1 1 reg			
Segment register	0 0 0 reg 1 1 1			
XCHG = Exchange:				
Register/memory with register	1 0 0 0 0 1 1 w	mod reg r/m		
Register with accumulator	1 0 0 1 0 reg			
IN = Input from:				
Fixed port	1 1 1 0 0 1 0 w	port		
Variable port	1 1 1 0 1 1 0 w			
OUT = Output to:				
Fixed port	1 1 1 0 0 1 1 w	port		
Variable port	1 1 1 0 1 1 1 w			
XLAT = Translate byte to AL	1 1 0 1 0 1 1 1			
LEA = Load EA to register	1 0 0 0 1 1 0 1	mod reg r/m		
LDS = Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m		
LES = Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m		
LAHF = Load AH with flags	1 0 0 1 1 1 1 1			
SAHF = Store AH into flags	1 0 0 1 1 1 1 0			
PUSHF = Push flags	1 0 0 1 1 1 0 0			
POPF = Pop flags	1 0 0 1 1 1 0 1			

ARITHMETIC

	76543210	76543210	76543210	76543210
ADD = Add:				
Reg./memory with register to either	0 0 0 0 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s:w = 01
Immediate to accumulator	0 0 0 0 0 1 0 w	data	data if w 1	
ADC = Add with carry:				
Reg/memory with register to either	0 0 0 1 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s:w = 01
Immediate to accumulator	0 0 0 1 0 1 0 w	data	data if w = 1	
INC = Increment:				
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m		
Register	0 1 0 0 0 reg			
AAA = ASCII adjust for add	0 0 1 1 0 1 1 1			
DAA = Decimal adjust for add	0 0 1 0 0 1 1 1			
SUB = Subtract:				
Reg./memory and register to either	0 0 1 0 1 0 d w	mod reg r/m		
Immediate from register/memory	1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s:w = 01
Immediate from accumulator	0 0 1 0 1 1 0 w	data	data if w = 1	
SBB = Subtract with borrow				
Reg./memory and register to either	0 0 0 1 1 0 d w	mod reg r/m		
Immediate from register/memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s:w = 01
Immediate from accumulator	0 0 0 1 1 1 0 w	data	data if w = 1	

	76543210	76543210	76543210	76543210
DEC Decrement:				
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m		
Register	0 1 0 0 1 reg			
NEG Change sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m		
CMP Compare:				
Register/memory and register	0 0 1 1 1 0 d w	mod reg r/m		
Immediate with register/memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s:w = 01
Immediate with accumulator	0 0 1 1 1 1 0 w	data	data if w = 1	
AAS ASCII adjust for subtract	0 0 1 1 1 1 1 1			
DAS Decimal adjust for subtract	0 0 1 0 1 1 1 1			
MUL Multiply (unsigned)	1 1 1 1 0 1 1 w	mod 1 0 0 r/m		
IMUL Integer multiply (signed)	1 1 1 1 0 1 1 w	mod 1 0 1 r/m		
AAM ASCII adjust for multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0		
DIV Divide (unsigned)	1 1 1 1 0 1 1 w	mod 1 1 0 r/m		
IDIV Integer divide (signed)	1 1 1 1 0 1 1 w	mod 1 1 1 r/m		
AAD ASCII adjust for divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0		
CBW Convert byte to word	1 0 0 1 1 0 0 0			
CWD Convert word to double word	1 0 0 1 1 0 0 1			

LOGIC

	76543210	76543210	76543210	76543210
NOT Invert	1 1 1 1 0 1 1 w	mod 0 1 0 r/m		
SHL/SAL Shift logical/arithmetic left	1 1 0 1 0 0 v w	mod 1 0 0 r/m		
SHR Shift logical right	1 1 0 1 0 0 v w	mod 1 0 1 r/m		
SAR Shift arithmetic right	1 1 0 1 0 0 v w	mod 1 1 1 r/m		
ROL Rotate left	1 1 0 1 0 0 v w	mod 0 0 0 r/m		
ROR Rotate right	1 1 0 1 0 0 v w	mod 0 0 1 r/m		
RCL Rotate through carry flag left	1 1 0 1 0 0 v w	mod 0 1 0 r/m		
RCR Rotate through carry right	1 1 0 1 0 0 v w	mod 0 1 1 r/m		
AND And:				
Reg./memory and register to either	0 0 1 0 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w = 1
Immediate to accumulator	0 0 1 0 0 1 0 w	data	data if w = 1	
TEST And function to flags, no result:				
Register/memory and register	1 0 0 0 0 1 0 w	mod reg r/m		
Immediate data and register/memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1
Immediate data and accumulator	1 0 1 0 1 0 0 w	data	data if w = 1	
OR Or:				
Reg./memory and register to either	0 0 0 0 1 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w = 1
Immediate to accumulator	0 0 0 0 1 1 0 w	data	data if w = 1	
XOR = Exclusive or:				
Reg./memory and register to either	0 0 1 1 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w = 1
Immediate to accumulator	0 0 1 1 0 1 0 w	data	data if w = 1	

STRING MANIPULATION

	76543210
REP = Repeat	1 1 1 1 0 0 1 z
MOVS = Move byte/word	1 0 1 0 0 1 0 w
CMPS = Compare byte/word	1 0 1 0 0 1 1 w
SCAS = Scan byte/word	1 0 1 0 1 1 1 w
LODS = Load byte/wd to AL/AX	1 0 1 0 1 1 0 w
STOS = Stor byte/wd from AL/A	1 0 1 0 1 0 1 w

231197–22

INSTRUCTION SET SUMMARY (Continued)

CONTROL TRANSFER

	76543210	76543210	76543210
CALL = Call:			
Direct within segment	11101000	disp-low	disp-high
Indirect within segment	11111111	mod 0 1 0 r/m	
Direct intersegment	10011010	offset-low	offset-high
		seg-low	seg-high
Indirect intersegment	11111111	mod 0 1 1 r/m	
JMP = Unconditional Jump:			
Direct within segment	11101001	disp-low	disp-high
Direct within segment-short	11101011	disp	
Indirect within segment	11111111	mod 1 0 0 r/m	
Direct intersegment	11101010	offset-low	offset-high
		seg-low	seg-high
Indirect intersegment	11111111	mod 1 0 1 r/m	
RET = Return from CALL:			
Within segment	11000011		
Within seg. adding immed to SP	11000010	data-low	data-high
Intersegment	11001011		
Intersegment adding immediate to SP	11001010	data-low	data-high
JE/JZ = Jump on equal/zero	01110100	disp	
JL/JNGE = Jump on less/not greater or equal	01111100	disp	
JLE/JNG = Jump on less or equal/not greater	01111110	disp	
JB/JNAE = Jump on below/not above or equal	01110010	disp	
JBE/JNA = Jump on below or equal/ not above	01110110	disp	
JP/JPE = Jump on parity/parity even	01111010	disp	
JO = Jump on overflow	01110000	disp	
JS = Jump on sign	01111000	disp	
JNE/JNZ = Jump on not equal/not zero	01110101	disp	
JNL/JGE = Jump on not less/greater or equal	01111101	disp	
JNLE/JG = Jump on not less or equal/ greater	01111111	disp	

	76543210	76543210
JNB/JAE Jump on not below/above or equal	01110011	disp
JNBE/JA Jump on not below or equal/above	01110111	disp
JNP/JPO = Jump on not par/par odd	01111011	disp
JNO = Jump on not overflow	01110001	disp
JNS Jump on not sign	01111001	disp
LOOP Loop CX times	11100010	disp
LOOPZ/LOOPE Loop while zero/equal	11100001	disp
LOOPNZ/LOOPNE Loop while not zero/equal	11100000	disp
JCXZ Jump on CX zero	11100011	disp
INT Interrupt		
Type specified	11001101	type
Type 3	11001100	
INTO = Interrupt on overflow	11001110	
IRET Interrupt return	11001111	

PROCESSOR CONTROL

CLC Clear carry	11111000	
CMC Complement carry	11110101	
STC Set carry	11111001	
CLD Clear direction	11111100	
STD Set direction	11111101	
CLI Clear interrupt	11111010	
STI Set interrupt	11111011	
HLT Halt	11110100	
WAIT Wait	10011011	
ESC Escape (to external device)	11011xxx	mod x x x r/m
LOCK Bus lock prefix	11110000	

Footnotes:

AL = 8-bit accumulator
AX = 16-bit accumulator
CX = Count register
DS = Data segment
ES = Extra segment
Above/below refers to unsigned value.
Greater = more positive:
Less = less positive (more negative) signed values
if d = 1 then "to" reg, if d = 0 then "from" reg
if w = 1 then word instruction, if w = 0 then byte instruction

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low

if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP
DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low
**MOV CS, REG/MEMORY not allowed.
Mnemonics © Intel, 1978

if s:w = 01 then 16 bits of immediate data form the operand.
if s:w = 11 then an immediate data byte is sign extended to form the 16-bit operand.
if v = 0 then "count" = 1; if v = 1 then "count" in (CL)
x = don't care
z is used for string primitives for comparison with ZF FLAG.

SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

REG is assigned according to the following table:

16-Bit (w = 1)	8-bit (w = 0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:

FLAGS = X:X:X:X:(OF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF)

231197–23

80188
HIGH INTEGRATION 8-BIT MICROPROCESSOR

- Integrated Feature Set
  - Enhanced 8088-2 CPU
  - Clock Generator
  - 2 Independent, High-Speed DMA Channels
  - Programmable Interrupt Controller
  - 3 Programmable 16-Bit Timers
  - Programmable Memory and Peripheral Chip-Select Logic
  - Programmable Wait State Generator
  - Local Bus Controller
- Available in 10 MHz (80188-10) and 8 MHz (80188) Versions
- High-Performance 8 MHz Processor
  - At 8 MHz Provides 2 Times the Performance of the Standard 80188
  - 2 MByte/Sec Bus Bandwidth Interface @8 MHz
  - 2.5 MByte/Sec Bus Bandwidth Interface @ 10 MHz
- 8-Bit Data Bus Interface; 16-Bit Internal Architecture
- Completely Object Code Compatible with All Existing iAPX 86, 88 Software
  - 10 New Instruction Types
- Direct Addressing Capability to 1 MByte of Memory and 64 KByte I/O
- Complete System Development Support
  - Development Software; Assembler, PL/M, Pascal, Fortran, and System Utilities
  - In-Circuit-Emulator (ICE™-188)
- High Performance Numerical Coprocessing Capability Through 8087 Interface
- Available in 68 Pin:
  - Ceramic Leadless Chip Carrier (LCC)
  - Ceramic Pin Grid Array (PGA)
  - Plastic Leaded Chip Carrier (PLCC)

(See Packaging Spec., Order #231369)

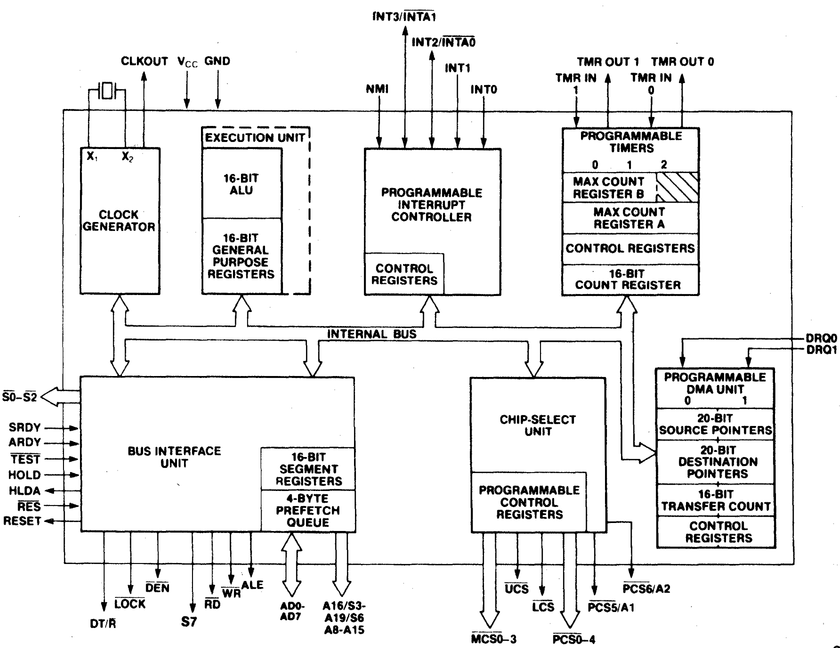


Figure 1. 80188 Block Diagram

Order Number: 210706-006

The Intel 80188 is a highly integrated microprocessor with an 8-bit data bus interface and a 16-bit internal architecture to give high performance. The 80188 effectively combines 15–20 of the most common 8088 system components onto one. The 80188 provides two times greater throughput than the standard 5 MHz 8088. The 80188 is upward compatible with 8086 and 8088 software and adds 10 new instruction types to the existing set.

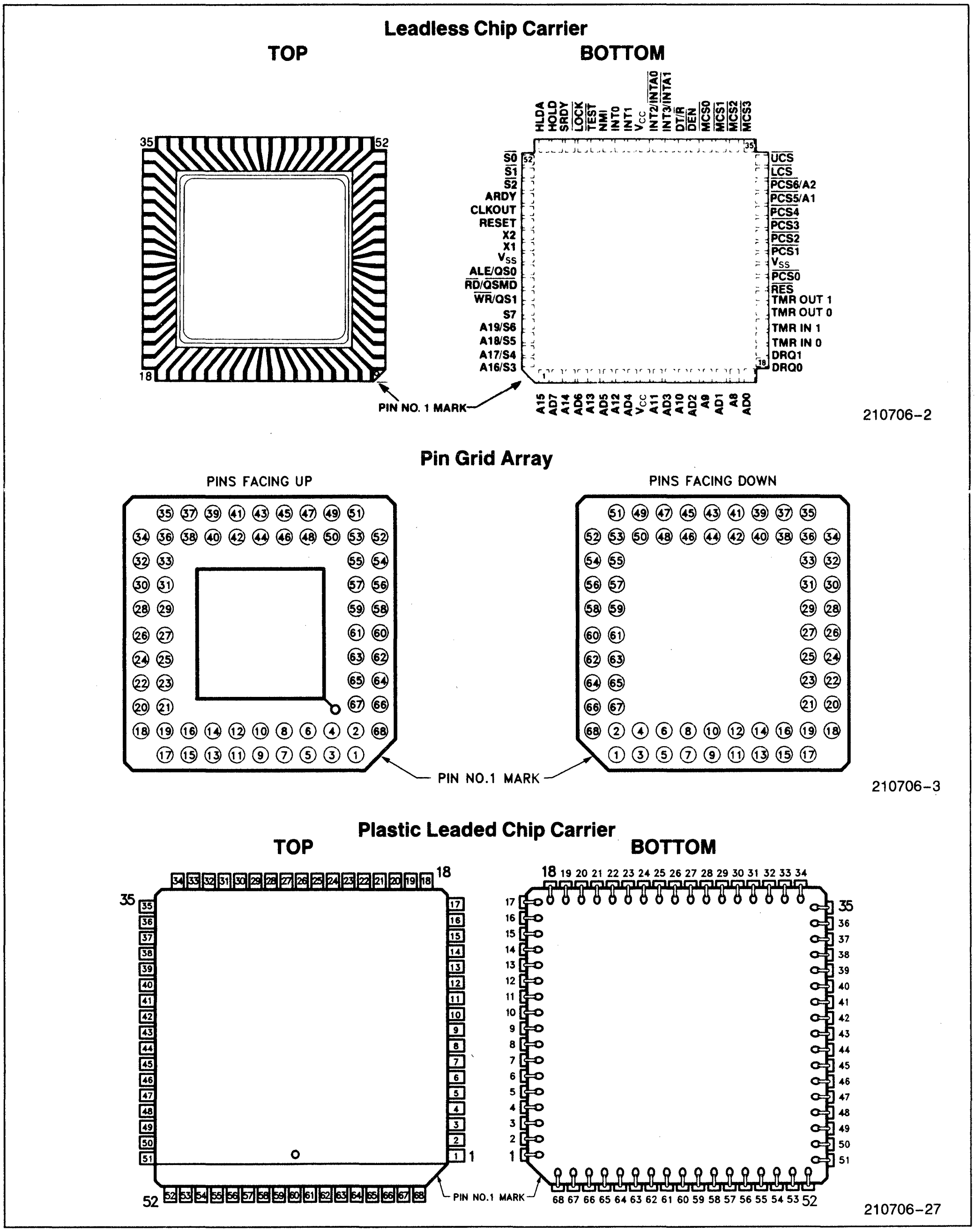


Figure 2. 80188 Pinout Diagram

Table 1. 80188 Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}, V_{CC}	9, 43	I	**SYSTEM POWER:** +5 volt power supply.
V_{SS}, V_{SS}	26, 60	I	**SYSTEM GROUND**
RESET	57	O	**RESET OUTPUT:** Indicates that the 80188 CPU is being reset, and can be used as a system reset. It is active HIGH, synchronized with the processor clock, and lasts an integer number of clock periods corresponding to the length of the $\overline{\text{RES}}$ signal.
X1, X2	59, 58	I	**CRYSTAL INPUTS:** X1 and X2 provide an external connection for a fundamental mode parallel resonant crystal for the internal crystal oscillator. X1 can interface to an external clock instead of a crystal. In this case, minimize the capacitance on X2 or drive X2 with complemented X1. The input or oscillator frequency is internally divided by two to generate the clock signal (CLKOUT).
CLKOUT	56	O	**CLOCK OUTPUT:** Provides the system with a 50% duty cycle waveform. All device pin timings are specified relative to CLKOUT. CLKOUT has sufficient MOS drive capabilities for the 8087 Numeric Processor Extension.
$\overline{\text{RES}}$	24	I	**SYSTEM RESET:** Causes the 80188 to immediately terminate its present activity, clear the internal logic, and enter a dormant state. This signal may be asynchronous to the 80188 clock. The 80188 begins fetching instructions approximately 7 clock cycles after $\overline{\text{RES}}$ is returned HIGH. $\overline{\text{RES}}$ is required to be LOW for greater than 4 clock cycles and is internally synchronized. For proper initialization, the LOW-to-HIGH transition of $\overline{\text{RES}}$ must occur no sooner than 50 microseconds after power up. This input is provided with a Schmitt-trigger to facilitate power-on $\overline{\text{RES}}$ generation via an RC network. When $\overline{\text{RES}}$ occurs, the 80188 will drive the status lines to an inactive level for one clock, and then tri-state them.
$\overline{\text{TEST}}$	47	I	**$\overline{\text{TEST}}$:** Is examined by the WAIT instruction. If the $\overline{\text{TEST}}$ input is HIGH when "WAIT" execution begins, instruction execution will suspend. $\overline{\text{TEST}}$ will be resampled until it goes LOW, at which time execution will resume. If interrupts are enabled while the 80188 is waiting for $\overline{\text{TEST}}$, interrupts will be serviced. This input is synchronized internally.
TMR IN 0, TMR IN 1	20 21	I I	**TIMER INPUTS:** Are used either as clock or control signals, depending upon the programmed timer mode. These inputs are active HIGH (or LOW-to-HIGH transitions are counted) and internally synchronized.
TMR OUT 0, TMR OUT 1	22 23	O O	**TIMER OUTPUTS:** Are used to provide single pulse or continuous waveform generation, depending upon the timer mode selected.
DRQ0, DRQ1	18 19	I I	**DMA REQUEST:** Is driven HIGH by an external device when it desires that a DMA channel (Channel 0 or 1) perform a transfer. These signals are active HIGH, level-triggered, and internally synchronized.
NMI	46	I	**NON-MASKABLE INTERRUPT:** Is an edge-triggered input which causes a type 2 interrupt. NMI is not maskable internally. A transition from a LOW to HIGH initiates the interrupt at the next instruction boundary. NMI is latched internally. An NMI duration of one clock or more will guarantee service. This input is internally synchronized.

Table 1. 80188 Pin Description (Continued)

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>INT0, INT1,
INT2/$\overline{\text{INTA0}}$,
INT3/$\overline{\text{INTA1}}$</td><td>45, 44
42
41</td><td>I
I/O
I/O</td><td>MASKABLE INTERRUPT REQUESTS: Can be requested by strobing one of these pins. When configured as inputs, these pins are active HIGH. Interrupt Requests are synchronized internally. INT2 and INT3 may be configured via software to provide active-LOW interrupt-acknowledge output signals. All interrupt inputs may be configured via software to be either edge- or level-triggered. To ensure recognition, all interrupt requests must remain active until the interrupt is acknowledged. When iRMX mode is selected, the function of these pins changes (see Interrupt Controller section of this data sheet).</td></tr>
<tr><td>A19/S6,
A18/S5,
A17/S4,
A16/S3</td><td>65
66
67
68</td><td>O
O
O
O</td><td>ADDRESS BUS OUTPUTS (16–19) and BUS CYCLE STATUS (3–6): Reflect the four most significant address bits during T_1. These signals are active HIGH. During T_2, T_3, T_W, and T_4, status information is available on these lines as encoded below:<table><tr><th></th><th>Low</th><th>High</th></tr><tr><td>S6</td><td>Processor Cycle</td><td>DMA Cycle</td></tr></table>S3, S4, and S5 are defined as LOW during T_2–T_4.</td></tr>
<tr><td>AD7–AD0</td><td>2, 4, 6, 8
11, 13, 15, 17</td><td>I/O</td><td>ADDRESS/DATA BUS (0-7): Signals constitute the time multiplexed memory or I/O address (T_1) and data (T_2, T_3, T_W, and T_4) bus. The bus is active HIGH.</td></tr>
<tr><td>A15–A8</td><td>1, 3, 5, 7
10, 12, 14, 16</td><td>O</td><td>ADDRESS-ONLY BUS (8-15): Containing valid address from T_1-T_4. The bus is active HIGH.</td></tr>
<tr><td>S7</td><td>64</td><td>O</td><td>This signal is always HIGH to indicate that the 80188 has an 8-bit data bus, and is tri-state OFF during bus HOLD.</td></tr>
<tr><td>ALE/QS0</td><td>61</td><td>O</td><td>ADDRESS LATCH ENABLE/QUEUE STATUS 0: Is provided by the 80188 to latch the address into the 8282/8283 address latches. ALE is active HIGH. Addresses are guaranteed to be valid on the trailing edge of ALE. The ALE rising edge is generated off the rising edge of the CLKOUT immediately preceding T_1 of the associated bus cycle, effectively one-half clock cycle earlier than in the standard 8088. The trailing edge is generated off the CLKOUT rising edge in T_1 as in the 8088. Note that ALE is never floated.</td></tr>
<tr><td>$\overline{\text{WR}}$/QS1</td><td>63</td><td>O</td><td>WRITE STROBE/QUEUE STATUS 1: Indicates that the data on the bus is to be written into a memory or an I/O device. $\overline{\text{WR}}$ is active for T_2, T_3, and T_W of any write cycle. It is active LOW, and floats during "HOLD." It is driven HIGH for one clock during Reset, and then floated. When the 80188 is in queue status mode, the ALE/QS0 and $\overline{\text{WR}}$/QS1 pins provide information about processor/instruction queue interaction.<table><tr><th>QS1</th><th>QS0</th><th>Queue Operation</th></tr><tr><td>0</td><td>0</td><td>No Queue Operation</td></tr><tr><td>0</td><td>1</td><td>First Opcode Byte Fetched from the Queue</td></tr><tr><td>1</td><td>1</td><td>Subsequent Byte Fetched from the Queue</td></tr><tr><td>1</td><td>0</td><td>Empty the Queue</td></tr></table></td></tr>
</table>

Table 1. 80188 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{RD}/\overline{QSMD}$	62	O	**READ STROBE:** Indicates that the 80188 is performing a memory or I/O read cycle. $\overline{RD}$ is active LOW for T_2, T_3, and T_W of any read cycle. It is guaranteed not to go LOW in T_2 until after the Address Bus is floated. $\overline{RD}$ is active LOW, and floats during "HOLD". $\overline{RD}$ is driven HIGH for one clock during Reset, and then the output driver is floated. A weak internal pull-up mechanism on the $\overline{RD}$ line holds it HIGH when the line is not driven. During RESET the pin is sampled to determine whether the 80188 should provide ALE, $\overline{WR}$, and $\overline{RD}$, or if the Queue-Status should be provided. $\overline{RD}$ should be connected to GND to provide Queue-Status data.
ARDY	55	I	**ASYNCHRONOUS READY:** Informs the 80188 that the addressed memory space or I/O device will complete a data transfer. The ARDY input pin will accept an asynchronous input, and is active HIGH. Only the rising edge is internally synchronized by the 80188. This means that the falling edge of ARDY must be synchronized to the 80188 clock. If connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active to terminate a bus cycle. If unused, this line should be tied LOW.
SRDY	49	I	**SYNCHRONOUS READY:** Must be synchronized externally to the 80188. The use of SRDY provides a relaxed system-timing specification on the Ready input. This is accomplished by eliminating the one-half clock cycle which is required for internally resolving the signal level when using the ARDY input. This line is active HIGH. If this line is connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active before a bus cycle is terminated. If unused, this line should be tied LOW.
$\overline{LOCK}$	48	O	**$\overline{LOCK}$:** Output indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active LOW. The $\overline{LOCK}$ signal is requested by the LOCK prefix instruction and is activated at the beginning of the first data cycle associated with the instruction following the LOCK prefix. It remains active until the completion of the instruction following the LOCK prefix. No prefetches will occur while $\overline{LOCK}$ is asserted. When executing more than one LOCK instruction, always make sure there are 6 bytes of code between the end of the first LOCK instruction and the start of the second LOCK instruction. $\overline{LOCK}$ is active LOW, is driven HIGH for one clock during RESET, and then floated.
$\overline{S0}$, $\overline{S1}$, $\overline{S2}$	52–54	O	**BUS CYCLE STATUS $\overline{S0}$–$\overline{S2}$:** Are encoded to provide bus-transaction information:

80188 Bus Cycle Status Information

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

The status pins float during "HOLD."
$\overline{S2}$ may be used as a logical M/$\overline{IO}$ indicator, and $\overline{S1}$ as a DT/$\overline{R}$ indicator.
The status lines are driven HIGH for one clock during Reset, and then floated until a bus cycle begins.

Table 1. **80188 Pin Description** (Continued)

Symbol	Pin No.	Type	Name and Function
HOLD (input) HLDA (output)	50 51	I O	**HOLD:** Indicates that another bus master is requesting the local bus. The HOLD input is active HIGH. HOLD may be asynchronous with respect to the 80188 clock. The 80188 will issue a HLDA in response to a HOLD request at the end of T_4 or T_I. Simultaneous with the issuance of HLDA, the 80188 will float the local bus and control lines. After HOLD is detected as being LOW, the 80188 will lower HLDA. When the 80188 needs to run another bus cycle, it will again drive the local bus and control lines.
$\overline{UCS}$	34	O	**UPPER MEMORY CHIP SELECT:** Is an active LOW output whenever a memory reference is made to the defined upper portion (1K–256K block) of memory. This line is not floated during bus HOLD. The address range activating $\overline{UCS}$ is software programmable.
$\overline{LCS}$	33	O	**LOWER MEMORY CHIP SELECT:** Is active LOW whenever a memory reference is made to the defined lower portion (1K–256K) of memory. This line is not floated during bus HOLD. The address range activating $\overline{LCS}$ is software programmable.
$\overline{MCS0-3}$	38, 37, 36, 35	O	**MID-RANGE MEMORY CHIP SELECT SIGNALS:** Are active LOW when a memory reference is made to the defined mid-range portion of memory (8K–512K). These lines are not floated during bus HOLD. The address ranges activating $\overline{MCS0-3}$ are software programmable.
$\overline{PCS0-4}$	25, 27-30	O	**PERIPHERAL CHIP SELECT SIGNALS 0–4:** Are active LOW when a reference is made to the defined peripheral area (64K byte I/O space). These lines are not floated during bus HOLD. The address ranges activating $\overline{PCS0-4}$ are software programmable.
$\overline{PCS5}$/A1	31	O	**PERIPHERAL CHIP SELECT 5 or LATCHED A1:** May be programmed to provide a sixth peripheral chip select, or to provide an internally latched A1 signal. The address range activating $\overline{PCS5}$ is software programmable. When programmed to provide latched A1, rather than $\overline{PCS5}$, this pin will retain the previously latched value of A1 during a bus HOLD. A1 is active HIGH.
$\overline{PCS6}$/A2	32	O	**PERIPHERAL CHIP SELECT 6 or LATCHED A2:** May be programmed to provide a seventh peripheral chip select, or to provide an internally latched A2 signal. The address range activating $\overline{PCS6}$ is software programmable. When programmed to provide latched A2, rather than $\overline{PCS6}$, this pin will retain the previously latched value of A2 during a bus HOLD. A2 is active HIGH.
DT/$\overline{R}$	40	O	**DATA TRANSMIT/RECEIVE:** Controls the direction of data flow through the external 8286/8287 data bus transceiver. When LOW, data is transferred to the 80188. When HIGH the 80188 places write data on the data bus.
$\overline{DEN}$	39	O	**DATA ENABLE:** Is provided as an 8286/8287 data bus transceiver output enable. $\overline{DEN}$ is active LOW during each memory and I/O access. $\overline{DEN}$ is HIGH whenever DT/$\overline{R}$ changes state.

FUNCTIONAL DESCRIPTION

Introduction

The following Functional Description describes the base architecture of the 80188. This architecture is common to the 8086, 8088 and 80286 microprocessor families as well. The 80188 is a very high integration 8-bit microprocessor. It combines 15–20 of the most common microprocessor system components onto one chip while providing twice the performance of the standard 8088. The 80188 is object code compatible with the 8086, 8088 microprocessors and adds 10 new instruction types to the existing 8086, 8088 instruction set.

80188 BASE ARCHITECTURE

The 8086, 8088, 80186, 80188 and 80286 family all contain the same basic set of registers, instructions, and addressing modes. The 80188 processor is upward compatible with the 8086, 8088, 80186, and 80286 CPUs.

Register Set

The 80188 base architecture has fourteen registers as shown in Figures 3a and 3b. These registers are grouped into the following categories.

GENERAL REGISTERS

Eight 16-bit general purpose registers may be used to contain arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used as 16-bit registers or split into pairs of separate 8-bit registers.

SEGMENT REGISTERS

Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

BASE AND INDEX REGISTERS

Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode selects the specific registers for operand and address calculations.

STATUS AND CONTROL REGISTERS

Two 16-bit special purpose registers record or alter certain aspects of the 80188 processor state. These are the Instruction Pointer Register, which contains the offset address of the next sequential instruction to be executed, and the Status Word Register, which contains status and control flag bits (see Figures 3a and 3b).

STATUS WORD DESCRIPTION

The Status Word records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80188 within a given operating mode (bits 8, 9, and 10). The Status Word Register is 16-bits wide. The function of the Status Word bits is shown in Table 2.

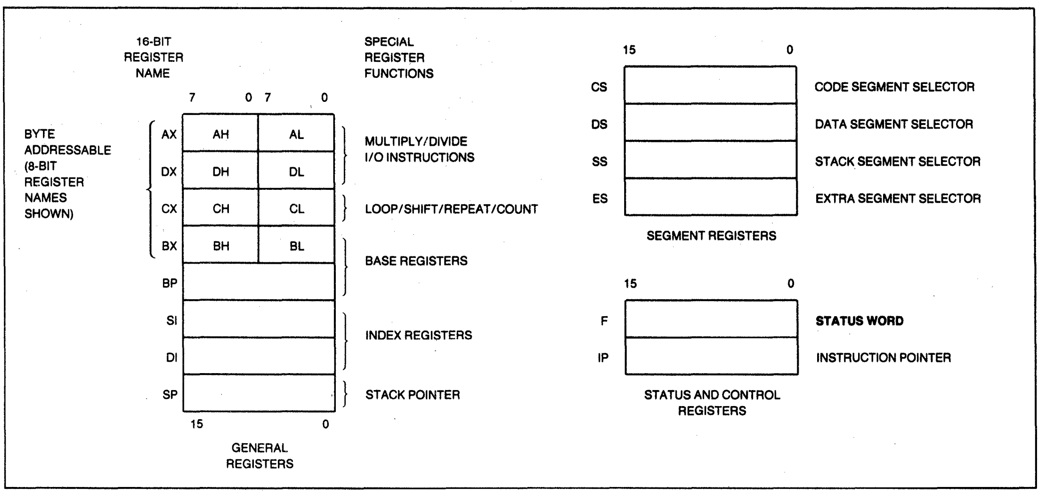


Figure 3a. 80188 General Purpose Register Set

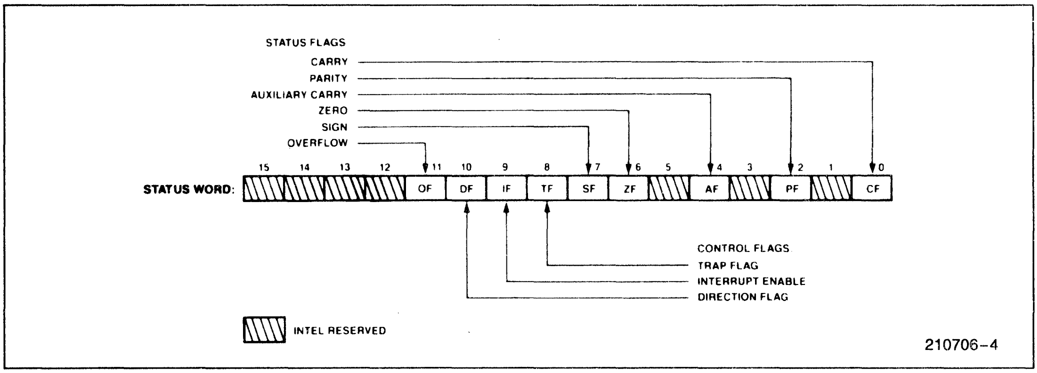


Figure 3b. Status Word Format

Table 2. Status Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag—Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-Enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index register when set. Clearing DF causes auto increment.
11	OF	Overflow Flag—Set if the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high-level instructions, and processor control. These categories are summarized in Figure 4.

An 80188 instruction can reference anywhere from zero to several operands. An operand can reside in a register, in the instruction itself, or in memory. Specific operand addressing modes are discussed later in this data sheet.

Memory Organization

Memory is organized in sets of segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two-component address (a pointer) that consists of a 16-bit base segment and a 16-bit offset. The 16-bit base values are contained in one of four internal segment registers (code, data, stack, extra). The physical address is calculated by shifting the base value LEFT by four bits and adding the 16-bit offset value to yield a 20-bit physical address (see Figure 5). This allows for a 1 MByte physical address size.

All instructions that address operands in memory must specify the base segment and the 16-bit offset value. For speed and compact instruction encoding, the segment register used for physical address generation is implied by the addressing mode used (see Table 3). These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs.

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack
ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word
FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{TEST}}$ pin active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
HIGH LEVEL INSTRUCTIONS	
ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4. 80188 Instruction Set

CONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign

JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign
UNCONDITIONAL TRANSFERS	
CALL	Call procedure
RET	Return from procedure
JMP	Jump
ITERATION CONTROLS	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCXZ	Jump if register CX = 0
INTERRUPTS	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Interrupt return

Figure 4. 80188 Instruction Set (Continued)

To access operands that do not reside in one of the four immediately available segments, a full 32-bit pointer can be used to reload both the base (segment) and offset values.

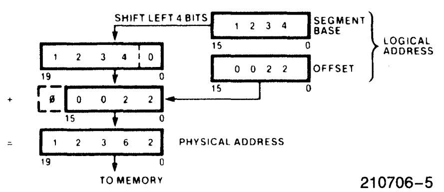


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Instruction prefetch and immediate data.
Stack	Stack (SS)	All stack pushes and pops; any memory references which use BP Register as a base register.
External Data (Global)	Extra (ES)	All string instruction references which use the DI register as an index.
Local Data	Data (DS)	All other data references.

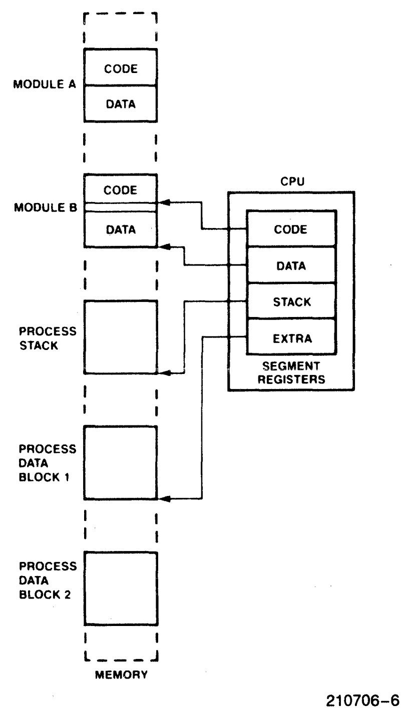


Figure 6. Segmented Memory Helps Structure Software

Addressing Modes

The 80188 provides eight categories of addressing modes to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

- *Register Operand Mode:* The operand is located in one of the 8- or 16-bit general registers.
- *Immediate Operand Mode:* The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: a segment base and an offset. The segment base is supplied by a 16-bit segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset, also called the effective address, is calculated by summing any combination of the following three address elements:

- the *displacement* (an 8- or 16-bit immediate value contained in the instruction);
- the *base* (contents of either the BX or BP base registers); and
- the *index* (contents of either the SI or DI index registers).

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

- *Direct Mode:* The operand's offset is contained in the instruction as an 8- or 16-bit displacement element.
- *Register Indirect Mode:* The operand's offset is in one of the registers SI, DI, BX, or BP.
- *Based Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of a base register (BX or BP).
- *Indexed Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of an index register (SI or DI).
- *Based Indexed Mode:* The operand's offset is the sum of the contents of a base register and an index register.
- *Based Indexed Mode with Displacement:* The operand's offset is the sum of a base register's contents, an index register's contents, and an 8- or 16-bit displacement.

Data Types

The 80188 directly supports the following data types:

- *Integer:* A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32- and 64-bit integers are supported using the 80188/20 Numeric Data Processor.
- *Ordinal:* An unsigned binary numeric value contained in an 8-bit byte or a 16-bit word.
- *Pointer:* A 16- or 32-bit quantity, composed of a 16-bit offset component or a 16-bit segment base component in addition to a 16-bit offset component.
- *String:* A contiguous sequence of bytes or words. A string may contain from 1 to 64K bytes.
- *ASCII:* A byte representation of alphanumeric and control characters using the ASCII standard of character representation.
- *BCD:* A byte (unpacked) representation of the decimal digits 0–9.
- *Packed BCD:* A byte (packed) representation of two decimal digits (0–9). One digit is stored in each nibble (4-bits) of the byte.
- *Floating Point:* A signed 32-, 64-, or 80-bit real number representation. (Floating point operands are supported using the 80188/20 Numeric Data Processor configuration.)

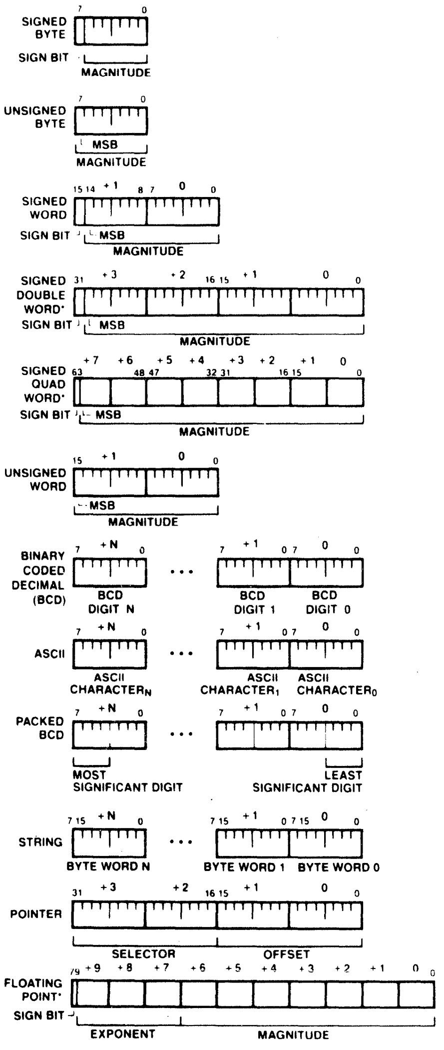
In general, individual data elements must fit within defined segment limits. Figure 7 graphically represents the data types supported by the 80188.

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. Separate instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A_{15}–A_8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Status Word) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable.



NOTE:
*Supported By 80188/20 Numeric Data Processor Configuration.

Figure 7. 80188 Supported Data Types

Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. If the exception was caused by executing an ESC instruction with the ESC trap bit set in the relocation register, the return instruction will point to the ESC instruction, or to the segment override prefix immediately preceding the ESC instruction if the prefix was present. In all other cases, the return address from an exception will point at the instruction immediately following the instruction causing the exception.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. Table 4 shows the 80188 predefined types and default priority levels. For each interrupt, an 8-bit vector must be supplied to the 80188 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. In addition, internal peripherals and noncascaded external interrupts will generate their own vectors through the internal interrupt controller. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

Interrupt Sources

The 80188 can service interrupts generated by software or hardware. The software interrupts are generated by specific instructions (INT, ESC, unused OP, etc.) or the results of conditions specified by instructions (array bounds check, INTO, DIV, IDIV, etc.). All interrupt sources are serviced by an indirect call through an element of a vector table. This vector table is indexed by using the interrupt vector type (Table 4), multiplied by four. All hardware-generated interrupts are sampled at the end of each instruction. Thus, the software interrupts will begin service first. Once the service routine is entered and interrupts are enabled, any hardware source of sufficient priority can interrupt the service routine in progress.

The software generated 80188 interrupts are described below.

DIVIDE ERROR EXCEPTION (TYPE 0)

Generated when a DIV or IDIV instruction quotient cannot be expressed in the number of bits in the destination.

SINGLE-STEP INTERRUPT (TYPE 1)

Generated after most instructions if the TF flag is set. Interrupts will not be generated after prefix instructions (e.g., REP), instructions which modify segment registers (e.g., POP DS), or the WAIT instruction.

NON-MASKABLE INTERRUPT—NMI (TYPE 2)

An external interrupt source which cannot be masked.

Table 4. 80188 Interrupt Vectors

Interrupt Name	Vector Type	Default Priority	Related Instructions
Divide Error Exception	0	*1	DIV, IDIV
Single Step Interrupt	1	12**2	All
NMI	2	1	All
Breakpoint Interrupt	3	*1	INT
INTO Detected Overflow Exception	4	*1	INTO
Array Bounds Exception	5	*1	BOUND
Unused-Opcode Exception	6	*1	Undefined Opcodes
ESC Opcode Exception	7	*1***	ESC Opcodes
Timer 0 Interrupt	8	2A****	
Timer 1 Interrupt	18	2B****	
Timer 2 Interrupt	19	2C****	
Reserved	9	3	
DMA 0 Interrupt	10	4	
DMA 1 Interrupt	11	5	
INT0 Interrupt	12	6	
INT1 Interrupt	13	7	
INT2 Interrupt	14	8	
INT3 Interrupt	15	9	

NOTES:
*1. These are generated as the result of an instruction execution.
**2. This is handled as in the 8088.
****3. All three timers constitute one source of request to the interrupt controller. The Timer Interrupts all have the same default priority level with respect to all other interrupt sources. However, they have a defined priority ordering amongst themselves. (Priority 2A is higher priority than 2B.) Each Timer Interrupt has a separate vector type number.
4. Default priorities for the interrupt sources are used only if the user does not program each source into a unique priority level.
***5. An escape opcode will cause a trap only if the proper bit is set in the peripheral control block relocation register.

BREAKPOINT INTERRUPT (TYPE 3)

A one-byte version of the INT instruction. It uses 12 as an index into the service routine address table (because it is a type 3 interrupt).

INTO DETECTED OVERFLOW EXCEPTION (TYPE 4)

Generated during an INTO instruction if the OF bit is set.

ARRAY BOUNDS EXCEPTION (TYPE 5)

Generated during a BOUND instruction if the array index is outside the array bounds. The array bounds are located in memory at a location indicated by one of the instruction operands. The other operand indicates the value of the index to be checked.

UNUSED OPCODE EXCEPTION (TYPE 6)

Generated if execution is attempted on undefined opcodes.

ESCAPE OPCODE EXCEPTION (TYPE 7)

Generated if execution is attempted of ESC opcodes (D8H–DFH). This exception will only be generated if a bit in the relocation register is set. The return address of this exception will point to the ESC instruction causing the exception. If a segment override prefix preceded the ESC instruction, the return address will point to the segment override prefix.

Hardware-generated interrupts are divided into two groups: maskable interrupts and non-maskable interrupts. The 80188 provides maskable hardware interrupt request pins INT0–INT3. In addition, maskable interrupts may be generated by the 80188 integrated DMA controller and the integrated timer unit. The vector types for these interrupts are shown in Table 4. Software enables these inputs by setting the Interrupt Flag bit (IF) in the Status Word. The interrupt controller is discussed in the peripheral section of this data sheet.

Further maskable interrupts are disabled while servicing an interrupt because the IF bit is reset as part of the response to an interrupt or exception. The saved Status Word will reflect the enable status of the processor prior to the interrupt. The interrupt flag will remain zero unless specifically set. The interrupt return instruction restores the Status Word, thereby restoring the original status of IF bit. If the interrupt return re-enables interrupts, and another interrupt is pending, the 80188 will immediately service the highest-priority interrupt pending, i.e., no instructions of the main line program will be executed.

Non-Maskable Interrupt Request (NMI)

A non-maskable interrupt (NMI) is also provided. This interrupt is serviced regardless of the state of the IF bit. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed. The IF bit is cleared at the beginning of an NMI interrupt to prevent maskable interrupts from being serviced.

Single-Step Interrupt

The 80188 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single-step interrupt and is controlled by the single-step flag bit (TF) in the Status Word. Once this bit is set, an internal single-step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single-stepped.

Initialization and Processor Reset

Processor initialization or startup is accomplished by driving the $\overline{RES}$ input pin LOW. $\overline{RES}$ forces the 80188 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as $\overline{RES}$ is active. After $\overline{RES}$ becomes inactive and an internal processing interval elapses, the 80188 begins execution with the instruction at physical location FFFF0(H). $\overline{RES}$ also sets some registers to predefined values as shown in Table 5.

Table 5. 80188 Initial Register State after RESET

Status Word	F002(H)
Instruction Pointer	0000(H)
Code Segment	FFFF(H)
Data Segment	0000(H)
Extra Segment	0000(H)
Stack Segment	0000(H)
Relocation Register	20FF(H)
UMCS	FFFB(H)

THE 80188 COMPARED TO THE 80186

The 80188 CPU is an 8-bit processor designed around the 80186 internal structure. Most internal functions of the 80188 are identical to the equivalent 80186 functions. The 80188 handles the external bus the same way the 80186 does with the distinction of handling only 8 bits at a time. Sixteen bit operands are fetched or written in two consecutive bus cycles. Both processors will appear identical to the software engineer, with the exception of execution time. The internal register structure is identical and all instructions have the same end result. The differences between the 80188 and the 80186 are outlined below. Internally, there are three differences between the 80188 and the 80186. All changes are related to the 8-bit bus interface.

- The queue length is 4 bytes in the 80188, whereas the 80186 queue contains 6 bytes, or three words. The queue was shortened to prevent overuse of the bus by the BIU when prefetching instructions. This was required because of the additional time necessary to fetch instructions 8 bits at a time.
- To further optimize the queue, the prefetching algorithm was changed. The 80188 BIU will fetch a new instruction to load into the queue each time there is a 1-byte hole (space available) in the queue. The 80186 waits until a 2-byte space is available.
- The internal execution time of the instruction is affected by the 8-bit interface. All 16-bit fetches and writes from/to memory take an additional four clock cycles. The CPU may also be limited by the speed of instruction fetches when a series of simple operations occur. When the more sophisticated instructions of the 80188 are being used, the queue has time to fill and the execution proceeds as fast as the execution unit will allow.

The 80188 and 80186 are completely software compatible by virtue of their identical execution units. Software that is system dependent may not be completely transferable, but software that is not system dependent will operate equally well on an 80188 or an 80186.

The hardware interface of the 80188 contains the major differences between the two CPUs. The pin assignments are nearly identical, however, with the following functional changes.

- A8–A15—These pins are only address outputs on the 80188. These address lines are latched internally and remain valid throughout a bus cycle in a manner similar to the 8085 upper address lines.
- $\overline{BHE}$ has no meaning on the 80188 and has been eliminated.

80188 Clock Generator

The 80188 provides an on-chip clock generator for both internal and external clock generation. The clock generator features a crystal oscillator, a divide-by-two counter, synchronous and asynchronous ready inputs, and reset circuitry.

Oscillator

The oscillator circuit of the 80188 is designed to be used with a parallel resonant fundamental mode crystal. This is used as the time base for the 80188. The crystal frequency selected will be double the CPU clock frequency. Use of an LC or RC circuit is not recommended with this oscillator. If an external oscillator is used, it can be connected directly to input pin X1 in lieu of a crystal. The output of the oscillator is not directly available outside the 80188. The recommended crystal configuration is shown in Figure 8.

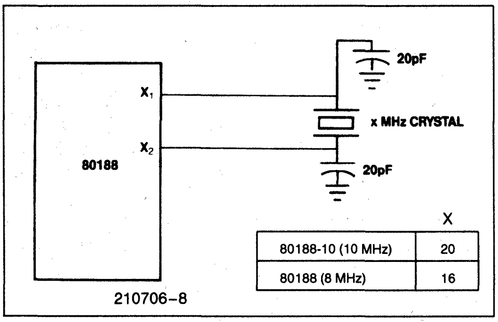


Figure 8. Recommended 80188 Crystal Configuration

The following parameters may be used for choosing a crystal:

Temperature Range:	0 to 70°C
ESR (Equivalent Series Resistance):	30Ω max
C_0 (Shunt Capacitance of Crystal):	7.0 pf max
C_L (Load Capacitance):	20 pf ±2 pf
Drive Level:	1 mW max

Clock Generator

The 80188 clock generator provides the 50% duty cycle processor clock for the 80188. It does this by dividing the oscillator output by 2 forming the symmetrical clock. If an external oscillator is used, the state of the clock generator will change on the falling edge of the oscillator signal. The CLKOUT pin provides the processor clock signal for use outside the 80188. This may be used to drive other system components. All timings are referenced to the output clock.

READY Synchronization

The 80188 provides both synchronous and asynchronous ready inputs. Asynchronous ready synchronization is accomplished by circuitry which samples ARDY in the middle of T_2, T_3 and again in the middle of each T_W until ARDY is sampled HIGH. One-half CLKOUT cycle of resolution time is used. Full synchronization is performed only on the rising edge of ARDY, i.e., the falling edge of ARDY must be synchronized to the CLKOUT signal if it will occur during T_2, T_3, or T_W. HIGH-to-LOW transitions of ARDY must be performed synchronously to the CPU clock.

A second ready input (SRDY) is provided to interface with externally synchronized ready signals. This input is sampled at the end of T_2, T_3 and again at the end of each T_W until it is sampled HIGH. By using this input rather than the asynchronous ready input, the half-clock cycle resolution time penalty is eliminated.

This input must satisfy set-up and hold times to guarantee proper operation of the circuit.

In addition, the 80188, as part of the integrated chip-select logic, has the capability to program WAIT states for memory and peripheral blocks. This is discussed in the Chip Select/Ready Logic description.

RESET Logic

The 80188 provides both a $\overline{\text{RES}}$ input pin and a synchronized RESET pin for use with other system components. The $\overline{\text{RES}}$ input pin on the 80188 is provided with hysteresis in order to facilitate power-on Reset generation via an RC network. RESET is guaranteed to remain active for at least five clocks given a $\overline{\text{RES}}$ input of at least six clocks. RESET may be delayed up to two and one-half clocks behind $\overline{\text{RES}}$.

Multiple 80188 processors may be synchronized through the $\overline{\text{RES}}$ input pin, since this input resets both the processor and divide-by-two internal counter in the clock generator. In order to insure that the divide-by-two counters all begin counting at the same time, the active going edge of $\overline{\text{RES}}$ must satisfy a 25 ns setup time before the falling edge of the

80188 clock input. In addition, in order to insure that all CPUs begin executing in the same clock cycle, the reset must satisfy a 25 ns setup time before the rising edge of the CLKOUT signal of all the processors.

LOCAL BUS CONTROLLER

The 80188 provides a local bus controller to generate the local bus control signals. In addition, it employs a HOLD/HLDA protocol for relinquishing the local bus to other bus masters. It also provides control lines that can be used to enable external buffers and to direct the flow of data on and off the local bus.

Memory/Peripheral Control

The 80188 provides ALE, RD, and WR bus control signals. The RD and WR signals are used to strobe data from memory to the 80188 or to strobe data from the 80188 to memory. The ALE line provides a strobe to address latches for the multiplexed address/data bus. The 80188 local bus controller does not provide a memory/I/O signal. If this is required, the user will have to use the S2 signal (which will require external latching), make the memory and I/O spaces nonoverlapping, or use only the integrated chip-select circuitry.

Transceiver Control

The 80188 generates two control signals to be connected to 8286/8287 transceiver chips. This capability allows the addition of transceivers for extra buffering without adding external logic. These control lines, DT/R and DEN, are generated to control the flow of data through the transceivers. The operation of these signals is shown in Table 6.

Table 6. Transceiver Control Signals Description

Pin Name	Function
DEN (Data Enable)	Enables the output drivers of the transceivers. It is active LOW during memory, I/O, or INTA cycles.
DT/R (Data Transmit/ Receive)	Determines the direction of travel through the transceivers. A HIGH level directs data away from the processor during write operations, while a LOW level directs data toward the processor during a read operation.

Local Bus Arbitration

The 80188 uses a HOLD/HLDA system of local bus exchange. This provides an asynchronous bus exchange mechanism. This means multiple masters utilizing the same bus can operate at separate clock frequencies. The 80188 provides a single HOLD/HLDA pair through which all other bus masters may gain control of the local bus. This requires external circuitry to arbitrate which external device will gain control of the bus from the 80188 when there is more than one alternate local bus master. When the 80188 relinquishes control of the local bus, it floats DEN, RD, WR, S0–S2, LOCK, AD0–AD15, A16–A19, S7, and DT/R to allow another master to drive these lines directly.

The 80188 HOLD latency time, i.e., the time between HOLD request and HOLD acknowledge, is a function of the activity occurring in the processor when the HOLD request is received. A HOLD request is the highest-priority activity request which the processor may receive: higher than instruction fetching or internal DMA cycles. However, if a DMA cycle is in progress, the 80188 will complete the transfer before relinquishing the bus. This implies that if a HOLD request is received just as a DMA transfer begins, the HOLD latency time can be as great as 4 bus cycles. This will occur if a DMA word transfer operation is taking place from an odd address to an odd address. This is a total of 16 clocks or more, if WAIT states are required. In addition, if locked transfers are performed, the HOLD latency time will be increased by the length of the locked transfer.

Local Bus Controller and Reset

Upon receipt of a RESET pulse from the RES input, the local bus controller will perform the following actions:

- Drive DEN, RD, and WR HIGH for one clock cycle, then float.

NOTE:
RD is also provided with an internal pull-up device to prevent the processor from inadvertently entering Queue Status mode during reset.

- Drive S0–S2 to the passive state (all HIGH) and then float.
- Drive LOCK HIGH and then float.
- Float AD0–7, A8–19, S7, DT/R.
- Drive ALE LOW (ALE is never floated).
- Drive HLDA LOW.

INTERNAL PERIPHERAL INTERFACE

All the 80188 integrated peripherals are controlled via 16-bit registers contained within an internal 256-byte control block. This control block may be mapped into either memory or I/O space. Internal logic will recognize the address and respond to the bus cycle. During bus cycles to internal registers, the bus controller will signal the operation externally (i.e., the $\overline{RD}$, $\overline{WR}$, status, address, data, etc., lines will be driven as in a normal bus cycle), but D_{7-0}, SRDY, and ARDY will be ignored. The base address of the control block must be on an even 256-byte boundary (i.e., the lower 8 bits of the base address are all zeros). All of the defined registers within this control block may be read or written by the 80188 CPU at any time. The location of any register contained within the 256-byte control block is determined by the current base address of the control block.

The control block base address is programmed via a 16-bit relocation register contained within the control block at offset FEH from the base address of the control block (see Figure 9). It provides the upper 12 bits of the base address of the control block. Note that mapping the control register block into an address range corresponding to a chip-select range is not recommended (the chip select circuitry is discussed later in this data sheet. In addition, bit 12 of this register determines whether the control block will be mapped into I/O or memory space. If this bit is 1, the control block will be located in memory space, whereas if the bit is 0, the control block will be located in I/O space. If the control register block is mapped into I/O space, the upper 4 bits of the base address must be programmed as 0 (since I/O addresses are only 16 bits wide).

Whenever mapping the 188 peripheral control block to another location, the programming of the relocation register should be done with a byte write (i.e. OUT DX,AL). Any access to the control block is done 16 bits at a time. Thus, internally, the relocation register will get written with 16 bits of the AX register while externally, the BIU will run only one 8 bit bus cycle. If a word instruction is used (i.e. OUT DX,AX), the relocation register will be written on the first bus cycle. The BIU will then run a second bus cycle which is unnecessary. The address of the second bus cycle will no longer be within the control block (i.e. the control block was moved on the first cycle), and therefore, will require the generation of an external ready signal to complete the cycle. For this reason we recommend byte operations to the relocation register. Byte instructions may also be used for the other registers in the control block and will eliminate half of the bus cycles required if a word operation had been specified. Byte operations are only valid on even addresses though, and are undefined on odd addresses.

In addition to providing relocation information for the control block, the relocation register contains bits which place the interrupt controller into iRMX mode, and cause the CPU to interrupt upon encountering ESC instructions. At RESET, the relocation register is set to 20FFH. This causes the control block to start at FF00H in I/O space. An offset map of the 256-byte control register block is shown in Figure 10.

The integrated 80188 peripherals operate semi-autonomously from the CPU. Access to them for the most part is via software read/write of the control and data locations in the control block. Most of these registers can be both read and written. A few dedicated lines, such as interrupts and DMA request provide real-time communication between the CPU and peripherals as in a more conventional system utilizing discrete peripheral blocks. The overall interaction and function of the peripheral blocks has not substantially changed. The data access from/to the 256-byte internal control block will always be 16-bit and done in one bus cycle. Externally the BIU will still run two bus cycles for each 16-bit operation.

CHIP-SELECT/READY GENERATION LOGIC

The 80188 contains logic which provides programmable chip-select generation for both memories and peripherals. In addition, it can be programmed to provide READY (or WAIT state) generation. It can also provide latched address bits A1 and A2. The chip-select lines are active for all memory and I/O cycles in their programmed areas, whether they be generated by the CPU or by the integrated DMA unit.

Memory Chip Selects

The 80188 provides 6 memory chip select outputs for 3 address areas: upper memory, lower memory, and midrange memory. One each is provided for upper memory and lower memory, while four are provided for midrange memory.

The range for each chip select is user-programmable and can be set to 2K, 4K, 8K, 16K, 32K, 64K, 128K (plus 1K and 256K for upper and lower chip selects). In addition, the beginning or base address

OFFSET: FEH

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ET	RMX	X	M/IO	Relocation Address Bits R19–R8											

ET = ESC Trap / No ESC Trap (1/0)
M/IO = Register block located in Memory / I/O Space (1/0)
RMX = Master Interrupt Controller mode /iRMX compatible Interrupt Controller mode (0/1)

Figure 9. Relocation Register

Register	OFFSET
Relocation Register	FEH
DMA Descriptors Channel 1	DAH–D0H
DMA Descriptors Channel 0	CAH–C0H
Chip-Select Control Registers	A8H–A0H
Timer 2 Control Registers	66H–60H
Timer 1 Control Registers	5EH–58H
Timer 0 Control Registers	56H–50H
Interrupt Controller Registers	3EH–20H

Figure 10. Internal Register Map

of the midrange memory chip select may also be selected. Only one chip select may be programmed to be active for any memory location at a time. All chip select sizes are in bytes.

Upper Memory $\overline{CS}$

The 80188 provides a chip select, called $\overline{UCS}$, for the top of memory. The top of memory is usually used as the system memory because after reset the 80188 begins executing at memory location FFFF0H.

The upper limit of memory defined by this chip select is always FFFFFH, while the lower limit is programmable. By programming the lower limit, the size of the select block is also defined. Table 7 shows the relationship between the base address selected and the size of the memory block obtained.

Table 7. UMCS Programming Values

Starting Address (Base Address)	Memory Block Size	UMCS Value (Assuming R0=R1=R2=0)
FFC00	1K	FFF8H
FF800	2K	FFB8H
FF000	4K	FF38H
FE000	8K	FE38H
FC000	16K	FC38H
F8000	32K	F838H
F0000	64K	F038H
E0000	128K	E038H
C0000	256K	C038H

The lower limit of this memory block is defined in the UMCS register (see Figure 11). This register is at offset A0H in the internal control block. The legal values for bits 6–13 and the resulting starting address and memory block sizes are given in Tabie 7. Any combination of bits 6–13 not shown in Table 7 will result in undefined operation. After reset, the UMCS register is programmed for a 1K area. It must be reprogrammed if a larger upper memory area is desired.

Any internally generated 20-bit address whose upper 16 bits are greater than or equal to UMCS (with bits 0–5 "0") will cause UCS to be activated. UMCS bits R2–R0 are used to specify READY mode for the area of memory defined by this chip-select register, as explained below.

Lower Memory $\overline{CS}$

The 80188 provides a chip select for low memory called $\overline{LCS}$. The bottom of memory contains the interrupt vector table, starting at location 00000H.

The lower limit of memory defined by this chip select is always 0H, while the upper limit is programmable. By programming the upper limit, the size of the memory block is also defined. Table 8 shows the relationship between the upper address selected and the size of the memory block obtained.

Table 8. LMCS Programming Values

Upper Address	Memory Block Size	LMCS Value (Assuming R0 = R1 = R2 = 0)
003FFFH	1K	0038H
007FFFH	2K	0078H
00FFFFH	4K	00F8H
01FFFFH	8K	01F8H
03FFFFH	16K	03F8H
07FFFFH	32K	07F8H
0FFFFFH	64K	0FF8H
1FFFFFH	128K	1FF8H
3FFFFFH	256K	3FF8H

The upper limit of this memory block is defined in the LMCS register (see Figure 12). This register is at offset A2H in the internal control block. The legal values for bits 6–15 and the resulting upper address and memory block sizes are given in Table 8. Any combination of bits 6–15 not shown in Table 8 will result in undefined operation. After reset, the LMCS register value is undefined. However, the $\overline{LCS}$ chip-select line will not become active until the LMCS register is accessed.

Any internally generated 20-bit address whose upper 16 bits are less than or equal to LMCS (with bits 0–5 "1") will cause $\overline{LCS}$ to be active. LMCS register bits R2–R0 are used to specify the READY mode for the area of memory defined by this chip-select register.

Mid-Range Memory $\overline{CS}$

The 80188 provides four $\overline{MCS}$ lines which are active within a user-locatable memory block. This block can be located anywhere within the 80188 1M byte memory address space exclusive of the areas defined by $\overline{UCS}$ and $\overline{LCS}$. Both the base address and size of this memory block are programmable.

The size of the memory block defined by the mid-range select lines, as shown in Table 9, is determined by bits 8–14 of the MPCS register (see Figure 13). This register is at location A8H in the internal control block. One and only one of bits 8–14 must be set at a time. Unpredictable operation of the $\overline{MCS}$ lines will otherwise occur. Each of the four chip-select lines is active for one of the four equal contiguous divisions of the mid-range block. Thus, if the total block size is 32K, each chip select is active for 8K of memory with $\overline{MCS0}$ being active for the first range and $\overline{MCS3}$ being active for the last range.

The EX and MS in MPCS relate to peripheral functionality as described in a later section.

Table 9. MPCS Programming Values

Total Block Size	Individual Select Size	MPCS Bits 14–8
8K	2K	0000001B
16K	4K	0000010B
32K	8K	0000100B
64K	16K	0001000B
128K	32K	0010000B
256K	64K	0100000B
512K	128K	1000000B

The base address of the mid-range memory block is defined by bits 15–9 of the MMCS register (see Figure 14). This register is at offset A6H in the internal control block. These bits correspond to bits A19–A13 of the 20-bit memory address. Bits A12–A0 of the base address are always 0. The base address may be set at any integer multiple of the size of the total memory block selected. For example, if the mid-range block size is 32K (or the size of the block for which each $\overline{MCS}$ line is active is 8K), the block could be located at 10000H or 18000H, but not at 14000H, since the first few integer multiples of a 32K memory block are 0H, 8000H, 10000H, 18000H, etc. After reset, the contents of both of these registers is undefined. However, none of the $\overline{MCS}$ lines will be active until both the MMCS and MPCS registers are accessed.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A0H	1	1	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0
	A19								A11							

Figure 11. UMCS Register

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A2H	0	0	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0
	A19								A11							

Figure 12. LMCS Register

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A8H	1	M6	M5	M4	M3	M2	M1	M0	EX	MS	1	1	1	R2	R1	R0

Figure 13. MPCS Register

	15						9						3			0
OFFSET: A6H	U	U	U	U	U	U	U	1	1	1	1	1	1	R2	R1	R0
	A19						A13									

Figure 14. MMCS Register

MMCS bits R2–R0 specify READY mode of operation for all mid-range chip selects. All devices in mid-range memory must use the same number of WAIT states.

The 512K block size for the mid-range memory chip selects is a special case. When using 512K, the base address would have to be at either locations 00000H or 80000H. If it were to be programmed at 00000H when the $\overline{LCS}$ line was programmed, there would be an internal conflict between the $\overline{LCS}$ ready generation logic and the $\overline{MCS}$ ready generation logic. Likewise, if the base address were programmed at 80000H, there would be a conflict with the $\overline{UCS}$ ready generation logic. Since the $\overline{LCS}$ chip-select line does not become active until programmed, while the $\overline{UCS}$ line is active at reset, the memory base can be set only at 00000H. If this base address is selected, however, the $\overline{LCS}$ range must not be programmed.

Peripheral Chip Selects

The 80188 can generate chip selects for up to seven peripheral devices. These chip selects are active for seven contiguous blocks of 128 bytes above a programmable base address. This base address may be located in either memory or I/O space.

Seven $\overline{CS}$ lines called $\overline{PCS0}$–6 are generated by the 80188. The base address is user-programmable; however it can only be a multiple of 1K bytes, i.e., the least significant 10 bits of the starting address are always 0.

$\overline{PCS5}$ and $\overline{PCS6}$ can also be programmed to provide latched address bits A1, A2. If so programmed, they cannot be used as peripheral selects. These outputs can be connected directly to the A0, A1 pins used for selecting internal registers of 8-bit peripheral chips. This scheme simplifies the hardware interface because the 8-bit registers of peripherals are simply treated as 16-bit registers located on even boundaries in I/O space or memory space where only the lower 8-bits of the register are significant: the upper 8-bits are "don't cares."

The starting address of the peripheral chip-select block is defined by the PACS register (see Figure 15). This register is located at offset A4H in the internal control block. Bits 15–6 of this register correspond to bits 19–10 of the 20-bit Programmable Base Address (PBA) of the peripheral chip-select block. Bits 9–0 of the PBA of the peripheral chip-select block are all zeros. If the chip-select block is located in I/O space, bits 12–15 must be programmed zero, since the I/O address is only 16 bits wide. Table 10 shows the address range of each peripheral chip select with respect to the PBA contained in PACS register.

	15									6	5		3			0
OFFSET: A4H	U	U	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0
	A19									A10						

Figure 15. PACS Register

The user should program bits 15–6 to correspond to the desired peripheral base location. PACS bits 0–2 are used to specify READY mode for PSC0–PCS3.

Table 10. PCS Address Ranges

PCS Line	Active between Locations
PCS0	PBA —PBA+127
PCS1	PBA+128—PBA+255
PCS2	PBA+256—PBA+383
PCS3	PBA+384—PBA+511
PCS4	PBA+512—PBA+639
PCS5	PBA+640—PBA+767
PCS6	PBA+768—PBA+895

The mode of operation of the peripheral chip selects is defined by the MPCS register (which is also used to set the size of the mid-range memory chip-select block, see Figure 16). This register is located at offset A8H in the internal control block. Bit 7 is used to select the function of PCS5 and PCS6, while bit 6 is used to select whether the peripheral chip selects are mapped into memory or I/O space. Table 11 describes the programming of these bits. After reset, the contents of both the MPCS and the PACS registers are undefined, however none of the PCS lines will be active until both of the MPCS and PACS registers are accessed.

Table 11. MS, EX Programming Values

Bit	Description
MS	1 = Peripherals mapped into memory space. 0 = Peripherals mapped into I/O space.
EX	0 = 5 PCS lines. A1, A2 provided. 1 = 7 PCS lines. A1, A2 are not provided.

MPCS bits 0–2 are used to specify READY mode for PCS4–PCS6 as outlined below.

READY Generation Logic

The 80188 can generate a "READY" signal internally for each of the memory or peripheral CS lines. The number of WAIT states to be inserted for each peripheral or memory is programmable to provide 0–3 wait states for all accesses to the area for which the chip select is active. In addition, the 80188 may be programmed to either ignore external READY for each chip-select range individually or to factor external READY with the integrated ready generator.

READY control consists of 3 bits for each CS line or group of lines generated by the 80188. The interpretation of the ready bits is shown in Table 12.

Table 12. READY Bits Programming

R2	R1	R0	Number of WAIT States Generated
0	0	0	0 wait states, external RDY also used.
0	0	1	1 wait state inserted, external RDY also used.
0	1	0	2 wait states inserted, external RDY also used.
0	1	1	3 wait states inserted, external RDY also used.
1	0	0	0 wait states, external RDY ignored.
1	0	1	1 wait state inserted, external RDY ignored.
1	1	0	2 wait states inserted, external RDY ignored.
1	1	1	3 wait states inserted, external RDY ignored.

The internal ready generator operates in parallel with external READY, not in series if the external READY is used (R2 = 0). This means, for example, if the internal generator is set to insert two wait states, but activity on the external READY lines will insert four wait states, the processor will only insert four wait states, not six. This is because the two wait states generated by the internal generator overlapped the first two wait states generated by the external ready signal. Note that the external ARDY and SRDY lines are always ignored during cycles accessing internal peripherals.

R2–R0 of each control word specifies the READY mode for the corresponding block, with the exception of the peripheral chip selects: R2–R0 of PACS set the PCS0–3 READY mode, R2–R0 of MPCS set the PCS4–6 READY mode.

Chip Select/Ready Logic and Reset

Upon reset, the Chip-Select/Ready Logic will perform the following actions:

- All chip-select outputs will be driven HIGH.
- Upon leaving RESET, the UCS line will be programmed to provide chip selects to a 1K block with the accompanying READY control bits set at 011 to allow the maximum number of internal wait states in conjunction with external Ready consideration (i.e., UMCS resets to FFFBH).
- No other chip select or READY control registers have any predefined values after RESET. They will not become active until the CPU accesses their control registers. Both the PACS and MPCS registers must be accessed before the PCS lines will become active.

OFFSET: A8H	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	M6	M5	M4	M3	M2	M1	M0	EX	MS	1	1	1	R2	R1	R0

Figure 16. MPCS Register

DMA Channels

The 80188 DMA controller provides two independent high-speed DMA channels. Data transfers can occur between memory and I/O spaces (e.g., Memory to I/O) or within the same space (e.g., Memory to Memory or I/O to I/O). Each DMA channel maintains both a 20-bit source and destination pointer which can be optionally incremented or decremented after each data transfer. Each data transfer consumes 2 bus cycles (a minimum of 8 clocks), one cycle to fetch data and the other to store data. This provides a data transfer rate of one MByte/sec at 8 MHz.

DMA Operation

Each channel has six registers in the control block which define each channel's specific operation. The control registers consist of a 20-bit Source pointer (2 words), a 20-bit Destination pointer (2 words), a 16-bit Transfer Counter, and a 16-bit Control Word. The format of the DMA Control Blocks is shown in Table 13. The Transfer Count Register (TC) specifies the number of DMA transfers to be performed. Up to 64K byte transfers can be performed with automatic termination. The Control Word defines the channel's operation (see Figure 18). All registers may be modified or altered during any DMA activity. Any changes made to these registers will be reflected immediately in DMA operation.

Table 13. DMA Control Block Format

Register Name	Register Address	
	Ch. 0	Ch. 1
Control Word	CAH	DAH
Transfer Count	C8H	D8H
Destination Pointer (upper 4 bits)	C6H	D6H
Destination Pointer	C4H	D4H
Source Pointer (upper 4 bits)	C2H	D2H
Source Pointer	C0H	D0H

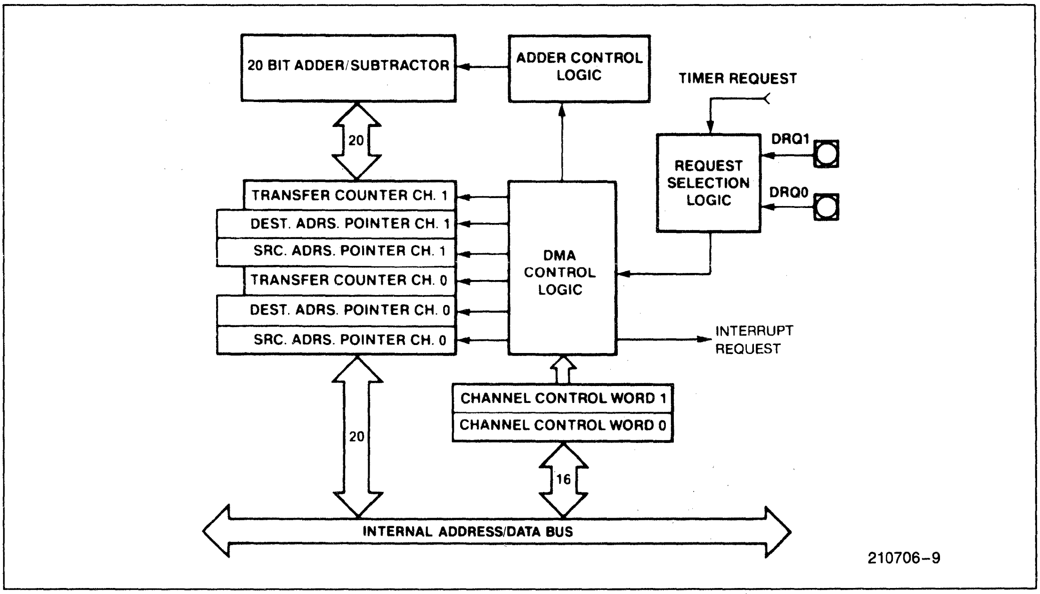


Figure 17. DMA Unit Block Diagram

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M/$\overline{IO}$	DESTINATION DEC	INC	M/$\overline{IO}$	SOURCE DEC	INC	TC	INT	SYN		P	TDRQ	X	CHG/$\overline{NOCHG}$	ST/$\overline{STOP}$	X

X = Don't Care

Figure 18. DMA Control Register

DMA Channel Control Word Register

Each DMA Channel Control Word determines the mode of operation for the particular 80188 DMA channel. This register specifies:

- the mode of synchronization;
- whether interrupts will be generated after the last transfer;
- whether DMA activity will cease after a programmed number of DMA cycles;
- the relative priority of the DMA channel with respect to the other DMA channel;
- whether the source pointer will be incremented, decremented, or maintained constant after each transfer;
- whether the source pointer addresses memory or I/O space;
- whether the destination pointer will be incremented, decremented, or maintained constant after each transfer; and
- whether the destination pointer will address memory or I/O space.

The DMA channel control registers may be changed while the channel is operating. However, any changes made during operation will affect the current DMA transfer.

DMA Control Word Bit Descriptions

ST/$\overline{STOP}$: Start/stop (1/0) Channel.

CHG/$\overline{NOCHG}$: Change/Do not change (1/0) ST/$\overline{STOP}$ bit. If this bit is set when writing to the control word, the ST/$\overline{STOP}$ bit will be programmed by the write to the control word. If this bit is cleared when writing the control word, the ST/$\overline{STOP}$ bit will not be altered. This bit is not stored; it will always be a 0 on read.

INT: Enable Interrupts to CPU on byte count termination.

TC: If set, DMA will terminate when the contents of the Transfer Count register reaches zero. The ST/$\overline{STOP}$ bit will also be reset at this point if TC is set. If this bit is cleared, the DMA unit will decrement the transfer count register for each DMA cycle, but the DMA transfer will not stop when the contents of the TC register reaches zero.

SYN: (2 bits) 00 No synchronization.

NOTE:

The ST bit will be cleared automatically when the contents of the TC register reaches zero regardless of the state of the TC bit.

01 Source synchronization.

10 Destination synchronization.

11 Unused.

SOURCE:INC Increment source pointer by 1 after each transfer.

M/$\overline{IO}$ Source pointer is in M/$\overline{IO}$ space (1/0).

DEC Decrement source pointer by 1 after each transfer.

DEST: INC Increment destination pointer by 1 after each transfer.

M/$\overline{IO}$ Destination pointer is in M/$\overline{IO}$ space (1/0).

DEC Decrement destination pointer by 1 after each transfer.

P Channel priority—relative to other channel.

0 low priority.

1 high priority.

Channels will alternate cycles if both set at same priority level.

TDRQ 0: Disable DMA requests from timer 2.

1: Enable DMA requests from timer 2.

Bit 3 Bit 3 is not used.

If both INC and DEC are specified for the same pointer, the pointer will remain constant after each cycle.

DMA Destination and Source Pointer Registers

Each DMA channel maintains a 20-bit source and a 20-bit destination pointer. Each of these pointers takes up two full 16-bit registers in the peripheral control block. The lower four bits of the upper register contain the upper four bits of the 20-bit physical address (see Figure 18a). These pointers may be individually incremented or decremented after each transfer. Each pointer may point into either memory or I/O space. Since the DMA channels can perform transfers to or from odd addresses, there is no restriction on values for the pointer registers.

DMA Transfer Count Register

Each DMA channel maintains a 16-bit transfer count register (TC). This register is decremented after every DMA cycle, regardless of the state of the TC bit in the DMA Control Register. If the TC bit in the DMA control word is set or unsynchronized transfers are programmed, DMA activity will terminate when the transfer count register reaches zero.

DMA Requests

Data transfers may be either source or destination synchronized, that is either the source of the data or the destination of the data may request the data transfer. In addition, DMA transfers may be unsynchronized; that is, the transfer will take place continually until the correct number of transfers has occurred. When source or unsynchronized transfers are performed, the DMA channel may begin another transfer immediately after the end of a previous DMA transfer. This allows a complete transfer to take place every 2 bus cycles or eight clock cycles (assuming no wait states). No prefetching occurs when destination synchronization is performed, however. Data will not be fetched from the source address until the destination device signals that it is ready to receive it. When destination synchronized transfers are requested, the DMA controller will relinquish control of the bus after every transfer. If no other bus activity is initiated, another DMA cycle will begin after two processor clocks. This is done to allow the destination device time to remove its request if another transfer is not desired. Since the DMA controller will relinquish the bus, the CPU can initiate a bus cycle. As a result, a complete bus cycle will often be inserted between destination synchronized transfers. These lead to the maximum DMA transfer rates shown in Table 14.

Table 14. Maximum DMA Transfer Rates

Type of Synchronization Selected	CPU Running	CPU Halted
Unsynchronized	1 MBytes/sec	1 MBytes/sec
Source Synch	1 MBytes/sec	1 MBytes/sec
Destination Synch	0.65 MBytes/sec	0.75 MBytes/sec

	15			0
HIGHER REGISTER ADDRESS	XXX	XXX	XXX	A19–A16
LOWER REGISTER ADDRESS	A15–A12	A11–A8	A7–A4	A3–A0

XXX = Don't Care

Figure 18a. DMA Memory Pointer Register Format

DMA Acknowledge

No explicit DMA acknowledge pulse is provided. Since both source and destination pointers are maintained, a read from a requesting source, or a write to a requesting destination, should be used as the DMA acknowledge signal. Since the chip-select lines can be programmed to be active for a given block of memory or I/O space, and the DMA pointers can be programmed to point to the same given block, a chip-select line could be used to indicate a DMA acknowledge.

DMA Priority

The DMA channels may be programmed such that one channel is always given priority over the other, or they may be programmed such as to alternate cycles when both have DMA requests pending. DMA cycles always have priority over internal CPU cycles except between locked memory accesses or word accesses the odd memory locations; however, an external bus hold takes priority over an internal DMA cycle. Because an interrupt request cannot suspend a DMA operation and the CPU cannot access memory during a DMA cycle, interrupt latency time will suffer during sequences of continuous DMA cycles. An NMI request, however, will cause all internal DMA activity to halt. This allows the CPU to quickly respond to the NMI request.

DMA Programming

DMA cycles will occur whenever the ST/STOP bit of the Control Register is set. If synchronized transfers are programmed, a DRQ must also have been generated. Therefore, the source and destination transfer pointers, and the transfer count register (if used) must be programmed before this bit is set.

Each DMA register may be modified while the channel is operating. If the CHG/NOCHG bit is cleared when the control register is written, the ST/STOP bit of the control register will not be modified by the write. If multiple channel registers are modified, it is recommended that a LOCKED string transfer be used to prevent a DMA transfer from occurring between updates to the channel registers.

DMA Channels and Reset

Upon RESET, the DMA channels will perform the following actions:

- The Start/Stop bit for each channel will be reset to STOP.
- Any transfer in progress is aborted.

TIMERS

The 80188 provides three internal 16-bit programmable timers (see Figure 19). Two of these are highly flexible and are connected to four external pins (2 per timer). They can be used to count external events, time external events, generate nonrepetitive waveforms, etc. The third timer is not connected to any external pins, and is useful for real-time coding and time delay applications. In addition, this third timer can be used as a prescaler to the other two, or as a DMA request source.

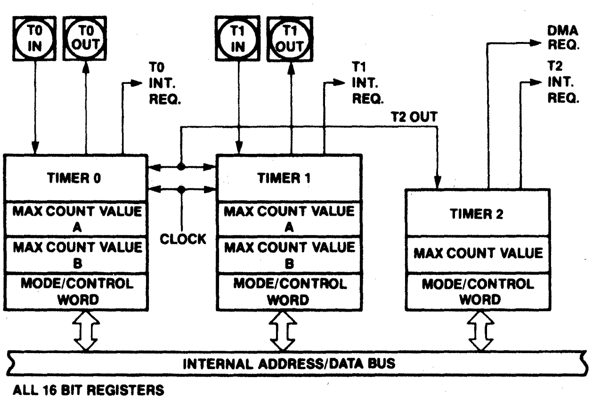


Figure 19. Timer Block Diagram

Timer Operation

The timers are controlled by 11 16-bit registers in the internal peripheral control block. The configuration of these registers is shown in Table 15. The count register contains the current value of the timer. It can be read or written at any time independent of whether the timer is running or not. The value of this register will be incremented for each timer event. Each of the timers is equipped with a MAX COUNT register, which defines the maximum count the timer will reach. After reaching the MAX COUNT register value, the timer count value will reset to zero during that same clock, i.e., the maximum count value is never stored in the count register itself. Timers 0 and 1 are, in addition, equipped with a second MAX COUNT register, which enables the timers to alternate their count between two different MAX COUNT values programmed by the user. If a single MAX COUNT register is used, the timer output pin will switch LOW for a single clock, 2 clocks after the maximum count value has been reached. In the dual MAX COUNT register mode, the output pin will indicate which MAX COUNT register is currently in use, thus allowing nearly complete freedom in selecting waveform duty cycles. For the timers with two MAX COUNT registers, the RIU bit in the control register determines which is used for the comparison.

Each timer gets serviced every fourth CPU-clock cycle, and thus can operate at speeds up to one-quarter the internal clock frequency (one-eighth the crystal rate). External clocking of the timers may be done at up to a rate of one-quarter of the internal CPU-clock rate (2 MHz for an 8 MHz CPU clock). Due to internal synchronization and pipelining of the timer circuitry, a timer output may take up to 6 clocks to respond to any individual clock or gate input.

Since the count registers and the maximum count registers are all 16 bits wide, 16 bits of resolution are provided. Any Read or Write access to the timers will add one wait state to the minimum four-clock bus cycle, however. This is needed to synchronize and coordinate the internal data flows between the internal timers and the internal bus.

The timers have several programmable options.

- All three timers can be set to halt or continue on a terminal count.
- Timers 0 and 1 can select between internal and external clocks, alternate between MAX COUNT registers and be set to retrigger on external events.
- The timers may be programmed to cause an interrupt on terminal count.

These options are selectable via the timer mode/control word.

Timer Mode/Control Register

The mode/control register (see Figure 20) allows the user to program the specific mode of operation or check the current programmed status for any of the three integrated timers.

Table 15. Timer Control Block Format

Register Name	Register Offset		
	Tmr. 0	**Tmr. 1**	**Tmr. 2**
Mode/Control Word	56H	5EH	66H
Max Count B	54H	5CH	not present
Max Count A	52H	5AH	62H
Count Register	50H	58H	60H

15	14	13	12	11		5	4	3	2	1	0
EN	$\overline{\text{INH}}$	INT	RIU	0	• • •	MC	RTG	P	EXT	ALT	CONT

Figure 20. Timer Mode/Control Register

ALT

The ALT bit determines which of two MAX COUNT registers is used for count comparison. If ALT = 0, register A for that timer is always used, while if ALT = 1, the comparison will alternate between register A and register B when each maximum count is reached. This alternation allows the user to change one MAX COUNT register while the other is being used, and thus provides a method of generating non-repetitive waveforms. Square waves and pulse outputs of any duty cycle are a subset of available signals obtained by not changing the final count registers. The ALT bit also determines the function of the timer output pin. If ALT is zero, the output pin will go LOW for one clock, the clock after the maximum count is reached. If ALT is one, the output pin will reflect the current MAX COUNT register being used (0/1 for B/A).

CONT

Setting the CONT bit causes the associated timer to run continuously, while resetting it causes the timer to halt upon maximum count. If CONT = 0 and ALT = 1, the timer will count to the MAX COUNT register A value, reset, count to the register B value, reset, and halt.

EXT

The external bit selects between internal and external clocking for the timer. The external signal may be asynchronous with respect to the 80188 clock. If this bit is set, the timer will count LOW-to-HIGH transitions on the input pin. If cleared, it will count an internal clock while using the input pin for control. In this mode, the function of the external pin is defined by the RTG bit. The maximum input to output transition latency time may be as much as 6 clocks. However, clock inputs may be pipelined as closely together as every 4 clocks without losing clock pulses.

P

The prescaler bit is ignored unless internal clocking has been selected (EXT = 0). If the P bit is a zero, the timer will count at one-fourth the internal CPU clock rate. If the P bit is a one, the output of timer 2 will be used as a clock for the timer. Note that the user must initialize and start timer 2 to obtain the prescaled clock.

RTG

Retrigger bit is only active for internal clocking (EXT = 0). In this case it determines the control function provided by the input pin.

If RTG = 0, the input level gates the internal clock on and off. If the input pin is HIGH, the timer will count; if the input pin is LOW, the timer will hold its value. As indicated previously, the input signal may be asynchronous with respect to the 80188 clock.

When RTG = 1, the input pin detects LOW-to-HIGH transitions. The first such transition starts the timer running, clearing the timer value to zero on the first clock, and then incrementing thereafter. Further transitions on the input pin will again reset the timer to zero, from which it will start counting up again. If CONT = 0, when the timer has reached maximum count, the EN bit will be cleared, inhibiting further timer activity.

EN

The enable bit provides programmer control over the timer's RUN/HALT status. When set, the timer is enabled to increment subject to the input pin constraints in the internal clock mode (discussed previously). When cleared, the timer will be inhibited from counting. All input pin transitions during the time EN is zero will be ignored. If CONT is zero, the EN bit is automatically cleared upon maximum count.

INH

The inhibit bit allows for selective updating of the enable (EN) bit. If INH is a one during the write to the mode/control word, then the state of the EN bit will be modified by the write. If INH is a zero during the write, the EN bit will be unaffected by the operation. This bit is not stored; it will always be a 0 on a read.

INT

When set, the INT bit enables interrupts from the timer, which will be generated on every terminal count. If the timer is configured in dual MAX COUNT register mode, an interrupt will be generated each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. If this enable bit is cleared after the interrupt request has been generated, but before a pending interrupt is serviced, the interrupt request will still be in force. (The request is latched in the Interrupt Controller.)

MC

The Maximum Count bit is set whenever the timer reaches its final maximum count value. If the timer is configured in dual MAX COUNT register mode, this bit will be set each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. This bit is set regardless of the timer's interrupt-enable bit. The MC bit gives the user the ability to monitor timer status through software instead of through interrupts. Programmer intervention is required to clear this bit.

RIU

The Register In Use bit indicates which MAX COUNT register is currently being used for comparison to the timer count value. A zero value indicates register A. The RIU bit cannot be written, i.e., its value is not affected when the control register is written. It is always cleared when the ALT bit is zero.

Not all mode bits are provided for timer 2. Certain bits are hardwired as indicated below:

ALT = 0, EXT = 0, P = 0, RTG = 0, RIU = 0

Count Registers

Each of the three timers has a 16-bit count register. The current contents of this register may be read or written by the processor at any time. If the register is written into while the timer is counting, the new value will take effect in the current count cycle.

Max Count Registers

Timers 0 and 1 have two MAX COUNT registers, while timer 2 has a single MAX COUNT register. These contain the number of events the timer will count. In timers 0 and 1, the MAX COUNT register used can alternate between the two max count values whenever the current maximum count is reached. The condition which causes a timer to reset is equivalent between the current count value and the max count being used. This means that if the count is changed to be above the max count value, or if the max count value is changed to be below the current value, the timer will not reset to zero, but rather will count to its maximum value, "wrap around" to zero, then count until the max count is reached.

Timers and Reset

Upon RESET, the Timers will perform the following actions:

- All EN (Enable) bits are reset preventing timer counting.
- All SEL (Select) bits are reset to zero. This selects MAX COUNT register A, resulting in the Timer Out pins going HIGH upon RESET.

INTERRUPT CONTROLLER

The 80188 can receive interrupts from a number of sources, both internal and external. The internal interrupt controller serves to merge these requests on a priority basis, for individual service by the CPU.

Internal interrupt sources (Timers and DMA channels) can be disabled by their own control registers or by mask bits within the interrupt controller. The 80188 interrupt controller has its own control register that set the mode of operation for the controller.

The interrupt controller will resolve priority among requests that are pending simultaneously. Nesting is provided so interrupt service routines for lower priority interrupts may themselves be interrupted by higher priority interrupts. A block diagram of the interrupt controller is shown in Figure 21.

The interrupt controller has a special iRMX 86 compatibility mode that allows the use of the 80188 within the iRMX 86 operating system interrupt structure. The controller is set in this mode by setting bit 14 in the peripheral control block relocation register (see iRMX 86 Compatibility Mode section). In this mode, the internal 80188 interrupt controller functions as a "slave" controller to an external "master" controller. Special initialization software must be included to properly set up the 80188 interrupt controller in iRMX 86 mode.

NON-iRMX™ MODE OPERATION

Interrupt Controller External Interface

For external interrupt sources, five dedicated pins are provided. One of these pins is dedicated to NMI, non-maskable interrupt. This is typically used for power-fail interrupts, etc. The other four pins may function either as four interrupt input lines with internally generated interrupt vectors, as an interrupt line and an interrupt acknowledge line (called the "cascade mode") along with two other input lines with internally generated interrupt vectors, or as two interrupt input lines and two dedicated interrupt acknowledge output lines. When the interrupt lines are configured in cascade mode, the 80188 interrupt controller will not generate internal interrupt vectors.

External sources in the cascade mode use externally generated interrupt vectors. When an interrupt is acknowledged, two $\overline{\text{INTA}}$ cycles are initiated and the vector is read into the 80188 on the second cycle. The capability to interface to external 8259A programmable interrupt controllers is thus provided when the inputs are configured in cascade mode.

Interrupt Controller Modes of Operation

The basic modes of operation of the interrupt controller in non-iRMX™ mode are similar to the 8259A. The interrupt controller responds indentically to internal interrupts in all three modes: the difference is only in the interpretation of function of the four external interrupt pins. The interrupt controller is set into one of these three modes by programming the correct bits in the INT0 and INT1 control registers. The modes of interrupt controller operation are as follows:

FULLY NESTED MODE

When in the fully nested mode four pins are used as direct interrupt requests. The vectors for these four inputs are generated internally. An in-service bit is provided for every interrupt source. If a lower-priority device requests an interrupt while the in-service bit (IS) is set, no interrupt will be generated by the interrupt controller. In addition, if another interrupt request occurs from the same interrupt source while the in-service bit is set, no interrupt will be generated by the interrupt controller. This allows interrupt service routines to operate with interrupts enabled without being themselves interrupted by lower-priority interrupts. Since interrupts are enabled, higher-priority interrupts will be serviced.

When a service routine is completed, the proper IS bit must be reset by writing the proper pattern to the EOI register. This is required to allow subsequent interrupts from this interrupt source and to allow servicing of lower-priority interrupts. An EOI command is issued at the end of the service routine just before the issuance of the return from interrupt instruction. If the fully nested structure has been upheld, the next highest-priority source with its IS bit set is then serviced.

CASCADE MODE

The 80188 has four interrupt pins and two of them have dual functions. In the fully nested mode the four pins are used as direct interrupt inputs and the corresponding vectors are generated internally. In the cascade mode, the four pins are configured into interrupt input-dedicated acknowledge signal pairs. The interconnection is shown in Figure 22. INT0 is an interrupt input interfaced to an 8259A, while INT2/$\overline{\text{INTA0}}$ serves as the dedicated interrupt acknowledge signal to that peripheral. The same is true for INT1 and INT3/$\overline{\text{INTA1}}$. Each pair can selectively be placed in the cascade or non-cascade mode by programming the proper value into INT0 and INT1 control registers. The use of the dedicated acknowledge signals eliminates the need for the use of external logic to generate $\overline{\text{INTA}}$ and device select signals.

The primary cascade mode allows the capability to serve up to 128 external interrupt sources through the use of external master and slave 8259As. Three levels of priority are created, requiring priority resolution in the 80188 interrupt controller, the master 8259As, and the slave 8259As. If an external interrupt is serviced, one IS bit is set at each of these levels. When the interrupt service routine is completed, up to three end-of-interrupt commands must be issued by the programmer.

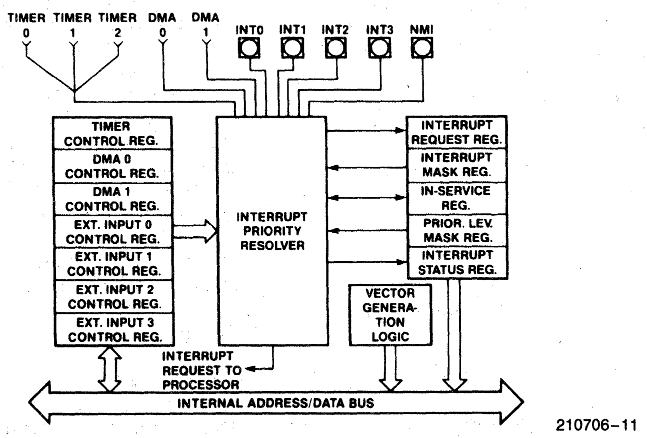


Figure 21. Interrupt Controller Block Diagram

SPECIAL FULLY NESTED MODE

This mode is entered by setting the SFNM bit in INT0 or INT1 control register. It enables complete nestability with external 8259A masters. Normally, an interrupt request from an interrupt source will not be recognized unless the in-service bit for that source is reset. If more than one interrupt source is connected to an external interrupt controller, all of the interrupts will be funneled through the same 80188 interrupt request pin. As a result, if the external interrupt controller receives a higher-priority interrupt, its interrupt will not be recognized by the 80188 controller until the 80188 in-service bit is reset. In special fully nested mode, the 80188 interrupt controller will allow interrupts from an external pin regardless of the state of the in-service bit for an interrupt source in order to allow multiple interrupts from a single pin. An in-service bit will continue to be set, however, to inhibit interrupts from other lower-priority 80188 interrupt sources.

Special procedures should be followed when resetting IS bits at the end of interrupt service routines. Software polling of the external master's IS register is required to determine if there is more than one bit set. If so, the IS bit in the 80188 remains active and the next interrupt service routine is entered.

Operation in a Polled Environment

The controller may be used in a polled mode if interrupts are undesirable. When polling, the processor disables interrupts and then polls the interrupt controller whenever it is convenient. Polling the interrupt controller is accomplished by reading the Poll Word (Figure 31). Bit 15 in the poll word indicates to the processor that an interrupt of high enough priority is requesting service. Bits 0–4 indicate to the processor the type vector of the highest-priority source requesting service. Reading the Poll Word causes the In-Service bit of the highest priority source to be set.

It is desirable to be able to read the Poll Word information without guaranteeing service of any pending interrupt, i.e., not set the indicated in-service bit. The 80188 provides a Poll Status Word in addition to the conventional Poll Word to allow this to be done. Poll Word information is duplicated in the Poll Status Word, but reading the Poll Status Word does not set the associated in-service bit. These words are located in two adjacent memory locations in the register file.

Non-iRMX™ Mode Features

PROGRAMMABLE PRIORITY

The user can program the interrupt sources into any of eight different priority levels. The programming is done by placing a 3-bit priority level (0–7) in the control register of each interrupt source. (A source with a priority level of 4 has higher priority over all priority levels from 5 to 7. Priority registers containing values lower than 4 have greater priority). All interrupt sources have preprogrammed default priority levels (see Table 4).

If two requests with the same programmed priority level are pending at once, the priority ordering scheme shown in Table 4 is used. If the serviced interrupt routine reenables interrupts, it allows other requests to be serviced.

END-OF-INTERRUPT COMMAND

The end-of-interrupt (EOI) command is used by the programmer to reset the In-Service (IS) bit when an interrupt service routine is completed. The EOI command is issued by writing the proper pattern to the EOI register. There are two types of EOI commands, specific and nonspecific. The nonspecific command does not specify which IS bit is reset. When issued, the interrupt controller automatically resets the IS bit of the highest priority source with an active service routine. A specific EOI command requires that the programmer send the interrupt vector type to the interrupt controller indicating which source's IS bit is to be reset. This command is used when the fully nested structure has been disturbed or the highest priority IS bit that was set does not belong to the service routine in progress.

TRIGGER MODE

The four external interrupt pins can be programmed in either edge- or level-trigger mode. The control register for each external source has a level-trigger mode (LTM) bit. All interrupt inputs are active HIGH. In the edge sense mode or the level-trigger mode, the interrupt request must remain active (HIGH) until the interrupt request is acknowledged by the 80188 CPU. In the edge-sense mode, if the level remains high after the interrupt is acknowledged, the input is disabled and no further requests will be generated. The input level must go LOW for at least one clock cycle to reenable the input. In the level-trigger mode, no such provision is made: holding the interrupt input HIGH will cause continuous interrupt requests.

INTERRUPT VECTORING

The 80188 Interrupt Controller will generate interrupt vectors for the integrated DMA channels and the integrated Timers. In addition, the Interrupt Controller will generate interrupt vectors for the external interrupt lines if they are not configured in Cascade or Special Fully Nested Mode. The interrupt vectors generated are fixed and cannot be changed (see Table 4).

Interrupt Controller Registers

The Interrupt Controller register model is shown in Figure 23. It contains 15 registers. All registers can both be read or written unless specified otherwise.

IN-SERVICE REGISTER

This register can be read from or written into. The format is shown in Figure 24. It contains the In-Service bit for each of the interrupt sources. The In-Service bit is set to indicate that a source's service routine is in progress. When an In-Service bit is set, the interrupt controller will not generate interrupts to the CPU when it receives interrupt requests from devices with a lower programmed priority level. The TMR bit is the In-Service bit for all three timers; the D0 and D1 bits are the In-Service bits for the two DMA channels; the I0–I3 are the In-Service bits for the external interrupt pins. The IS bit is set when the processor acknowledges an interrupt request either by an interrupt acknowledge or by reading the poll register. The IS bit is reset at the end of the interrupt service routine by an end-of-interrupt command issued by the CPU.

INTERRUPT REQUEST REGISTER

The internal interrupt sources have interrupt request bits inside the interrupt controller. The format of this register is shown in Figure 24. A read from this register yields the status of these bits. The TMR bit is the logical OR of all timer interrupt requests. D0 and D1 are the interrupt request bits for the DMA channels.

The state of the external interrupt input pins is also indicated. The state of the external interrupt pins is not a stored condition inside the interrupt controller, therefore the external interrupt bits cannot be written. The external interrupt request bits show exactly when an interrupt request is given to the interrupt controller, so if edge-triggered mode is selected, the bit in the register will be HIGH only after an inactive-to-active transition. For internal interrupt sources, the register bits are set when a request arrives and are reset when the processor acknowledges the requests.

MASK REGISTER

This is a 16-bit register that contains a mask bit for each interrupt source. The format for this register is shown in Figure 24. A one in a bit position corresponding to a particular source serves to mask the source from generating interrupts. These mask bits are the exact same bits which are used in the individual control registers; programming a mask bit using the mask register will also change this bit in the individual control registers, and vice versa.

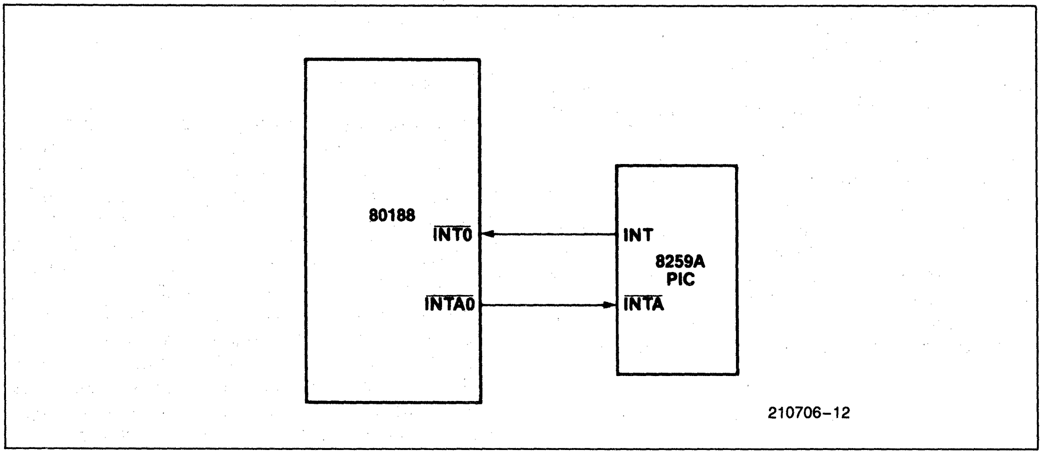


Figure 22. Cascade Mode Interrupt Connection

	OFFSET
INT3 CONTROL REGISTER	3EH
INT2 CONTROL REGISTER	3CH
INT1 CONTROL REGISTER	3AH
INT0 CONTROL REGISTER	38H
DMA 1 CONTROL REGISTER	36H
DMA 0 CONTROL REGISTER	34H
TIMER CONTROL REGISTER	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
POLL STATUS REGISTER	26H
POLL REGISTER	24H
EOI REGISTER	22H

Figure 23. Interrupt Controller Registers (Non-iRMX™ 86 Mode)

PRIORITY MASK REGISTER

This register is used to mask all interrupts below particular interrupt priority levels. The format of this register is shown in Figure 25. The code in the lower three bits of this register inhibits interrupts of priority lower (a higher priority number) than the code specified. For example, 100 written into this register masks interrupts of level five (101), six (110), and seven (111). The register is reset to seven (111) upon RESET so all interrupts are unmasked.

INTERRUPT STATUS REGISTER

This register contains general interrupt controller status information. The format of this register is shown in Figure 26. The bits in the status register have the following functions:

DHLT: DMA Halt Transfer; setting this bit halts all DMA transfers. It is automatically set whenever a non-maskable interrupt occurs, and it is reset when an IRET instruction is executed. The purpose of this bit is to allow prompt service of all non-maskable interrupts. This bit may also be set by the CPU.

IRTx: These three bits represent the individual timer interrupt request bits. These bits are used to differentiate the timer interrupts, since the timer IR bit in the interrupt request register is the "OR" function of all timer interrupt requests. Note that setting any one of these three bits initiates an interrupt request to the interrupt controller.

15	14				10	9	8	7	6	5	4	3	2	1	0
0	0	•	•	•	0	0	0	I3	I2	I1	I0	D1	D0	0	TMR

Figure 24. In-Service, Interrupt Request, and Mask Register Formats

15	14											3	2	1	0
0	0	•	•	•	•	•	•	•	•	•	•	0	PRM2	PRM1	PRM0

Figure 25. Priority Mask Register Format

15	14							7	6	5	4	3	2	1	0
DHLT	0	•	•	•	•	•	•	0	0	0	0	0	IRT2	IRT1	IRT0

Figure 26. Interrupt Status Register Format

TIMER, DMA 0, 1; CONTROL REGISTERS

These registers are the control words for all the internal interrupt sources. The format for these registers is shown in Figure 27. The three bit positions PR0, PR1, and PR2 represent the programmable priority level of the interrupt source. The MSK bit inhibits interrupt requests from the interrupt source. The MSK bits in the individual control registers are the exact same bits as are in the Mask Register; modifying them in the individual control registers will also modify them in the Mask Register, and vice versa.

INT0-INT3 CONTROL REGISTERS

These registers are the control words for the four external input pins. Figure 28 shows the format of the INT0 and INT1 Control registers; Figure 29 shows the format of the INT2 and INT3 Control registers. In cascade mode or special fully nested mode, the control words for INT2 and INT3 are not used.

The bits in the various control registers are encoded as follows:

PRO-2: Priority programming information. Highest priority = 000, lowest priority = 111.

LTM: Level-trigger mode bit. 1 = level-triggered; 0 = edge-triggered. Interrupt Input levels are active high. In level-triggered mode, an interrupt is generated whenever the external line is high. In edge-triggered mode, an interrupt will be generated only when this level is preceded by an inactive-to-active transition on the line. In both cases, the level must remain active until the interrupt is acknowledged.

MSK: Mask bit, 1 = mask; 0 = non-mask.

C: Cascade mode bit, 1 = cascade; 0 = direct.

SFNM: Special fully nested mode bit, 1 = SFNM.

EOI REGISTER

The end of the interrupt register is a command register which can only be written into. The format of this register is shown in Figure 30. It initiates an EOI command when written to by the 80188 CPU.

The bits in the EOI register are encoded as follows:

S_X: Encoded information that specifies an interrupt source vector type as shown in Table 4. For example, to reset the In-Service bit for DMA channel 0, these bits should be set to 01010, since the vector type for DMA channel 0 is 10. Note that to reset the single In-Service bit for any of the three timers, the vector type for timer 0 (8) should be written in this register.

NSPEC/SPEC: A bit that determines the type of EOI command. Nonspecific = 1, Specific = 0.

15	14										4	3	2	1	0
0	0	•	•	•	•	•	•	•	•	•	0	MSK	PR2	PR1	PR0

Figure 27. Timer/DMA Control Register Formats

15	14							7	6	5	4	3	2	1	0
0	0	•	•	•	•	•	•	0	SFNM	C	LTM	MSK	PR2	PR1	PR0

Figure 28. INT0/INT1 Control Register Formats

15	14									5	4	3	2	1	0
0	0	•	•	•	•	•	•	•	•	0	LTM	MSK	PR2	PR1	PR0

Figure 29. INT2/INT3 Control Register Formats

POLL AND POLL STATUS REGISTERS

These registers contain polling information. The format of these registers is shown in Figure 31. They can only be read. Reading the Poll register constitutes a software poll. This will set the IS bit of the highest priority pending interrupt. Reading the poll status register will not set the IS bit of the highest priority pending interrupt; only the status of pending interrupts will be provided.

Encoding of the Poll and Poll Status register bits are as follows:

S_x: Encoded information that indicates the vector type of the highest priority interrupting source. Valid only when INTREQ = 1.

INTREQ: This bit determines if an interrupt request is present. Interrupt Request = 1; no Interrupt Request = 0.

iRMX™ 86 COMPATIBILITY MODE

This mode allows iRMX 86-80188 compatibility. The interrupt model of iRMX 86 requires one master and multiple slave 8259As in cascaded fashion. When iRMX mode is used, the internal 80188 interrupt controller will be used as a slave controller to an external master interrupt controller. The internal 80188 resources will be monitored through the internal interrupt controller, while the external controller functions as the system master interrupt controller.

Upon reset, the 80188 interrupt controller will be in the non-iRMX 86 mode of operation. To set the controller in the iRMX 86 mode, bit 14 of the Relocation Register should be set.

Because of pin limitations caused by the need to interface to an external 8259A master, the internal interrupt controller will no longer accept external inputs. There are however, enough 80188 interrupt controller inputs (internally) to dedicate one to each timer. In this mode, each timer interrupt source has its own mask bit, IS bit, and control word.

The iRMX 86 operating system requires peripherals to be assigned fixed priority levels. This is incompatible with the normal operation of the 80188 interrupt controller. Therefore, the initialization software must program the proper priority levels for each source. The required priority levels for the internal interrupt sources in iRMX mode are shown in Table 16.

Table 16. Internal Source Priority Level

Priority Level	Interrupt Source
0	Timer 0
1	(reserved)
2	DMA 0
3	DMA 1
4	Timer 1
5	Timer 2

These level assignments must remain fixed in the iRMX 86 mode of operation.

iRMX™ 86 Mode External Interface

The configuration of the 80188 with respect to an external 8259A master is shown in Figure 32. The INT0 input is used as the 80188 CPU interrupt input. INT3 functions as an output to send the 80188 slave-interrupt-request to one of the 8 master-PIC-inputs.

15	14	13		5	4	3	2	1	0
SPEC/NSPEC	0	0	• • • • • • •	0	S4	S3	S2	S1	S0

Figure 30. EOI Register Format

15	14	13		5	4	3	2	1	0
INT REQ	0	0	• • • • • • •	0	S4	S3	S2	S1	S0

Figure 31. Poll Register Format

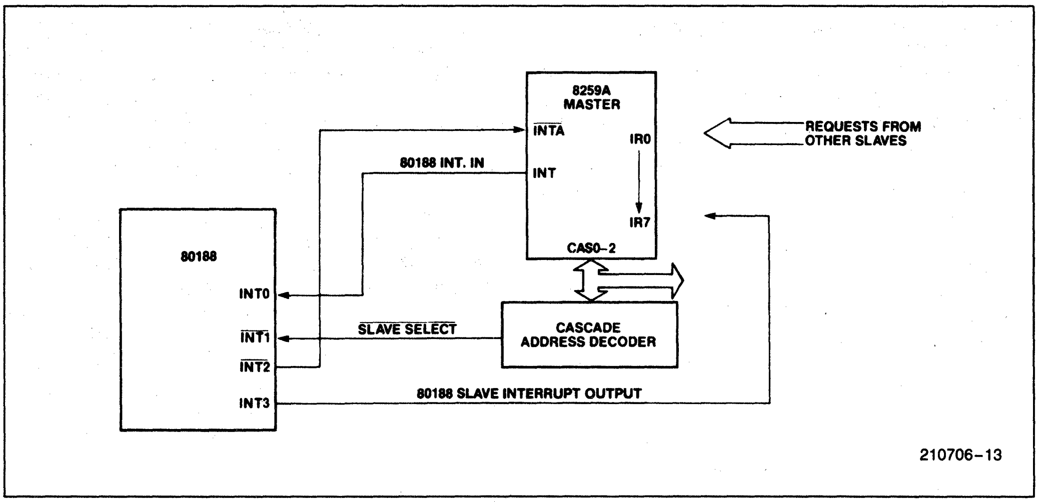


Figure 32. iRMX™ 86 Interrupt Controller Interconnection

Correct master-slave interface requires decoding of the slave addresses (CAS0–2). Slave 8259As do this internally. Because of pin limitations, the 80188 slave address will have to be decoded externally. $\overline{\text{INT1}}$ is used as a slave-select input. Note that the slave vector address is transferred internally, but the READY input must be supplied externally.

$\overline{\text{INT2}}$ is used as an acknowledge output, suitable to drive the $\overline{\text{INTA}}$ input of an 8259A.

Interrupt Nesting

iRMX 86 mode operation allows nesting of interrupt requests. When an interrupt is acknowledged, the priority logic masks off all priority levels except those with equal or higher priority.

Vector Generation in the iRMX™ 86 Mode

Vector generation in iRMX mode is exactly like that of an 8259A slave. The interrupt controller generates an 8-bit vector which the CPU multiplies by four and uses as an address into a vector table. The significant five bits of the vector are user-programmable while the lower three bits are generated by the priority logic. These bits represent the encoding of the priority level requesting service. The significant five bits of the vector are programmed by writing to the Interrupt Vector register at offset 20H.

Specific End-of-Interrupt

In iRMX mode the specific EOI command operates to reset an in-service bit of a specific priority. The user supplies a 3-bit priority-level value that points to an in-service bit to be reset. The command is executed by writing the correct value in the Specific EOI register at offset 22H.

Interrupt Controller Registers in the iRMX™ 86 Mode

All control and command registers are located inside the internal peripheral control block. Figure 33 shows the offsets of these registers.

END-OF-INTERRUPT REGISTER

The end-of-interrupt register is a command register which can only be written. The format of this register is shown in Figure 34. It initiates an EOI command when written by the 80188 CPU.

The bits in the EOI register are encoded as follows:

L_X: Encoded value indicating the priority of the IS bit to be reset.

IN-SERVICE REGISTER

This register can be read from or written into. It contains the in-service bit for each of the internal interrupt sources. The format for this register is shown in Figure 35. Bit positions 2 and 3 correspond to the DMA channels; positions 0, 4, and 5 correspond to the integral timers. The source's IS bit is set when the processor acknowledges its interrupt request.

INTERRUPT REQUEST REGISTER

This register indicates which internal peripherals have interrupt requests pending. The format of this register is shown in Figure 35. The interrupt request bits are set when a request arrives from an internal source, and are reset when the processor acknowledges the request.

MASK REGISTER

The register contains a mask bit for each interrupt source. The format for this register is shown in Figure 35. If the bit in this register corresponding to a particular interrupt source is set, any interrupts from that source will be masked. These mask bits are exactly the same bits which are used in the individual control registers, i.e., changing the state of a mask bit in this register will also change the state of the mask bit in the individual interrupt control register corresponding to the bit.

CONTROL REGISTERS

These registers are the control words for all the internal interrupt sources. The format of these registers is shown in Figure 36. Each of the timers and both of the DMA channels have their own Control Register.

The bits of the Control Registers are encoded as follows:

pr_x: 3-bit encoded field indicating a priority level for the source; note that each source must be programmed at specified levels.

msk: mask bit for the priority level indicated by pr_x bits.

Register	OFFSET
LEVEL 5 CONTROL REGISTER (TIMER 2)	3AH
LEVEL 4 CONTROL REGISTER (TIMER 1)	38H
LEVEL 3 CONTROL REGISTER (DMA 1)	36H
LEVEL 2 CONTROL REGISTER (DMA 0)	34H
LEVEL 0 CONTROL REGISTER (TIMER 0)	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY-LEVEL MASK REGISTER	2AH
MASK REGISTER	28H
SPECIFIC EOI REGISTER	22H
INTERRUPT VECTOR REGISTER	20H

Figure 33. Interrupt Controller Registers (iRMX™ 86 Mode)

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	0	L2	L1	L0

Figure 34. Specific EOI Register Format

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	TMR2	TMR1	D1	D0	0	TMR0

Figure 35. In-Service, Interrupt Request, and Mask Register Format

INTERRUPT VECTOR REGISTER

This register provides the upper five bits of the interrupt vector address. The format of this register is shown in Figure 37. The interrupt controller itself provides the lower three bits of the interrupt vector as determined by the priority level of the interrupt request.

The format of the bits in this register is:

t_x: 5-bit field indicating the upper five bits of the vector address.

PRIORITY-LEVEL MASK REGISTER

This register indicates the lowest priority-level interrupt which will be serviced.

The encoding of the bits in this register is:

m_x: 3-bit encoded field indication priority-level value. All levels of lower priority will be masked.

INTERRUPT STATUS REGISTER

This register is defined exactly as in non-iRMX mode (See Figure 26).

Interrupt Controller and Reset

Upon RESET, the interrupt controller will perform the following actions:

- All SFNM bits reset to 0, implying Fully Nested Mode.
- All PR bits in the various control registers set to 1. This places all sources at lowest priority (level 111).
- All LTM bits reset to 0, resulting in edge-sense mode.
- All Interrupt Service bits reset to 0.
- All Interrupt Request bits reset to 0.
- All MSK (Interrupt Mask) bits set to 1 (mask).
- All C (Cascade) bits reset to 0 (non-cascade).
- All PRM (Priority Mask) bits set to 1, implying no levels masked.
- Initialized to non-iRMX 86 mode.

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	MSK	PR2	PR1	PR0

Figure 36. Control Word Format

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	t4	t3	t2	t1	t0	0	0	0

Figure 37. Interrupt Vector Register Format

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	0	m2	m1	m0

Figure 38. Priority Level Mask Register

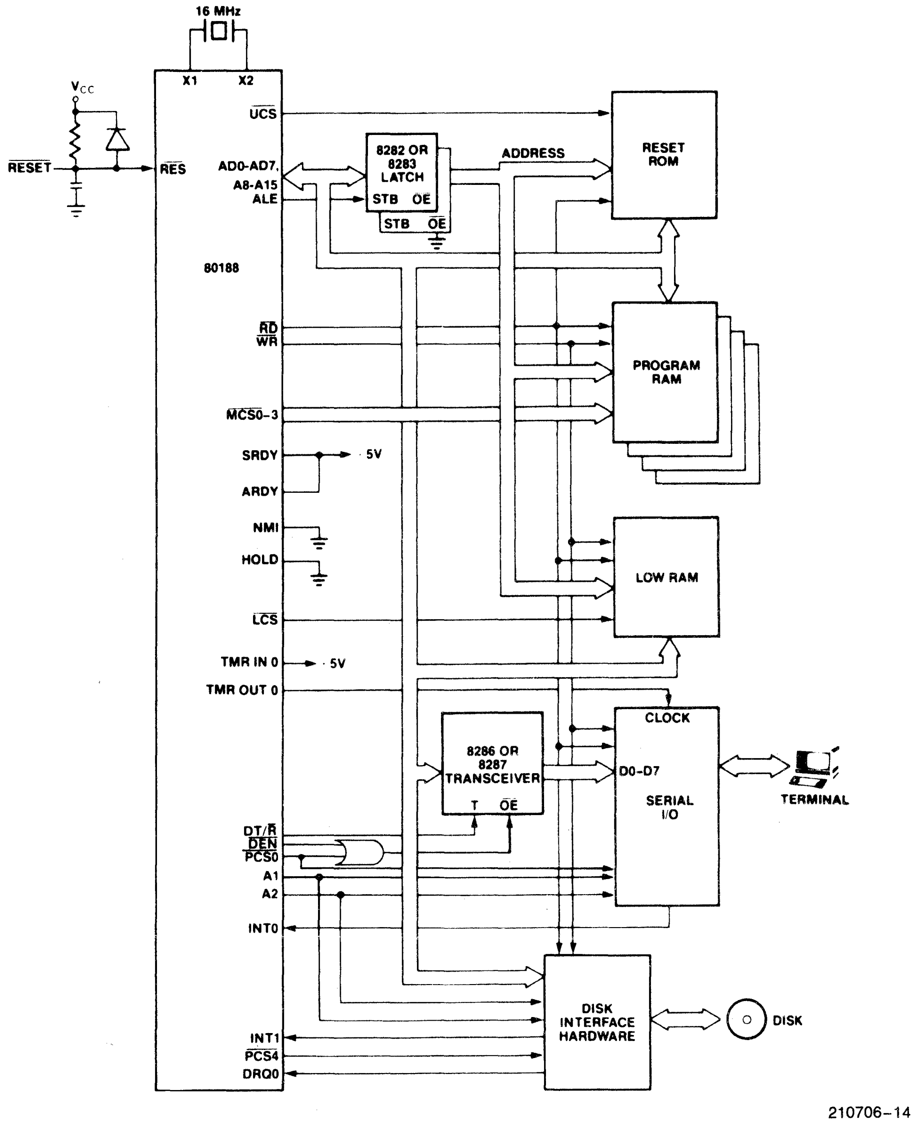


Figure 39. Typical 80188 Computer

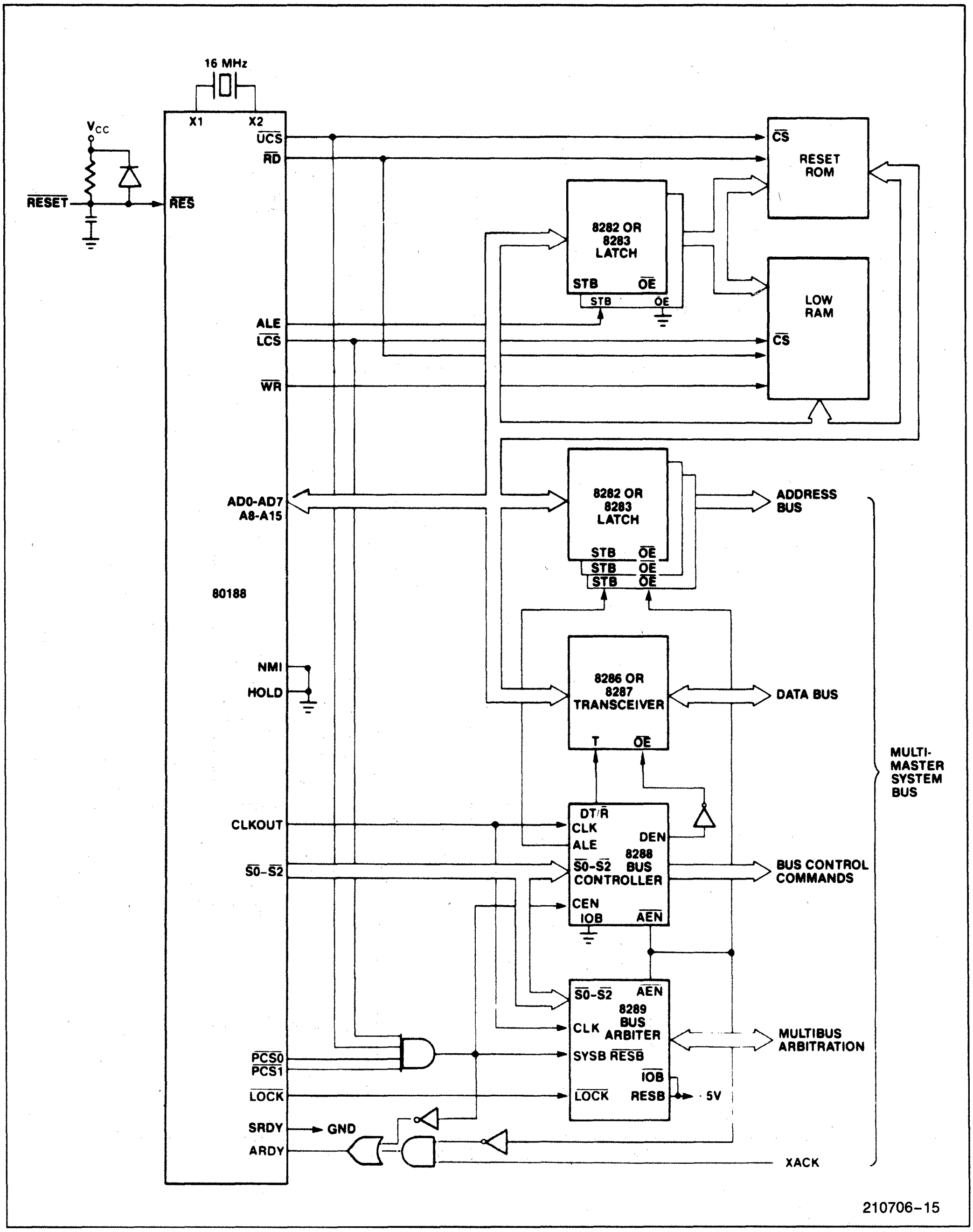


Figure 40. Typical 80188 Multi-Master Bus Interface

PACKAGE

The 80188 is available in two 68 pin hermetic packages. They are the JEDEC type A leadless chip carrier and the JEDEC type CG pin grid array. Figures 41A and 41B illustrate the package dimensions.

NOTE:
The IDT 3M Textool 68-pin JEDEC Socket is required for I2ICE™ 188 operation. See Figure 42 for details.

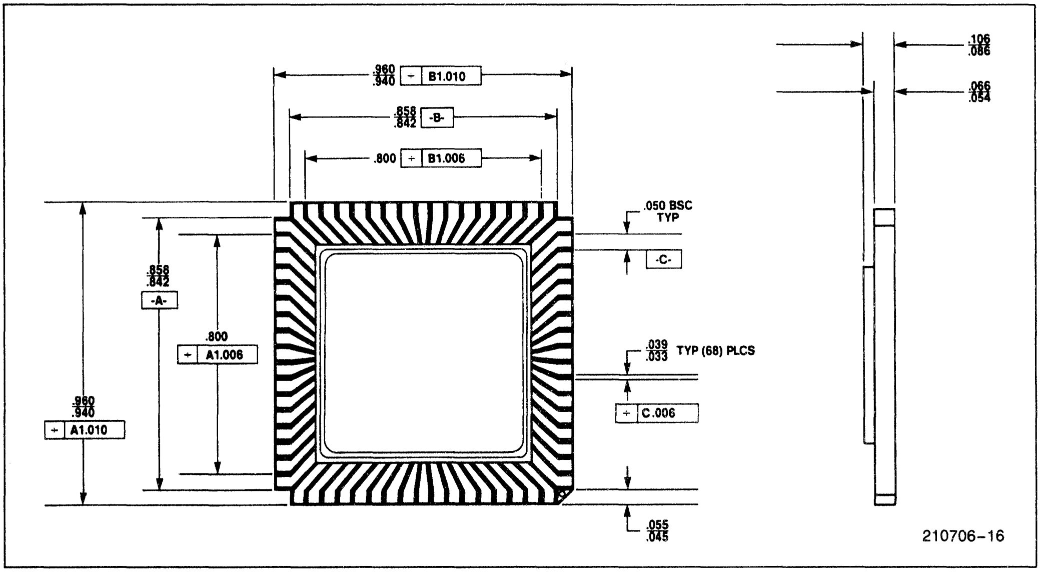


Figure 41A. 80188 JEDEC Type A Package

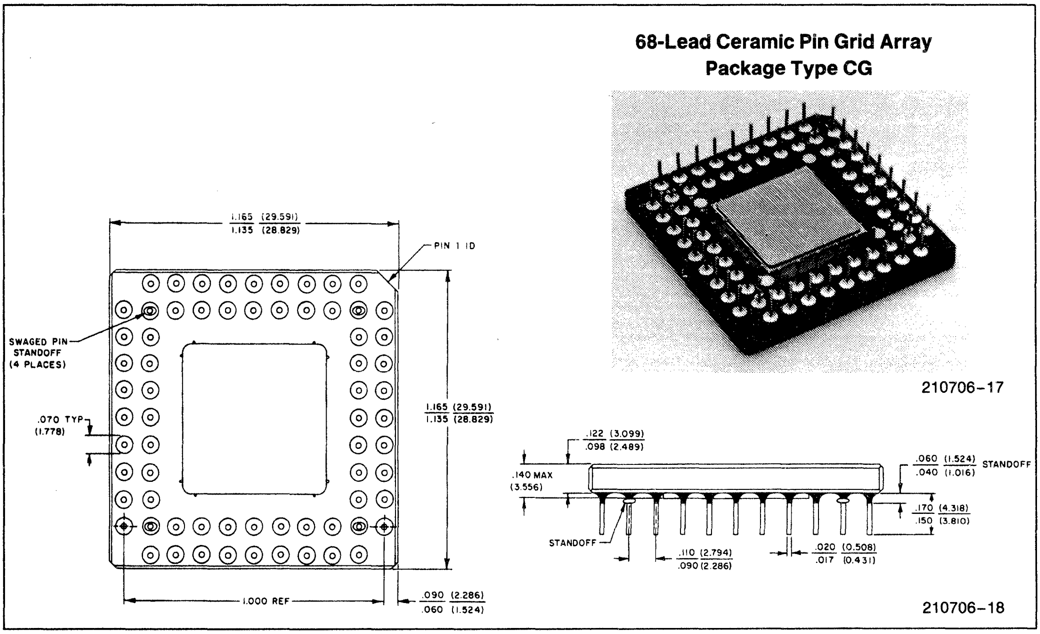
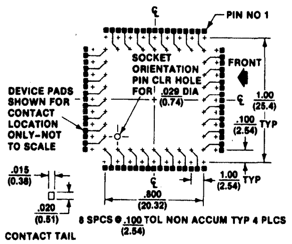
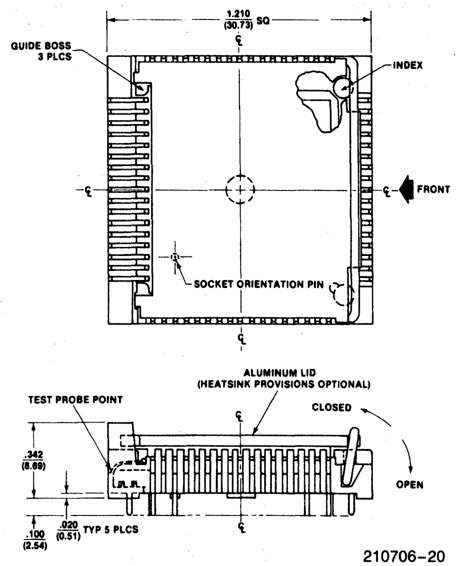


Figure 41B. Ceramic Pin Grid Array Package Type CG

PC BOARD PATTERN



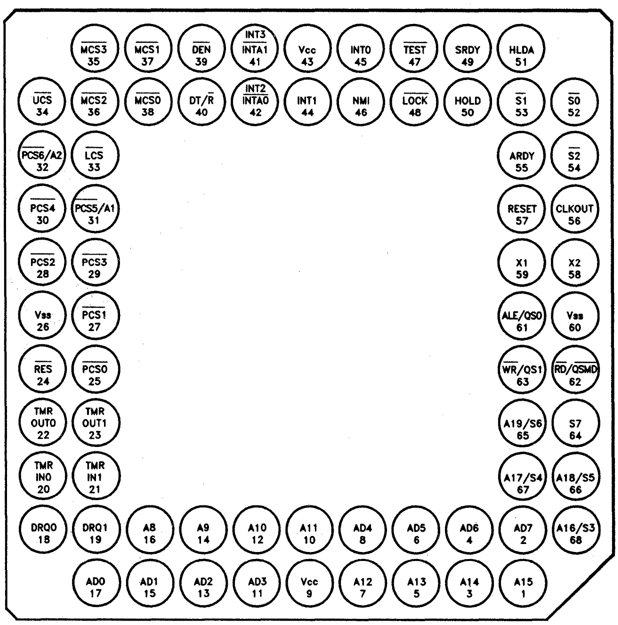
210706-19



210706-20

NOTE:
Physical dimensions shown are for reference only. Please consult 3M Textool for complete information on the socket.

Figure 42. Textool 68 Lead Chip Carrier Socket



Bottom View (Pins Facing Up)

210706-21

Figure 43. Pin Grid Array, PLCC, and LCC Socket Pinout

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias . . . . 0°C to +70°C
Case Temperature under Bias . . . . . . 0°C to +110°C
Storage Temperature . . . . −65°C to +150°C
Voltage on any Pin with
Respect to Ground . . . . −1.0V to +7V
Power Dissipation . . . . 3 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to +70°C, T_C = 0°C to +110°C, V_{CC} = 5V ±10%)

Applicable to 80188 (8 MHz), and 80188-10 (10 MHz)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	0.5	+0.8	V	
V_{IH}	Input High Voltage (All except X1 and $\overline{RES}$)	2.0	V_{CC} + 0.5	V	
V_{IH1}	Input High Voltage ($\overline{RES}$)	3.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_a = 2.5 mA for $\overline{S0}$–$\overline{S2}$ I_a = 2.0 mA for all other outputs
V_{OH}	Output High Voltage	2.4		V	I_{oa} = −400 μA
I_{CC}	Power Supply Current		550 450	mA	Max measured at T_A = 0°C T_A = 70°C
I_{LI}	Input Leakage Current		±10	μA	0V < V_{IN} < V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V < V_{OUT} < V_{CC}
V_{CLO}	Clock Output Low		0.6	V	I_a = 4.0 mA
V_{CHO}	Clock Output High	4.0		V	I_{oa} = −200 μA
V_{CLI}	Clock Input Low Voltage	−0.5	0.6	V	
V_{CHI}	Clock Input High Voltage	3.9	V_{CC} + 1.0	V	
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		20	pF	

PIN TIMINGS

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, T_C = 0°C to +110°C, V_{CC} = 5V ±10%)

80188 Timing Requirements All Timings Measured At 1.5 Volts Unless Otherwise Noted

Symbol	Parameter	80188 (8 MHz)		80188-10 (10 MHz)		Units	Test Conditions
		Min	Max	Min	Max		
T_{DVCL}	Data in Setup (A/D)	20		15		ns	
T_{CLDX}	Data in Hold (A/D)	10		8		ns	
T_{ARYHCH}	Asynchronous Ready (AREADY) active setup time*	20		15		ns	
T_{ARYLCL}	AREADY inactive setup time	35		25		ns	
T_{CHARYX}	AREADY hold time	15		15		ns	
T_{ARYCHL}	Asynchronous Ready inactive hold time	15		15		ns	
T_{SRYCL}	Synchronous Ready (SREADY) transition setup time	20		20		ns	
T_{CLSRY}	SREADY transition hold time	15		15		ns	
T_{HVCL}	HOLD Setup*	25		20		ns	
T_{INVCH}	INTR, NMI, $\overline{TEST}$, TIMERIN, Setup*	25		25		ns	
T_{INVCL}	DRQ0, DRQ1, Setup*	25		20		ns	

80188 Master Interface Timing Responses

Symbol	Parameter	80188 Min	80188 Max	80188-10 Min	80188-10 Max	Units	Test Conditions
T_{CLAV}	Address Valid Delay	5	55	5	50	ns	C_L = 20–200 pF all outputs (except T_{CLTMV}) @ 8 & 10 MHz
T_{CLAX}	Address Hold	10		10		ns	
T_{CLAZ}	Address Float Delay	T_{CLAX}	35	T_{CLAX}	30	ns	
T_{CHCZ}	Command Lines Float Delay		45		40	ns	
T_{CHCV}	Command Lines Valid Delay (after float)		55		45	ns	
T_{LHLL}	ALE Width	$T_{CLCL}-35$		$T_{CLCL}-30$		ns	
T_{CHLH}	ALE Active Delay		35		30	ns	
T_{CHLL}	ALE Inactive Delay		35		30	ns	
T_{LLAX}	Address Hold to ALE Inactive	$T_{CHCL}-25$		$T_{CHCL}-20$		ns	
T_{CLDV}	Data Valid Delay	10	44	10	40	ns	
T_{CLDOX}	Data Hold Time	10		10		ns	
T_{WHDX}	Data Hold after WR	$T_{CLCL}-40$		$T_{CLCL}-34$		ns	
T_{CVCTV}	Control Active Delay 1	10	70	5	56	ns	
T_{CHCTV}	Control Active Delay 2	10	55	10	44	ns	
T_{CVCTX}	Control Inactive Delay	5	55	5	44	ns	
T_{CVDEX}	$\overline{DEN}$ Inactive Delay (Non-Write Cycle)	10	70	10	56	ns	

*To guarantee recognition at next clock.

PIN TIMINGS (Continued)

A.C. CHARACTERISTICS

(T_A = 0°C to +70°C, T_C = 0°C to +110°C, V_{CC} = 5V ±10%) (Continued)

80188 Master Interface Timing Responses (Continued)

Symbol	Parameter	80188 (8 MHz)		80188-10 (10 MHz)		Units	Test Conditions
		Min	Max	Min	Max		
T_{AZRL}	Address Float to $\overline{RD}$ Active	0		0		ns	
T_{CLRL}	$\overline{RD}$ Active Delay	10	70	10	56	ns	
T_{CLRH}	$\overline{RD}$ Inactive Delay	10	55	10	44	ns	
T_{RHAV}	$\overline{RD}$ Inactive to Address Active	$T_{CLCL}-40$		$T_{CLCL}-40$		ns	
T_{CLHAV}	HLDA Valid Delay	5	50	5	40	ns	
T_{RLRH}	$\overline{RD}$ Width	$2T_{CLCL}-50$		$2T_{CLCL}-46$		ns	
T_{WLWH}	$\overline{WR}$ Width	$2T_{CLCL}-40$		$2T_{CLCL}-34$		ns	
T_{AVAL}	Address Valid to ALE Low	$T_{CLCH}-25$		$T_{CLCH}-19$		ns	
T_{CHSV}	Status Active Delay	10	55	10	45	ns	
T_{CLSH}	Status Inactive Delay	10	65	10	50	ns	
T_{CLTMV}	Timer Output Delay		60		48	ns	100 pF max
T_{CLRO}	Reset Delay		60		48	ns	
T_{CHQSV}	Queue Status Delay		35		28	ns	
T_{CHDX}	Status Hold Time	10		10		ns	
T_{AVCH}	Address Valid to Clock High	10		10		ns	
T_{CLLV}	$\overline{LOCK}$ Valid/Invalid Delay	5	65	5	60	ns	

80188 Chip-Select Timing Responses

Symbol	Parameter	80188 Min	80188 Max	80188-10 Min	80188-10 Max	Units	Test Conditions
T_{CLCSV}	Chip-Select Active Delay		66		45	ns	
T_{CXCSX}	Chip-Select Hold from Command Inactive	35		35		ns	
T_{CHCSX}	Chip-Select Inactive Delay	5	35	5	32	ns	

80188 CLKIN Requirements

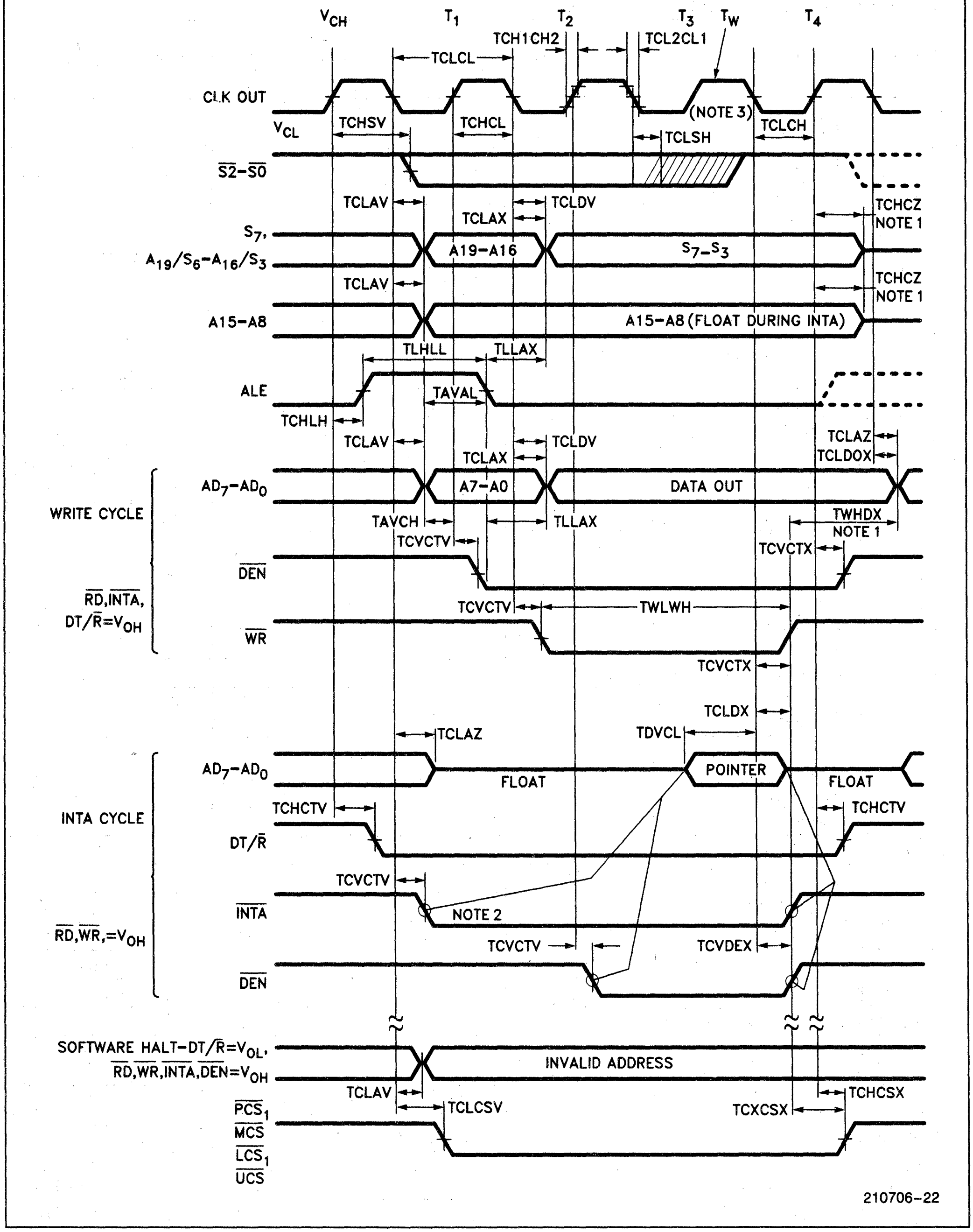
Symbol	Parameter	80188 Min	80188 Max	80188-10 Min	80188-10 Max	Units	Test Conditions
T_{CKIN}	CLKIN Period	62.5	250	50	250	ns	
T_{CKHL}	CLKIN Fall Time		10		10	ns	3.5 to 1.0V
T_{CKLH}	CLKIN Rise Time		10		10	ns	1.0 to 3.5V
T_{CLCK}	CLKIN Low Time	25		20		ns	1.5V
T_{CHCK}	CLKIN High Time	25		20		ns	1.5V

80188 CLKOUT Timing (200 pF load)

Symbol	Parameter	80188 Min	80188 Max	80188-10 Min	80188-10 Max	Units	Test Conditions
T_{CICO}	CLKIN to CLKOUT Skew		50		25	ns	
T_{CLCL}	CLKOUT Period	125	500	100	500	ns	
T_{CLCH}	CLKOUT Low Time	$\frac{1}{2} T_{CLCL}-7.5$		$\frac{1}{2} T_{CLCL}-6.0$		ns	1.5V
T_{CHCL}	CLKOUT High Time	$\frac{1}{2} T_{CLCL}-7.5$		$\frac{1}{2} T_{CLCL}-6.0$		ns	1.5V
T_{CH1CH2}	CLKOUT Rise Time		15		12	ns	1.0 to 3.5V
T_{CL2CL1}	CLKOUT Fall Time		15		12	ns	3.5 to 1.0V

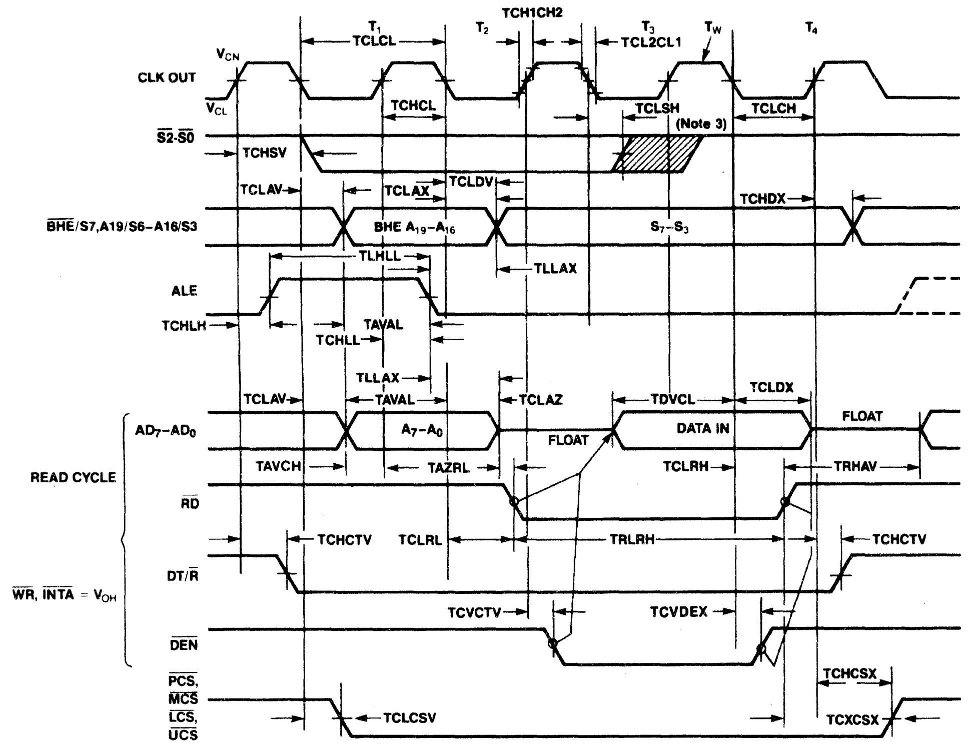
WAVEFORMS

Major Cycle Timing



WAVEFORMS (Continued)

Major Cycle Timing (Continued)

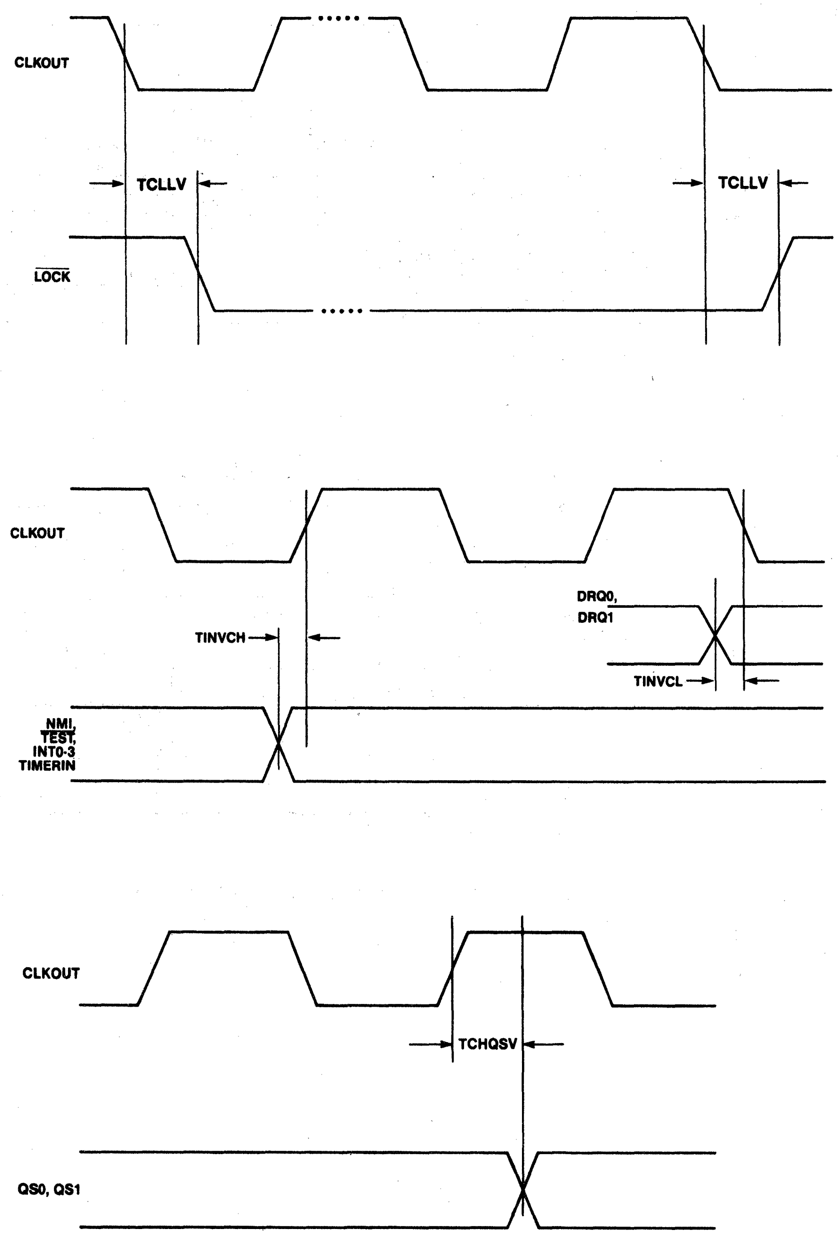


210706-23

NOTES:

1. Following a Write cycle, the Local Bus is floated by the 80188 only when the 80188 enters a "Hold Acknowledge" state.
2. INTA occurs one clock later in RMX-mode.
3. Status inactive just prior to T_4.

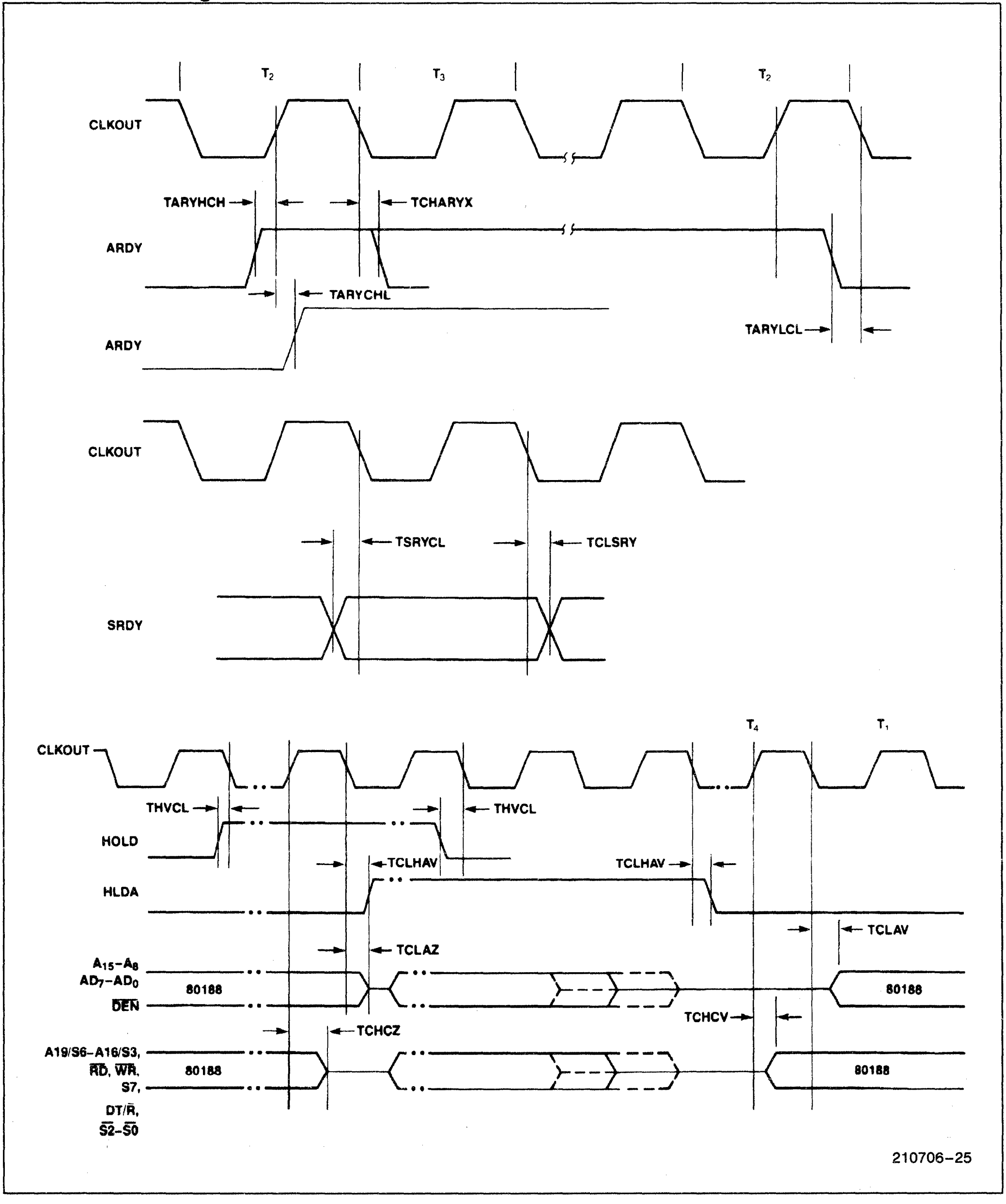
WAVEFORMS (Continued)



210706–24

WAVEFORMS (Continued)

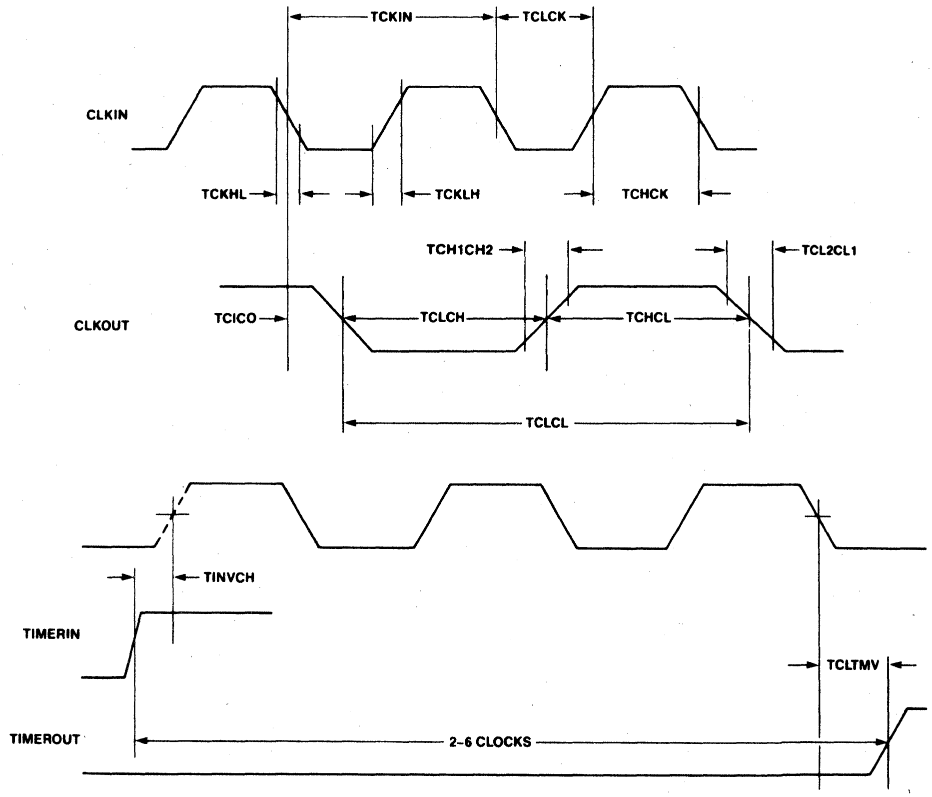
Hold-Hlda Timing



210706–25

WAVEFORMS (Continued)

Timer On 80188



210706–26

80188 INSTRUCTION TIMINGS

The following instruction timings represent the minimum execution time in clock cycles for each instruction. The timings given are based on the following assumptions:

- The opcode, along with any data or displacement required for execution of a particular instruction, has been prefetched and resides in the queue at the time it is needed.
- No wait states or bus HOLDS occur.
- All word-data is located on even-address boundaries.

All jumps and calls include the time required to fetch the opcode of the next instruction at the destination address.

All instructions which involve memory reference can require one (and in some cases, two) additional clocks above the minimum timings shown. This is due to the asynchronous nature of the handshake between the BIU and the Execution unit.

INSTRUCTION SET SUMMARY

Function	Format				Clock Cycles	Comments
DATA TRANSFER						
MOV = **Move:**						
Register to register/memory	1 0 0 0 1 0 0 w	mod reg r/m			2/12*	
Register/memory to register	1 0 0 0 1 0 1 w	mod reg r/m			2/9*	
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 000 r/m	data	data if w = 1	12/13*	8/16-bit
Immediate to register	1 0 1 1 w reg	data	data if w = 1		3/4	8/16-bit
Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high		9*	
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high		8*	
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 reg r/m			2/13	
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 reg r/m			2/15	
PUSH = **Push:**						
Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m			20	
Register	0 1 0 1 0 reg				14	
Segment register	0 0 0 reg 1 1 0				13	
Immediate	0 1 1 0 1 0 s 0	data	data if s = 0		14	
PUSHA = Push All	0 1 1 0 0 0 0 0				68	
POP = **Pop:**						
Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m			24	
Register	0 1 0 1 1 reg				14	
Segment register	0 0 0 reg 1 1 1	(reg≠01)			12	
POPA = Pop All	0 1 1 0 0 0 0 1				83	
XCHG = **Exchange:**						
Register/memory with register	1 0 0 0 0 1 1 w	mod reg r/m			4/17*	
Register with accumulator	1 0 0 1 0 reg				3	
IN = **Input from:**						
Fixed port	1 1 1 0 0 1 0 w	port			10*	
Variable port	1 1 1 0 1 1 0 w				8*	
OUT = **Output to:**						
Fixed port	1 1 1 0 0 1 1 w	port			9*	
Variable port	1 1 1 0 1 1 1 w				7*	
XLAT = Translate byte to AL	1 1 0 1 0 1 1 1				15	
LEA = Load EA to register	1 0 0 0 1 1 0 1	mod reg r/m			6	
LDS = Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m	(mod≠11)		26	
LES = Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m	(mod≠11)		26	
LAHF = Load AH with flags	1 0 0 1 1 1 1 1				2	
SAHF = Store AH into flags	1 0 0 1 1 1 1 0				3	
PUSHF = Push flags	1 0 0 1 1 1 0 0				13	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
DATA TRANSFER (Continued)						
POPF = Pop flags	10011101				12	
SEGMENT = Segment Override:						
CS	00101110				2	
SS	00110110				2	
DS	00111110				2	
ES	00100110				2	
ARITHMETIC						
ADD = Add:						
Reg/memory with register to either	000000dw	mod reg r/m			3/10*	
Immediate to register/memory	100000sw	mod 0 0 0 r/m	data	data if s w=01	4/16*	
Immediate to accumulator	0000010w	data	data if w=1		3/4	8/16-bit
ADC = Add with carry:						
Reg/memory with register to either	000100dw	mod reg r/m			3/10*	
Immediate to register/memory	100000sw	mod 0 1 0 r/m	data	data if s w=01	4/16*	
Immediate to accumulator	0001010w	data	data if w=1		3/4	8/16-bit
INC = Increment:						
Register/memory	1111111w	mod 0 0 0 r/m			3/15*	
Register	01000 reg				3	
SUB = Subtract:						
Reg/memory and register to either	001010dw	mod reg r/m			3/10*	
Immediate from register/memory	100000sw	mod 1 0 1 r/m	data	data if s w=01	4/16*	
Immediate from accumulator	0010110w	data	data if w=1		3/4	8/16-bit
SBB = Subtract with borrow:						
Reg/memory and register to either	000110dw	mod reg r/m			3/10*	
Immediate from register/memory	100000sw	mod 0 1 1 r/m	data	data if s w=01	4/16*	
Immediate from accumulator	0001110w	data	data if w=1		3/4	8/16-bit
DEC = Decrement:						
Register/memory	1111111w	mod 0 0 1 r/m			3/15*	
Register	01001 reg				3	
CMP = Compare:						
Register/memory with register	0011101w	mod reg r/m			3/10*	
Register with register/memory	0011100w	mod reg r/m			3/10*	
Immediate with register/memory	100000sw	mod 1 1 1 r/m	data	data if s w=01	3/10*	
Immediate with accumulator	0011110w	data	data if w=1		3/4	8/16-bit
NEG = Change sign	1111011w	mod 0 1 1 r/m			3	
AAA = ASCII adjust for add	00110111				8	
DAA = Decimal adjust for add	00100111				4	
AAS = ASCII adjust for subtract	00111111				7	
DAS = Decimal adjust for subtract	00101111				4	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
ARITHMETIC (Continued)						
MUL = Multiply (unsigned):	1 1 1 1 0 1 1 w	mod 100 r/m				
Register-Byte					26–28	
Register-Word					35–37	
Memory-Byte					32–34	
Memory-Word					41–43*	
IMUL = Integer multiply (signed):	1 1 1 1 0 1 1 w	mod 1 0 1 r/m				
Register-Byte					25–28	
Register-Word					34–37	
Memory-Byte					31–34	
Memory-Word					40–43*	
IMUL = Integer immediate multiply (signed)	0 1 1 0 1 0 s 1	mod reg r/m	data	data if s=0	22–25/ 29–32	
DIV = Divide (unsigned):	1 1 1 1 0 1 1 w	mod 1 1 0 r/m				
Register-Byte					29	
Register-Word					38	
Memory-Byte					35	
Memory-Word					44*	
IDIV = Integer divide (signed):	1 1 1 1 0 1 1 w	mod 1 1 1 r/m				
Register-Byte					44–52	
Register-Word					53–61	
Memory-Byte					50–58	
Memory-Word					59–67*	
AAM = ASCII adjust for multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0			19	
AAD = ASCII adjust for divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0			15	
CBW = Convert byte to word	1 0 0 1 1 0 0 0				2	
CWD = Convert word to double word	1 0 0 1 1 0 0 1				4	
LOGIC						
Shift/Rotate Instructions:						
Register/Memory by 1	1 1 0 1 0 0 0 w	mod TTT r/m			2/15	
Register/Memory by CL	1 1 0 1 0 0 1 w	mod TTT r/m			5+n/17+n	
Register/Memory by Count	1 1 0 0 0 0 0 w	mod TTT r/m	count		5+n/17+n	
	TTT	**Instruction**				
	0 0 0	ROL				
	0 0 1	ROR				
	0 1 0	RCL				
	0 1 1	RCR				
	1 0 0	SHL/SAL				
	1 0 1	SHR				
	1 1 1	SAR				
AND = And:						
Reg/memory and register to either	0 0 1 0 0 0 d w	mod reg r/m			3/10*	
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w=1	4/16*	
Immediate to accumulator	0 0 1 0 0 1 0 w	data	data if w=1		3/4	8/16-bit

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
LOGIC (Continued)						
TEST = And function to flags, no result:						
Register/memory and register	1 0 0 0 0 1 0 w	mod reg r/m			3/10*	
Immediate data and register/memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1	4/10*	
Immediate data and accumulator	1 0 1 0 1 0 0 w	data	data if w = 1		3/4	8/16-bit
OR = Or:						
Reg/memory and register to either	0 0 0 0 1 0 d w	mod reg r/m			3/10*	
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w = 1	4/16*	
Immediate to accumulator	0 0 0 0 1 1 0 w	data	data if w = 1		3/4	8/16-bit
XOR = Exclusive or:						
Reg/memory and register to either	0 0 1 1 0 0 d w	mod reg r/m			3/10*	
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w = 1	4/16*	
Immediate to accumulator	0 0 1 1 0 1 0 w	data	data if w = 1		3/4	8/16-bit
NOT = Invert register/memory	1 1 1 1 0 1 1 w	mod 0 1 0 r/m			3	
STRING MANIPULATION:						
MOVS = Move byte/word	1 0 1 0 0 1 0 w				14*	
CMPS = Compare byte/word	1 0 1 0 0 1 1 w				22*	
SCAS = Scan byte/word	1 0 1 0 1 1 1 w				15*	
LODS = Load byte/wd to AL/AX	1 0 1 0 1 1 0 w				12*	
STOS = Stor byte/wd from AL/A	1 0 1 0 1 0 1 w				10*	
INS = Input byte/wd from DX port	0 1 1 0 1 1 0 w				14	
OUTS = Output byte/wd to DX port	0 1 1 0 1 1 1 w				14	
Repeated by count in CX						
MOVS = Move string	1 1 1 1 0 0 1 0	1 0 1 0 0 1 0 w			8+8n*	
CMPS = Compare string	1 1 1 1 0 0 1 z	1 0 1 0 0 1 1 w			5+22n*	
SCAS = Scan string	1 1 1 1 0 0 1 z	1 0 1 0 1 1 1 w			5+15n*	
LODS = Load string	1 1 1 1 0 0 1 0	1 0 1 0 1 1 0 w			6+11n*	
STOS = Store string	1 1 1 1 0 0 1 0	1 0 1 0 1 0 1 w			6+9n*	
INS = Input string	1 1 1 1 0 0 1 0	0 1 1 0 1 1 0 w			8+8n*	
OUTS = Output string	1 1 1 1 0 0 1 0	0 1 1 0 1 1 1 w			8+8n*	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
CONTROL TRANSFER			
CALL = Call:			
Direct within segment	11101000 | disp-low | disp-high	19	
Register/memory indirect within segment	11111111 | mod 0 1 0 r/m	17/27	
Direct intersegment	10011010 | segment offset segment selector	31	
Indirect intersegment	11111111 | mod 0 1 1 r/m (mod ≠ 11)	54	
JMP = Unconditional jump:			
Short/long	11101011 | disp-low	14	
Direct within segment	11101001 | disp-low | disp-high	14	
Register/memory indirect within segment	11111111 | mod 1 0 0 r/m	11/21	
Direct intersegment	11101010 | segment offset segment selector	14	
Indirect intersegment	11111111 | mod 1 0 1 r/m (mod ≠ 11)	34	
RET = Return from CALL:			
Within segment	11000011	20	
Within seg adding immed to SP	11000010 | data-low | data-high	22	
Intersegment	11001011	30	
Intersegment adding immediate to SP	11001010 | data-low | data-high	33	
JE/JZ = Jump on equal/zero	01110100 | disp	4/13	JMP not taken/JMP taken
JL/JNGE = Jump on less/not greater or equal	01111100 | disp	4/13	
JLE/JNG = Jump on less or equal/not greater	01111110 | disp	4/13	
JB/JNAE = Jump on below/not above or equal	01110010 | disp	4/13	
JBE/JNA = Jump on below or equal/not above	01110110 | disp	4/13	
JP/JPE = Jump on parity/parity even	01111010 | disp	4/13	
JO = Jump on overflow	01110000 | disp	4/13	
JS = Jump on sign	01111000 | disp	4/13	
JNE/JNZ = Jump on not equal/not zero	01110101 | disp	4/13	
JNL/JGE = Jump on not less/greater or equal	01111101 | disp	4/13	
JNLE/JG = Jump on not less or equal/greater	01111111 | disp	4/13	
JNB/JAE = Jump on not below/above or equal	01110011 | disp	4/13	
JNBE/JA = Jump on not below or equal/above	01110111 | disp	4/13	
JNP/JPO = Jump on not par/par odd	01111011 | disp	4/13	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
CONTROL TRANSFER (Continued)			
JNO = Jump on not overflow	01110001 | disp	4/13	
JNS = Jump on not sign	01111001 | disp	4/13	
JCXZ = Jump on CX zero	11100011 | disp	5/15	
LOOP = Loop CX times	11100010 | disp	6/16	LOOP not taken/LOOP taken
LOOPZ/LOOPE = Loop while zero/equal	11100001 | disp	6/16	
LOOPNZ/LOOPNE = Loop while not zero/equal	11100000 | disp	6/16	
ENTER = Enter Procedure	11001000 | data-low | data-high | L		
L = 0		15	
L = 1		25	
L > 1		22+16(n−1)	
LEAVE = Leave Procedure	11001001	8	
INT = **Interrupt:**			
Type specified	11001101 | type	47	
Type 3	11001100	45	if INT. taken/ if INT. not taken
INTO = Interrupt on overflow	11001110	48/4	
IRET = Interrupt return	11001111	28	
BOUND = Detect value out of range	01100010 | mod reg r/m	33-35	
PROCESSOR CONTROL			
CLC = Clear carry	11111000	2	
CMC = Complement carry	11110101	2	
STC = Set carry	11111001	2	
CLD = Clear direction	11111100	2	
STD = Set direction	11111101	2	
CLI = Clear interrupt	11111010	2	
STI = Set interrupt	11111011	2	
HLT = Halt	11110100	2	
WAIT = Wait	10011011	6	if $\overline{\text{test}}$ = 0
LOCK = Bus lock prefix	11110000	2	
ESC = Processor Extension Escape	11011TTT | mod LLL r/m	6	
	(TTT LLL are opcode to processor extension)		

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

FOOTNOTES

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as a REG field

if mod = 00 then DISP = 0*, disp-low and disp-high are absent

if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent

if mod = 10 then DISP = disp-high: disp-low

if r/m = 000 then EA = (BX) + (SI) + DISP

if r/m = 001 then EA = (BX) + (DI) + DISP

if r/m = 010 then EA = (BP) + (SI) + DISP

if r/m = 011 then EA = (BP) + (DI) + DISP

if r/m = 100 then EA = (SI) + DISP

if r/m = 101 then EA = (DI) + DISP

if r/m = 110 then EA = (BP) + DISP*

if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

EA calculation time is 4 clock cycles for all modes, and is included in the execution times given whenever appropriate.

Segment Override Prefix

0	0	1	reg	1	1	0

reg is assigned according to the following:

reg	Segment Register
00	ES
01	CS
10	SS
11	DS

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

8087
NUMERIC DATA COPROCESSOR
8087/8087-2/8087-1

- **High Performance Numeric Data Coprocessor**
- **Adds Arithmetic, Trigonometric, Exponential, and Logarithmic Instructions to the Standard 8086 and 80186 Instruction Set for All Data Types**
- **CPU/8087 Supports 7 Data Types: 16-, 32-, 64-Bit Integers, 32-, 64-, 80-Bit Floating Point, and 18-Digit BCD Operands**
- **All 24 Addressing Modes Available with 8086, 8088, 80186, 80188 CPUs.**
- **Compatible with IEEE Floating Point Standard 754**
- **Available in 5 MHz (8087), 8 MHz (8087-2) and 10 MHz (8087-1): 8 MHz 80186 system operation supported with the 8087-1.**
- **Adds 8 x 80-Bit Individually Addressable Register Stack**
- **7 Built-in Exception Handling Functions**
- **MULTIBUS® System Compatible Interface**

The 8087 Numeric Data Coprocessor provies the instructions and data types needed for high performance numeric numeric applications, providing up to 100 times the performance of a CPU alone. The 8087 is implemented in N-channel, depletion load, silicon gate technology (HMOS III), housed in a 40-pin package. Sixty-eight numeric processing instructions are added to the 8086, 80186 instruction sets, and eight 80-bit registers are added to the register set. The 8087 is compatible with the IEEE Floating Point Standard 754.

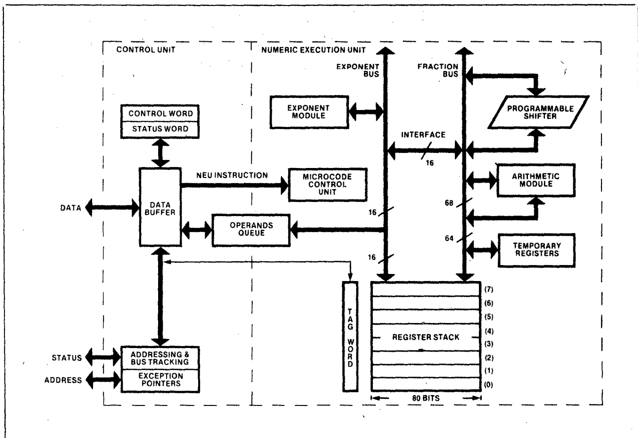


Figure 1. 8087 Block Diagram

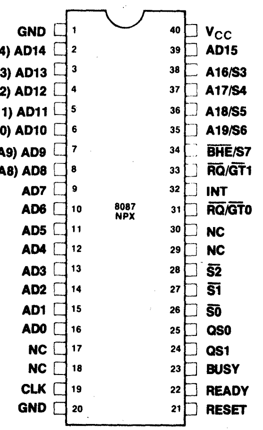


Figure 2. 8087 Pin Configuration

October 1985
ORDER NUMBER: 205835-005

Table 1. 8087 Pin Description

Symbol	Type	Name and Function
AD15-AD0	I/O	**Address Data:** These lines constitute the time multiplexed memory address (T_1) and data (T_2, T_3, T_W, T_4) bus. A0 is analogous to $\overline{BHE}$ for the lower byte of the data bus, pins D7-D0. It is LOW during T_1 when a byte is to be transferred on the lower portion of the bus in memory operations. Eight-bit oriented devices tied to the lower half of the bus would normally use A0 to condition chip select functions. These lines are active HIGH. They are input/output lines for 8087-driven bus cycles and are inputs which the 8087 monitors when the CPU is in control of the bus. A15-A8 do not require an address latch in an 8088/8087 or 80188/8087. The 8087 will supply an address for the T_1-T_4 period.
A19/S6, A18/S5, A17/S4, A16/S3	I/O	**Address Memory:** During T_1 these are the four most significant address lines for memory operations. During memory operations, status information is available on these lines during T_2, T_3, T_W, and T_4. For 8087-controlled bus cycles, S6, S4, and S3 are reserved and currently one (HIGH), while S5 is always LOW. These lines are inputs which the 8087 monitors when the CPU is in control of the bus.
BHE/S7	I/O	**Bus High Enable:** During T_1 the bus high enable signal ($\overline{BHE}$) should be used to enable data onto the most significant half of the data bus, pins D15-D8. Eight-bit-oriented devices tied to the upper half of the bus would normally use $\overline{BHE}$ to condition chip select functions. $\overline{BHE}$ is LOW during T_1 for read and write cycles when a byte is to be transferred on the high portion of the bus. The S7 status information is available during T_2, T_3, T_W, and T_4. The signal is active LOW. S7 is an input which the 8087 monitors during the CPU-controlled bus cycles.
$\overline{S2}$, $\overline{S1}$, $\overline{S0}$	I/O	**Status:** For 8087-driven bus cycles, these status lines are encoded as follows: $\overline{S2}$ — $\overline{S1}$ — $\overline{S0}$ 0 (LOW) X X Unused 1 (HIGH) 0 0 Unused 1 0 1 Read Memory 1 1 0 Write Memory 1 1 1 Passive Status is driven active during T_4, remains valid during T_1 and T_2, and is returned to the passive state (1, 1, 1) during T_3 or during T_W when READY is HIGH. This status is used by the 8288 Bus Controller (or the 82188 Integrated Bus Controller with an 80186/80188 CPU) to generate all memory access control signals. Any change in $\overline{S2}$, $\overline{S1}$, or $\overline{S0}$ during T_4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T_3 or T_W is used to indicate the end of a bus cycle. These signals are monitored by the 8087 when the CPU is in control of the bus.
$\overline{RQ}/\overline{GT0}$	I/O	**Request/Grant:** This request/grant pin is used by the 8087 to gain control of the local bus from the CPU for operand transfers or on behalf of another bus master. It must be connected to one of the two processor request/grant pins. The request grant sequence on this pin is as follows: 1. A pulse one clock wide is passed to the CPU to indicate a local bus request by either the 8087 or the master connected to the 8087 $\overline{RQ}/\overline{GT1}$ pin. 2. The 8087 waits for the grant pulse and when it is received will either initiate bus transfer activity in the clock cycle following the grant or pass the grant out on the $\overline{RQ}/\overline{GT1}$ pin in this clock if the initial request was for another bus master. 3. The 8087 will generate a release pulse to the CPU one clock cycle after the completion of the last 8087 bus cycle or on receipt of the release pulse from the bus master on $\overline{RQ}/\overline{GT1}$. For 80186/80188 systems the same sequence applies except $\overline{RQ}/\overline{GT}$ signals are converted to appropriate HOLD, HLDA signals by the 82188 Integrated Bus Controller. This is to conform with 80186/80188's HOLD, HLDA bus exchange protocol. Refer to the 82188 data sheet for further

Table 1. 8087 Pin Description (Continued)

Symbol	Type	Name and Function
$\overline{RQ}/\overline{GT}1$	I/O	**Request/Grant:** This request/grant pin is used by another local bus master to force the 8087 to request the local bus. If the 8087 is not in control of the bus when the request is made the request/grant sequence is passed through the 8087 on the $\overline{RQ}/\overline{GT}0$ pin one cycle later. Subsequent grant and release pulses are also passed through the 8087 with a two and one clock delay, respectively, for resynchronization. $\overline{RQ}/\overline{GT}1$ has an internal pullup resistor, and so may be left unconnected. If the 8087 has control of the bus the request/grant sequence is as follows: 1. A pulse 1 CLK wide from another local bus master indicates a local bus request to the 8087 (pulse 1). 2. During the 8087's next T_4 or T_1 a pulse 1 CLK wide from the 8087 to the requesting master (pulse 2) indicates that the 8087 has allowed the local bus to float and that it will enter the "RQ/GT acknowledge" state at the next CLK. The 8087's control unit is disconnected logically from the local bus during "RQ/GT acknowledge." 3. A pulse 1 CLK wide from the requesting master indicates to the 8087 (pulse 3) that the "RQ/GT" request is about to end and that the 8087 can reclaim the local bus at the next CLK. Each master-master exchange of the local bus is a sequence of 3 pulses. There must be one dead CLK cycle after each bus exchange. Pulses are active LOW. For 80186/80188 system, the $\overline{RQ}/\overline{GT}1$ line may be connected to the 82188 Integrated Bus Controller. In this case, a third processor with a HOLD, HLDA bus exchange system may acquire the bus from the 8087. For this configuration, $\overline{RQ}/\overline{GT}1$ will only be used if the 8087 is the bus master. Refer to 82188 data sheet for further information.
QS1, QS0	I	**QS1, QS0:** QS1 and QS0 provide the 8087 with status to allow tracking of the CPU instruction queue. **QS1 QS0** 0 (LOW) 0 No Operation 0 1 First Byte of Op Code from Queue 1 (HIGH) 0 Empty the Queue 1 1 Subsequent Byte from Queue
INT	O	**Interrupt:** This line is used to indicate that an unmasked exception has occurred during numeric instruction execution when 8087 interrupts are enabled. This signal is typically routed to an 8259A for 8086 systems and to INT0 for 80186/80188 systems. INT is active HIGH.
BUSY	O	**Busy:** This signal indicates that the 8087 NEU is executing a numeric instruction. It is connected to the CPU's $\overline{TEST}$ pin to provide synchronization. In the case of an unmasked exception BUSY remains active until the exception is cleared. BUSY is active HIGH.
READY	I	**Ready:** READY is the acknowledgement from the addressed memory device that it will complete the data transfer. The RDY signal from memory is synchronized by the 8284A Clock Generator to form READY for 8086 systems. For 80186/80188 systems, RDY is synchronized by the 82188 Integrated Bus Controller to form READY. This signal is active HIGH.
RESET	I	**Reset:** RESET causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. RESET is internally synchronized.
CLK	I	**Clock:** The clock provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.
V_{CC}		**Power:** V_{CC} is the +5V power supply pin.
GND		**Ground:** GND are the ground pins.

NOTE:
For the pin descriptions of the 8086, 8088, 80186 and 80188 CPUs, reference the respective data sheets (8086, 8088, 80186, 80188).

APPLICATION AREAS

The 8087 provides functions meant specifically for high performance numeric processing requirements. Trigonometric, logarithmic, and exponential functions are built into the coprocessor hardware. These functions are essential in scientific, engineering, navigational, or military applications.

The 8087 also has capabilities meant for business or commercial computing. An 8087 can process Binary Coded Decimal (BCD) numbers up to 18 digits without roundoff errors. It can also perform arithmetic on integers as large as 64 bits $\pm 10^{18}$).

PROGRAMMING LANGUAGE SUPPORT

Programs for the 8087 can be written in Intel's high-level languages for 8086/8088 and 80186/80188 Systems; ASM-86 (the 8086, 8088 assembly language), PL/M-86, FORTRAN-86, and PASCAL-86.

RELATED INFORMATION

For 8086, 8088, 80186 or 80188 details, refer to the respective data sheets, For 80186 or 80188 systems, also refer to the 82188 Integrated Bus Controller data sheet.

FUNCTIONAL DESCRIPTION

The 8087 Numeric Data Processor's architecture is designed for high performance numeric computing in conjunction with general purpose processing.

The 8087 is a numeric processor extension that provides arithmetic and logical instruction support for a variety of numeric data types. It also executes numerous built-in transcendental functions (e.g., tangent and log functions). The 8087 executes instructions as a coprocessor to a maximum mode CPU. It effectively extends the register and instruction set of the system and adds several new data types as well. Figure 3 presents the registers of the CPU+8087. Table 2 shows the range of data types supported by the 8087. The 8087 is treated as an extension to the CPU, providing register, data types, control, and instruction capabilities at the hardware level. At the programmers level the CPU and the 8087 are viewed as a single unified processor.

System Configuration

As a coprocessor to an 8086 or 8088, the 8087 is wired in parallel with the CPU as shown in Figure 4. Figure 5 shows the 80186/80188 system configuration. The CPU's status (S0–S2) and queue status lines (QS0–QS1) enable the 8087 to monitor and decode instructions in synchronization with the CPU and without any CPU overhead. For 80186/80188 systems, the queue status signals of the 80186/80188 are synchronized to 8087 requirements by the 8288 Integrated Bus Controller. Once started, the 8087 can process in parallel with, and independent of, the host CPU. For resynchronization, the 8087's BUSY signal informs the CPU that the 8087 is executing an instruction and the CPU WAIT instruction tests this signal to insure that the 8087 is ready to execute subsequent instructions. The 8087 can interrupt the CPU when it detects an error or

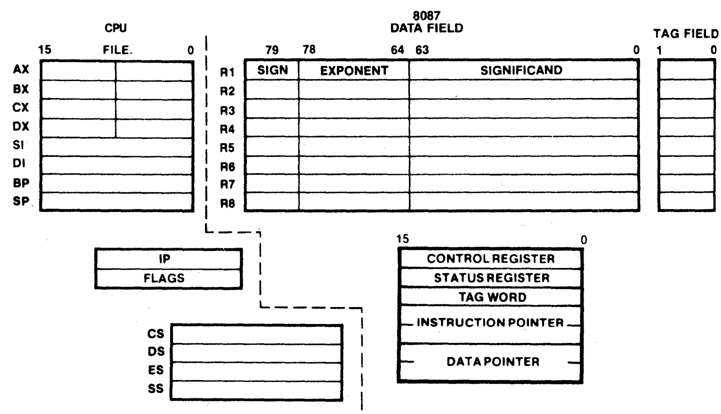


Figure 3. CPU+8087 Architecture

exception. The 8087's interrupt request line is typically routed to the CPU through an 8259A Programmable Interrupt Controller for 8086, 8088 systems and INTO for 80186/80188.

The 8087 uses one of the request/grant lines of the 8086/8088 architecture (typically $\overline{RQ}/\overline{GT}0$) to obtain control of the local bus for data transfers. The other request/grant line is available for general system use (for instance by an I/O processor in LOCAL mode). A bus master can also be connected to the 8087's $\overline{RQ}/\overline{GT}1$ line. In this configuration the 8087 will pass the request/grant handshake signals between the CPU and the attached master when the 8087 is not in control of the bus and will relinquish the bus to the master directly when the 8087 is in control. In this way two additional masters can be configured in an 8086/8088 system; one will share the 8086 bus with the 8087 on a first come first served basis, and the second will be guaranteed to be higher in priority than the 8087.

For 80186/80188 systems, $\overline{RQ}/\overline{GT}0$ and $\overline{RQ}/\overline{GT}1$ are connected to the corresponding inputs of the 82188 Integrated Bus Controller. Because the 80186/80188 has a HOLD, HLDA bus exchange protocol, an interface is needed which will translate $\overline{RQ}/\overline{GT}$ signals to corresponding HOLD, HDLA signals and vice versa. One of the functions of the 82188 IBC is to provide this translation. $\overline{RQ}/\overline{GT}0$ is translated to HOLD, HLDA signals which are then directly connected to the 80186/80188. The $\overline{RQ}/\overline{GT}1$ line is also translated into HOLD, HLDA signals (referred to as SYSHOLD, SYSHLDA signals) by the 82188 IBC. This allows a third processor (using a HOLD, HLDA bus exchange protocol) to gain control of the bus.

Unlike an 8086/8087 system, $\overline{RQ}/\overline{GT}$ is only used when the 8087 has bus control. If the third processor requests the bus when the current bus master is the 80186/80188, the 82188 IBC will directly pass the request onto the 80186/80188 without going through the 8087. The third processor has the highest bus priority in the system. If the 8087 requests the bus while the third processor has bus control, the grant pulse will not be issued until the third processor releases the bus (using SYSHOLD). In this configuration, the third processor has the highest priority, the 8087 has the next highest, and the 80186/80188 has the lowest bus priority.

Table 2. 8087 Data Types

Data Formats	Range	Precision	Most Significant Byte 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0
Word Integer	10^4	16 Bits	I_{15} … I_0 Two's Complement
Short Integer	10^9	32 Bits	I_{31} … I_0 Two's Complement
Long Integer	10^{18}	64 Bits	I_{63} … I_0 Two's Complement
Packed BCD	10^{18}	18 Digits	S | — $D_{17}D_{16}$ … D_1 D_0
Short Real	$10^{\pm 38}$	24 Bits	S | E_7 … E_0 | F_1 … F_{23} F_0 Implicit
Long Real	$10^{\pm 308}$	53 Bits	S | E_{10} … E_0 | F_1 … F_{52} F_0 Implicit
Temporary Real	$10^{\pm 4932}$	64 Bits	S | E_{14} … E_0 | F_0 … F_{63}

Integer: I

Packed BCD: $(-1)^S(D_{17} \ldots D_0)$

Real: $(-1)^S(2^{E-BIAS})(F_0 \cdot F_1 \ldots)$

Bias = 127 for Short Real
1023 for Long Real
16383 for Temp Real

Bus Operation

The 8087 bus structure, operation and timing are identical to all other processors in the 8086/8088 series (maximum mode configuration). The address is time multiplexed with the data on the first 16/8 lines of the address/data bus. A16 through A19 are time multiplexed with four status lines S3-S6. S3, S4 and S6 are always one (HIGH) for 8087-driven bus cycles while S5 is always zero (LOW). When the 8087 is monitoring CPU bus cycles (passive mode) S6 is also monitored by the 8087 to differentiate 8086/8088 activity from that of a local I/O processor or any other local bus master. (The 8086/8088 must be the only processor on the local bus to drive S6 LOW). S7 is multiplexed with and has the same value as $\overline{BHE}$ for all 8087 bus cycles.

The first three status lines, $\overline{S0}$-$\overline{S2}$, are used with an 8288 bus controller or 82188 Integrated Bus Controller to determine the type of bus cycle being run:

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	
0	X	X	Unused
1	0	0	Unused
1	0	1	Memory Data Read
1	1	0	Memory Data Write
1	1	1	Passive (no bus cycle)

Programming Interface

The 8087 includes the standard 8086, 8088 instruction set for general data manipulation and program control. It also includes 68 numeric instructions for extended precision integer, floating point, trigonometric, logarithmic, and exponential functions. Sample execution times for several 8087 functions are shown in Table 3. Overall performance is up to 100 times that of an 8086 processor for numeric instructions.

Any instruction executed by the 8087 is the combined result of the CPU and 8087 activity. The CPU and the 8087 have specialized functions and registers providing fast concurrent operation. The CPU controls overall program execution while the 8087 uses the coprocessor interface to recognize and perform numeric operations.

Table 2 lists the seven data types the 8087 supports and presents the format for each type. Internally, the 8087 holds all numbers in the temporary real format. Load and store instructions automatically convert operands represented in memory as 16-, 32-, or 64-bit integers, 32- or 64-bit floating point numbers or 18-digit packed BCD numbers into temporary real format and vice versa. The 8087 also provides the capability to control round off, underflow, and overflow errors in each calculation.

Computations in the 8087 use the processor's register stack. These eight 80-bit registers provide the equivalent capacity of 20 32-bit registers. The 8087 register set can be accessed as a stack, with instructions operating on the top one or two stack elements, or as a fixed register set, with instructions operating on explicitly designated registers.

Table 5 lists the 8087's instructions by class. All appear as ESCAPE instructions to the host. Assembly language programs are written in ASM-86, the 8086, 8088 assembly language.

Table 3. Execution Times for Selected 8086/8087 Numeric Instructions and Corresponding 8086 Emulation

Floating Point Instruction	Approximate Execution Time (μs)	
	8086/8087 (8 MHz (Clock)	8086 Emulation
Add/Subtract	10.6	1000
Multiply (single precision)	11.9	1000
Multiply (extended precision)	16.9	1312.5
Divide	24.4	2000
Compare	5.6	812.5
Load (double precision)	6.3	1062.5
Store (double precision)	13.1	750
Square Root	22.5	12250
Tangent	56.3	8125
Exponentiation	62.5	10687.5

NUMERIC PROCESSOR EXTENSION ARCHITECTURE

As Shown in Figure 1, the 8087 is internally divided into two processing elements, the control unit (CU) and the numeric execution unit (NEU). The NEU executes all numeric instructions, while the CU receives and decodes instructions, reads and writes memory operands and executes 8087 control instructions. The two elements are able to operate independently of one another, allowing the CU to maintain synchronization with the CPU while the NEU is busy processing a numeric instruction.

Control Unit

The CU keeps the 8087 operating in synchronization with its host CPU. 8087 instructions are intermixed with CPU instructions in a single instruction stream. The CPU fetches all instructions from memory; by monitoring the status ($\overline{S0}$-$\overline{S2}$, S6) emitted by the CPU, the control unit determines when an instruction is being fetched. The

Figure 4. 8086/8087, 8088/8087 System Configuration

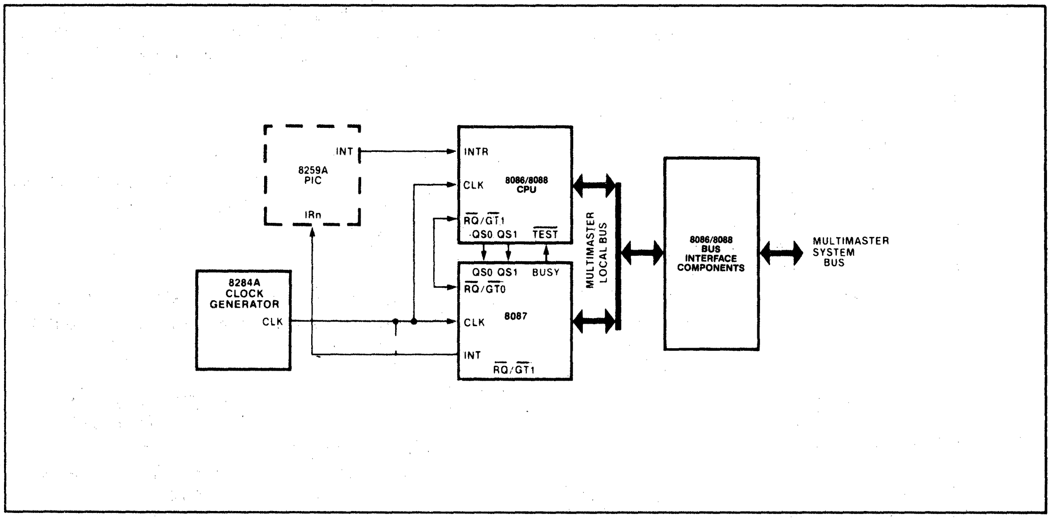
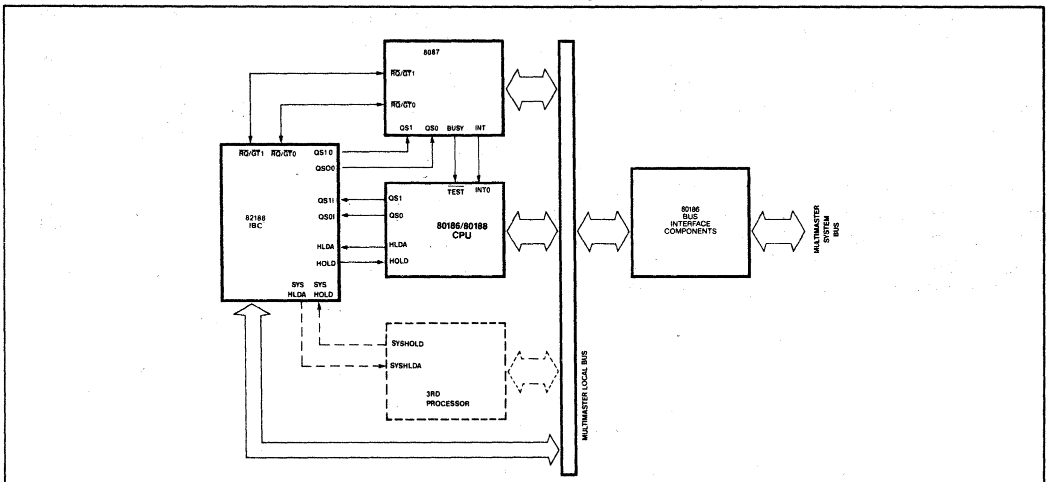


Figure 5. 80186/8087, 80188/8087 System Configuration



CU monitors the data bus in parallel with the CPU to obtain instructions that pertain to the 8087.

The CU maintains an instruction queue that is identical to the queue in the host CPU. The CU automatically determines if the CPU is an 8086/186 or an 8088/188 immediately after reset (by monitoring the $\overline{BHE}$/S7 line) and matches its queue length accordingly. By monitoring the CPU's queue status lines (QS0, QS1), the CU obtains and decodes instructions from the queue in synchronization with the CPU.

A numeric instruction appears as an ESCAPE instruction to the CPU. Both the CPU and 8087 decode and execute the ESCAPE instruction together. The 8087 only recognizes the numeric instructions shown in Table 5. The start of a numeric operation is acomplished when the CPU executes the ESCAPE instruction. The instruction may or may not identify a memory operand.

The CPU does, however, distinguish between ESC instructions that reference memory and those that do not. If the instruction refers to a memory operand, the CPU calculates the operand's address using any one of its available addressing modes, and then performs a "dummy read" of the word at that location. (Any location within the 1M byte address space is allowed.) This is a normal read cycle except that the CPU ignores the data it receives. If the ESC instruction does not contain a memory reference (e.g. an 8087 stack operation), the CPU simply proceeds to the next instruction.

An 8087 instruction can have one of three memory reference options; (1) not reference memory; (2) load an operand word from memory into the 8087; or (3) store an operand word from the 8087 into memory. If no memory reference is required, the 8087 simply executes its instruction. If a memory reference is required, the CU uses a "dummy read" cycle initiated by the CPU to capture and save the address that the CPU places on the bus. If the instruction is a load, the CU additionally captures the data word when it becomes available on the local data bus. If data required is longer than one word, the CU immediately obtains the bus from the CPU using the request/grant protocol and reads the rest of the information in consecutive bus cycles. In a store operation, the CU captures and saves the store address as in a load, and ignores the data word that follows in the "dummy read" cycle. When the 8087 is ready to perform the store, the CU obtains the bus from the CPU and writes the operand starting at the specified address.

Numeric Execution Unit

The NEU executes all instructions that involve the register stack; these include arithmetic, logical, transcendental, constant and data transfer instructions. The data path in the NEU is 84 bits wide (68 fraction bits, 15 exponent bits and a sign bit) which allows internal operand transfers to be performed at very high speeds.

When the NEU begins executing an instruction, it activates the 8087 BUSY signal. This signal can be used in conjunction with the CPU WAIT instruction to resynchronize both processors when the NEU has completed its current instruction.

Register Set

The CPU+8087 register set is shown in Figure 3. Each of the eight data registers in the 8087's register stack is 80 bits and is divided into "fields" corresponding to the 8087's temporary real data type.

At a given point in time the TOP field in the control word identifies the current top-of-stack register. A "push" operation decrements TOP by 1 and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments TOP by 1. Like CPU stacks in memory, the 8087 register stack grows "down" toward lower-addressed registers.

Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the top of the stack. These instructions implicitly address the register pointed to by the TOP. Other instructions allow the programmer to explicitly specify the register which is to be used. Explicit register addressing is "top-relative."

Status Word

The status word shown in Figure 6 reflects the overall state of the 8087; it may be stored in memory and then inspected by CPU code. The status word is a 16-bit register divided into fields as shown in Figure 6. The busy bit (bit 15) indicates whether the NEU is either executing an instruction or has an interrupt request pending (B = 1), or is idle (B = 0). Several instructions which store and manipulate the status word are executed exclusively by the CU, and these do not set the busy bit themselves.

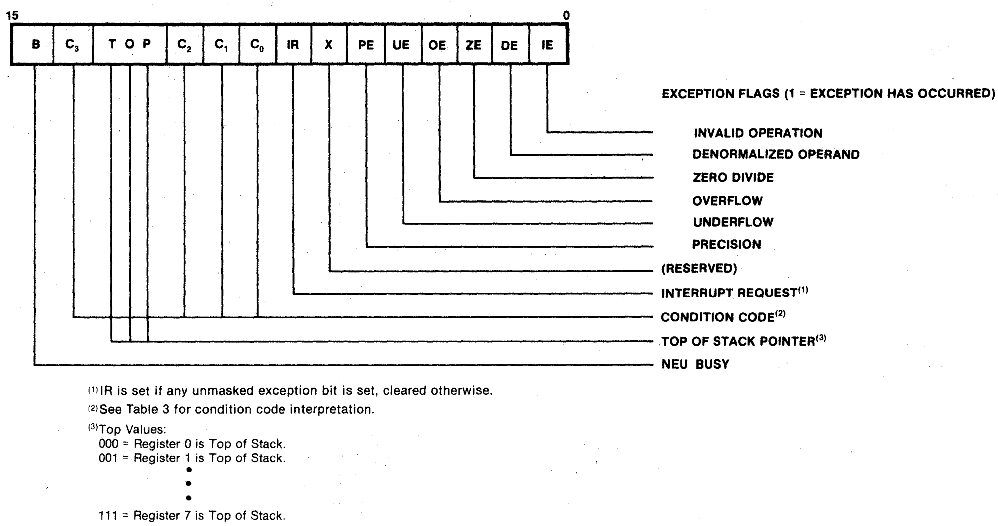


Figure 6. 8087 Status Word

The four numeric condition code bits (C_0-C_3) are similar to flags in a CPU: various instructions update these bits to reflect the outcome of 8087 operations. The effect of these instructions on the condition code bits is summarized in Table 4.

Bits 14–12 of the status word point to the 8087 register that is the current top-of-stack (TOP) as described above.

Bit 7 is the interrupt request bit. This bit is set if any unmasked exception bit is set and cleared otherwise.

Bits 5–0 are set to indicate that the NEU has detected an exception while executing an instruction.

Tag Word

The tag word marks the content of each register as shown in Figure 7. The principal function of the tag word is to optimize the 8087's performance. The tag word can be used, however, to interpret the contents of 8087 registers.

Instruction and Data Pointers

The instruction and data pointers (see Figure 8) are provided for user-written error handlers. Whenever the 8087 executes an NEU instruction, the CU saves the instruction address, the operand address (if present) and the instruction opcode. 8087 instructions can store this data into memory.

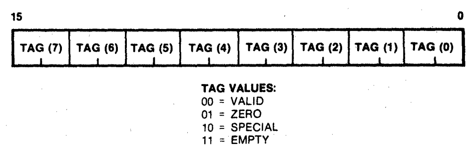


Figure 7. 8087 Tag Word

Table 4a. Condition Code Interpretation

Instruction Type	C_3	C_2	C_1	C_0	Interpretation
Compare, Test	0	0	X	0	ST > Source or 0 (FTST)
	0	0	X	1	ST < Source or 0 (FTST)
	1	0	X	0	ST = Source or 0 (FTST)
	1	1	X	1	ST is not comparable
Remainder	Q_1	0	Q_0	Q_2	Complete reduction with three low bits of quotient (See Table 4b)
	U	1	U	U	Incomplete Reduction
Examine	0	0	0	0	Valid, positive unnormalized
	0	0	0	1	Invalid, positive, exponent =0
	0	0	1	0	Valid, negative, unnormalized
	0	0	1	1	Invalid, negative, exponent =0
	0	1	0	0	Valid, positive, normalized
	0	1	0	1	Infinity, positive
	0	1	1	0	Valid, negative, normalized
	0	1	1	1	Infinity, negative
	1	0	0	0	Zero, positive
	1	0	0	1	Empty
	1	0	1	0	Zero, negative
	1	0	1	1	Empty
	1	1	0	0	Invalid, positive, exponent = 0
	1	1	0	1	Empty
	1	1	1	0	Invalid, negative, exponent = 0
	1	1	1	1	Empty

NOTES:
1. ST = Top of stack
2. X = value is not affected by instruction
3. U = value is undefined following instruction
4. Q_n = Quotient bit n

Table 4b. Condition Code Interpretation after FPREM Instruction As a Function of Dividend Value

Dividend Range	Q_2	Q_1	Q_0
Dividend < 2 * Modulus	C_3[1]	C_1[1]	Q_0
Dividend < 4 * Modulus	C_3[1]	Q_1	Q_0
Dividend ≥ 4 * Modulus	Q_2	Q_1	Q_0

NOTE:
1. Previous value of indicated bit, not affected by FPREM instruction execution.

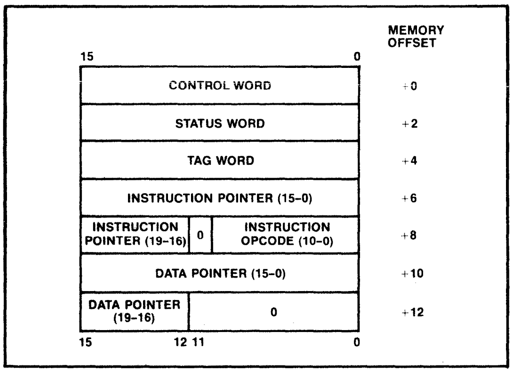


Figure 8. 8087 Instruction and Data Pointer Image in Memory

Control Word

The 8087 provides several processing options which are selected by loading a word from memory into the control word. Figure 9 shows the format and encoding of the fields in the control word.

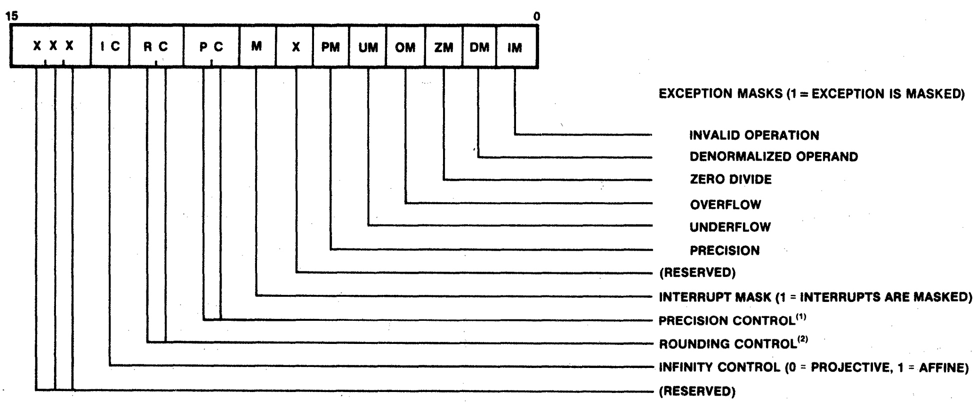
The low order byte of this control word configures 8087 interrupts and exception masking. Bits 5–0 of the control word contain individual masks for each of the six exceptions that the 8087 recognizes and bit 7 contains a general mask bit for all 8087 interrupts. The high order byte of the control word configures the 8087 operating mode including precision, rounding, and infinity controls. The precision control bits (bits 9–8) can be used to set the 8087 internal operating precision at less than the default of temporary real precision. This can be useful in providing compatibility with earlier generation arithmetic processors of smaller precision than the 8087. The rounding control bits (bits 11–10) provide for directed rounding and true chop as well as the unbiased round to nearest mode specified in the proposed IEEE standard. Control over closure of the number space at infinity is also provided (either affine closure, ±∞, or projective closure, ∞, is treated as unsigned, may be specified).

Exception Handling

The 8087 detects six different exception conditions that can occur during instruction execution. Any or all exceptions will cause an interrupt if unmasked and interrupts are enabled.

If interrupts are disabled the 8087 will simply continue execution regardless of whether the host clears the exception. If a specific exception class is masked and that exception occurs, however, the 8087 will post the exception in the status register and perform an on-chip default exception handling procedure, thereby allowing processing to continue. The exceptions that the 8087 detects are the following:

1. INVALID OPERATION: Stack overflow, stack underflow, indeterminate form (0/0, ∞− ∞, etc.) or the use of a Non-Number (NAN) as an operand. An exponent value is reserved and any bit pattern with this value in the exponent field is termed a Non-Number and causes this exception. If this exception is masked, the 8087's default response is to generate a specific NAN called INDEFINITE, or to propagate already existing NANs as the calculation result.



(1)Precision Control
00 = 24 bits
01 = Reserved
10 = 53 bits
11 = 64 bits

(2)Rounding Control
00 = Round to Nearest or Even
01 = Round Down (toward − ∞)
10 = Round Up (toward + ∞)
11 = Chop (truncate toward zero)

Figure 9. 8087 Control Word

2. OVERFLOW: The result is too large in magnitude to fit the specified format. The 8087 will generate an encoding for infinity if this exception is masked.

3. ZERO DIVISOR: The divisor is zero while the dividend is a non-infinite, non-zero number. Again, the 8087 will generate an encoding for infinity if this exception is masked.

4. UNDERFLOW: The result is non-zero but too small in magnitude to fit in the specified format. If this exception is masked the 8087 will denormalize (shift right) the fraction until the exponent is in range. This process is called gradual underflow.

5. DENORMALIZED OPERAND: At least one of the operands or the result is denormalized; it has the smallest exponent but a non-zero significand. Normal processing continues if this exception is masked off.

6. INEXACT RESULT: If the true result is not exactly representable in the specified format, the result is rounded according to the rounding mode, and this flag is set. If this exception is masked, processing will simply continue.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ....0°C to 70°C
Storage Temperature ....−65°C to +150°C
Voltage on Any Pin with
Respect to Ground ....−1.0V to +7V
Power Dissipation ....3.0 Watt

**NOTICE: Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5V ±5%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} +0.5	V	
V_{OL}	Output Low Voltage (See Note 8)		0.45	V	I_{OL} = 2.5 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{CC}	Power Supply Current		475	mA	T_A = 25°C
I_{LI}	Input Leakage Current		±10	μA	0V ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}
V_{CL}	Clock Input Low Voltage	−0.5	+0.6	V	
V_{CH}	Clock Input High Voltage	3.9	V_{CC} + 1.0	V	
C_{IN}	Capacitance of Inputs		10	pF	fc = 1 MHz
C_{IO}	Capacitance of I/O Buffer (AD0-15, A_{16}–A_{19}, BHE, S2-S0, RQ/GT) and CLK		15	pF	fc = 1 MHz
C_{OUT}	Capacitance of Outputs BUSY, INT		10	pF	fc = 1 MHz

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5V ±5%)

TIMING REQUIREMENTS		8087		8087-2		8087-1 (Preliminary: See Note 7)			
Symbol	Parameter	Min.	Max.	Min.	Max.	Min.	Max.	Units	Test Conditions
TCLCL	CLK Cycle Period	200	500	125	500	100	500	ns	
TCLCH	CLK Low Time	118		68		53		ns	
TCHCL	CLK High Time	69		44		39		ns	
TCH1CH2	CLK Rise Time		10		10		15	ns	From 1.0V to 3.5V
TCL2CL2	CLK Fall Time		10		10		15	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		20		15		ns	
TCLDX	Data in Hold Time	10		10		10		ns	
TRYHCH	READY Setup Time	118		68		53		ns	
TCHRYX	READY Hold Time	30		20		5		ns	
TRYLCL	READY Inactive to CLK**	- 8		- 8		-10		ns	
TGVCH	RQ/GT Setup Time(See Note 8)	30		15		15		ns	
TCHGX	RQ/GT Hold Time	40		30		20		ns	
TQVCL	QS0-1 Setup Time (See Note 8)	30		30		30		ns	
TCLQX	QS0-1 Hold Time	10		10		5		ns	
TSACH	Status Active Setup Time	30		30		30		ns	
TSNCL	Status Inactive Setup Time	30		30		30		ns	
TILIH	Input Rise Time (Except CLK)		20		20		20	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		12		12		15	ns	From 2.0V to 0.8V

**See Note 6

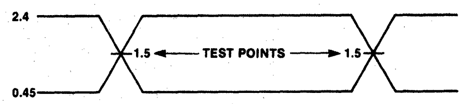
A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES

		8087		8087-2		8087-1 (Preliminary: See Note 7)			
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
TCLML	Command Active Delay (See Notes 1,2)	10/0	35/70	10/0	35/70	10/0	35/70	ns	C_L = 20 - 100pF for all 8087 Outputs (in addition to 8087 self-load)
TCLMH	Command Inactive Delay (See Notes 1,2)	10/0	35/55	10/0	35/55	10/0	35/70	ns	
TRYHSH	Ready Active to Status Passive (See Note 5)		110		65		45	ns	
TCHSV	Status Active Delay	10	110	10	60	10	45	ns	
TCLSH	Status Inactive Delay	10	130	10	70	10	55	ns	
TCLAV	Address Valid Delay	10	110	10	60	10	55	ns	
TCLAX	Address Hold Time	10		10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	TCLAX	50	TCLAX	45	ns	
TSVLH	Status Valid to ALE High (See Notes 1,2)		15/30		15/30		15/30	ns	
TCLLH	CLK Low to ALE Valid (See Notes 1,2)		15/30		15/30		15/30	ns	
TCHLL	ALE Inactive Delay (See Notes 1,2)		15/30		15/30		15/30	ns	
TCLDV	Data Valid Delay	10	110	10	60	10	50	ns	
TCHDX	Data Hold Time	10		10		10	45	ns	
TCVNV	Control Active Delay (See Notes 1,3)	5	45	5	45	5	45	ns	
TCVNX	Control Inactive Delay (See Notes 1,3)	10	45	10	45	10	45	ns	
TCHBV	BUSY and INT Valid Delay	10	150	10	85	10	65	ns	
TCHDTL	Direction Control Active Delay (See Notes 1,3)		50		50		50	ns	
TCHDTH	Direction Control Inactive Delay (See Notes 1,3)		30		30		30	ns	
TSVDTV	STATUS to DT/$\overline{R}$ Delay (See Notes 1,4)	0	30	0	30	0	30	ns	
TCLDTV	DT/$\overline{R}$ Active Delay (See Notes 1,4)	0	55	0	55	0	55	ns	
TCHDNV	$\overline{DEN}$ Active Delay (See Notes 1,4)	0	55	0	55	0	55	ns	
TCHDNX	$\overline{DEN}$ Inactive Delay (See Notes 1,4)	5	55	5	55	5	55	ns	
TCLGL	RQ/GT Active Delay (see Note 8)	0	85	0	50	0	38	ns	C_L=40pF (in addition to 8087 self-load)
TCLGH	RQ/GT Inactive Delay	0	85	0	50	0	45	ns	
TOLOH	Output Rise Time		20		20		15	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		12		12		12	ns	From 2.0V to 0.8V

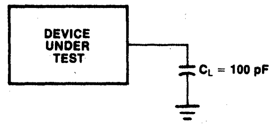
NOTES:
1. Signal at 8284A, 8288, or 82188 shown for reference only.
2. 8288 timing/82188 timing
3. 8288 timing
4. 82188 timing
5. Applies only to T_3 and wait states
6. Applies only to T_2 state (8ns into T_3)
7. IMPORTANT SYSTEM CONSIDERATION: Some 8087-1 timing parameters are constrained relative to the corresponding 8086-1 specifications. Therefore, 8086-1 systems incorporating the 8087-1 should be designed with the 8087-1 specifications.
8. Changes since last revision.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0."

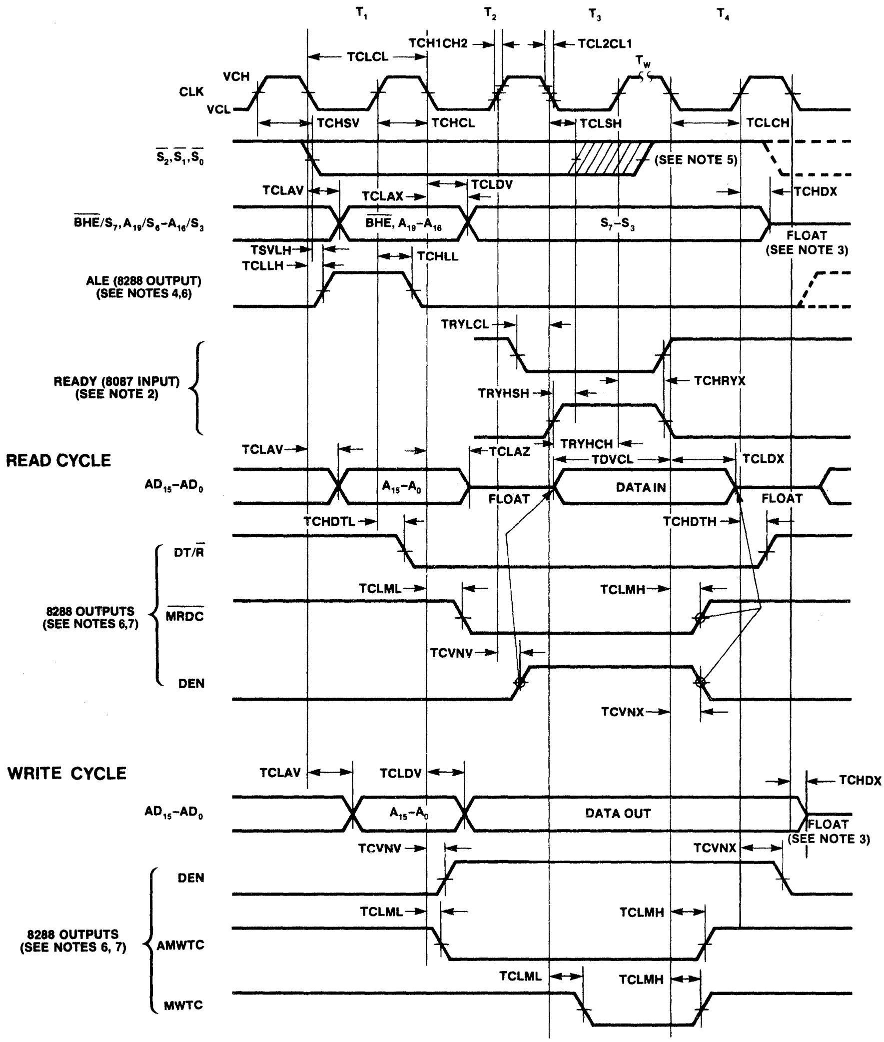
A.C. TESTING LOAD CIRCUIT



C_L INCLUDES JIG CAPACITANCE

WAVEFORMS

MASTER MODE (with 8288 references)

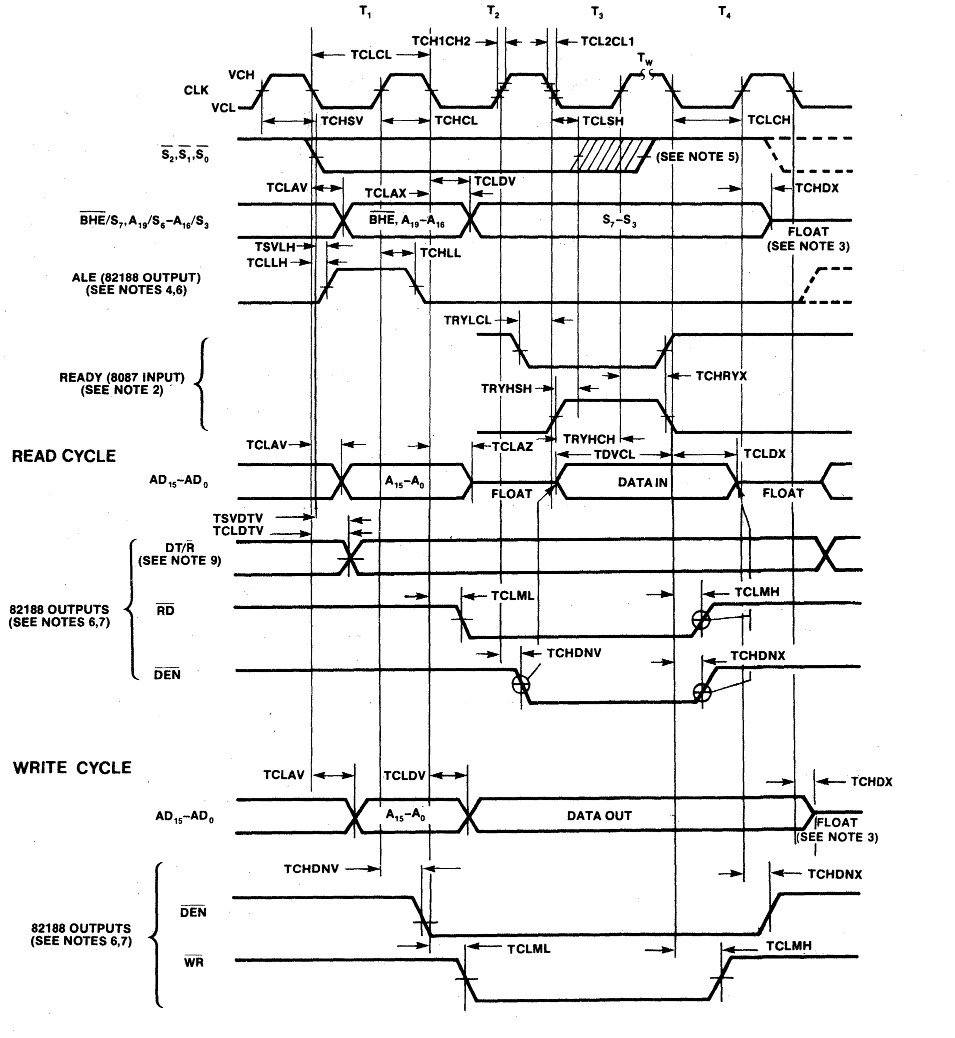


NOTES:

1. ALL SIGNALS SWITCH BETWEEN V_{OL} AND V_{OH} UNLESS OTHERWISE SPECIFIED.
2. READY IS SAMPLED NEAR THE END OF T_2, T_3 AND T_W TO DETERMINE IF T_W MACHINE STATES ARE TO BE INSERTED.
3. THE LOCAL BUS FLOATS ONLY IF THE 8087 IS RETURNING CONTROL TO THE 8086/8088.
4. ALE RISES AT LATER OF (TSVLH, TCLLH).
5. STATUS INACTIVE IN STATE JUST PRIOR TO T_4.
6. SIGNALS AT 8284A OR 8288 ARE SHOWN FOR REFERENCE ONLY.
7. THE ISSUANCE OF 8288 COMMAND AND CONTROL SIGNALS ($\overline{MRDC}$, $\overline{MWTC}$, $\overline{AMWC}$ AND DEN) LAGS THE ACTIVE HIGH 8288 CEN.
8. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS OTHERWISE NOTED.

WAVEFORMS (Continued)

MASTER MODE (with 82188 references)

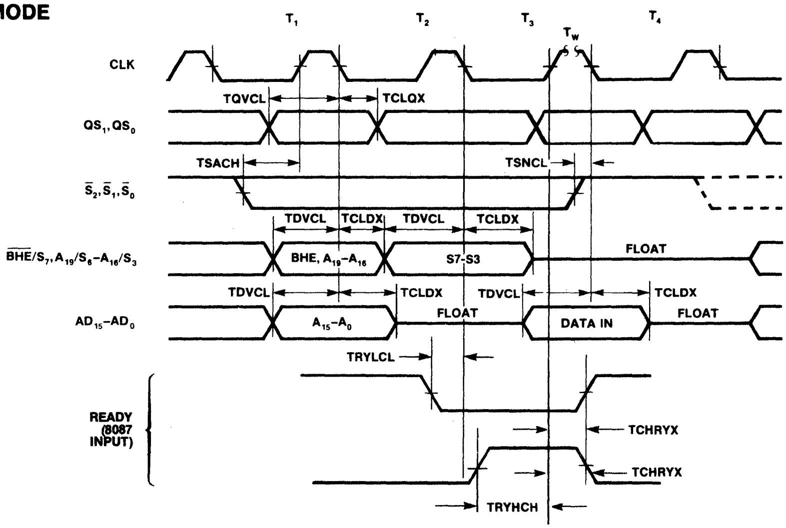


NOTES:

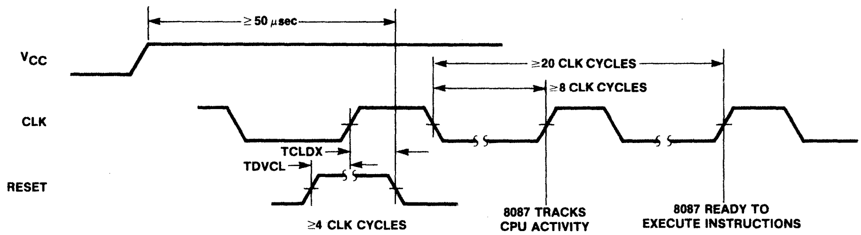
1. ALL SIGNALS SWITCH BETWEEN V_{OL} AND V_{OH} UNLESS OTHERWISE SPECIFIED.
2. READY IS SAMPLED NEAR THE END OF T_2, T_3 AND T_W TO DETERMINE IF T_W MACHINE STATES ARE TO BE INSERTED.
3. THE LOCAL BUS FLOATS ONLY IF THE 8087 IS RETURNING CONTROL TO THE 80186/80188
4. ALE RISES AT LATER OF (TSVLH, TCLLH).
5. STATUS INACTIVE IN STATE JUST PRIOR TO T_4.
6. SIGNALS AT 8284A OR 82188 ARE SHOWN FOR REFERENCE ONLY.
7. THE ISSUANCE OF 8288 COMMAND AND CONTROL SIGNALS (MRDC, MWTC, AMWC, AND DEN) LAGS THE ACTIVE HIGH 8288 CEN.
8. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS OTHERWISE NOTED.
9. DT/R BECOMES VALID AT THE LATER OF (TSVDTV, TCLDTV).

WAVEFORMS (Continued)

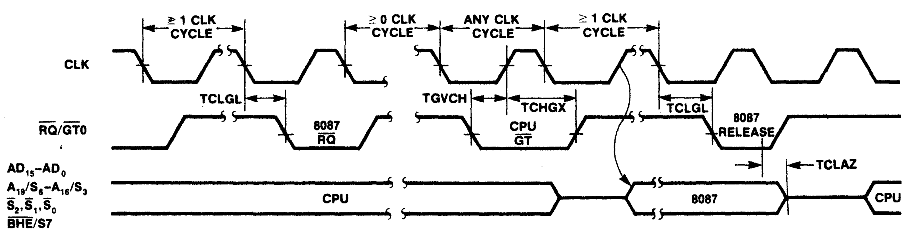
PASSIVE MODE



RESET TIMING



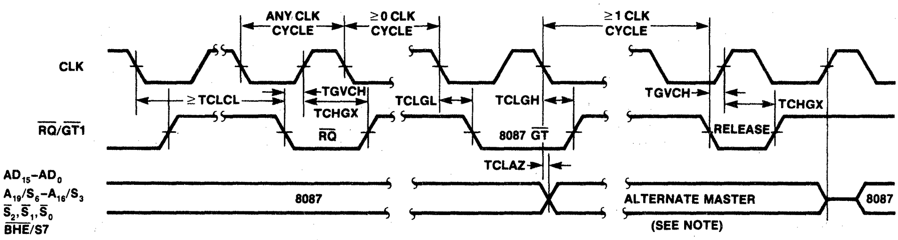
REQUEST/GRANT$_0$ TIMING



NOTE: THE CPU PROVIDES ACTIVE PULLUP OF RQ/GT0, SEE TCLGH SPEC.

WAVEFORMS (Continued)

REQUEST/GRANT$_1$ TIMING



NOTE: ALTERNATE MASTER MAY NOT DRIVE THE BUSES OUTSIDE OF THE REGION SHOWN WITHOUT RISKING BUS CONTENTION.

BUSY AND INTERRUPT TIMING

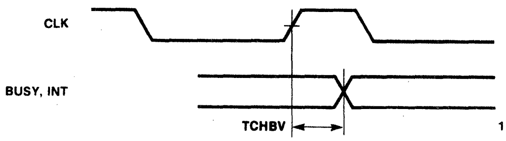


Table 5. 8087 Extensions to the 86/186 Instructions Sets

Data Transfer			Optional 8,16 Bit Displacement	Clock Count Range 32 Bit Real	32 Bit Integer	64 Bit Real	16 Bit Integer
FLD = LOAD	MF =			00	01	10	11
Integer/Real Memory to ST(0)	ESCAPE MF 1	MOD 0 0 0 R/M	DISP	38–56 + EA	52–60 + EA	40–60 + EA	46–54 + EA
Long Integer Memory to ST(0)	ESCAPE 1 1 1	MOD 1 0 1 R/M	DISP	60–68 + EA			
Temporary Real Memory to ST(0)	ESCAPE 0 1 1	MOD 1 0 1 R/M	DISP	53–65 + EA			
BCD Memory to ST(0)	ESCAPE 1 1 1	MOD 1 0 0 R/M	DISP	290–310 + EA			
ST(i) to ST(0)	ESCAPE 0 0 1	1 1 0 0 0 ST(i)		17–22			
FST = STORE							
ST(0) to Integer/Real Memory	ESCAPE MF 1	MOD 0 1 0 R/M	DISP	84–90 + EA	82–92 + EA	96–104 + EA	80–90 + EA
ST(0) to ST(i)	ESCAPE 1 0 1	1 1 0 1 0 ST(i)		15–22			
FSTP = STORE AND POP							
ST(0) to Integer/Real Memory	ESCAPE MF 1	MOD 0 1 1 R/M	DISP	86–92 + EA	84–94 + EA	98–106 + EA	82–92 + EA
ST(0) to Long Integer Memory	ESCAPE 1 1 1	MOD 1 1 1 R/M	DISP	94–105 + EA			
ST(0) to Temporary Real Memory	ESCAPE 0 1 1	MOD 1 1 1 R/M	DISP	52–58 + EA			
ST(0) to BCD Memory	ESCAPE 1 1 1	MOD 1 1 0 R/M	DISP	520–540 + EA			
ST(0) to ST(i)	ESCAPE 1 0 1	1 1 0 1 1 ST(i)		17–24			
FXCH = Exchange ST(i) and ST(0)	ESCAPE 0 0 1	1 1 0 0 1 ST(i)		10–15			
Comparison							
FCOM = Compare							
Integer/Real Memory to ST(0)	ESCAPE MF 0	MOD 0 1 0 R/M	DISP	60–70 + EA	78–91 + EA	65–75 + EA	72–86 + EA
ST(i) to ST (0)	ESCAPE 0 0 0	1 1 0 1 0 ST(i)		40–50			
FCOMP = Compare and Pop							
Integer/Real Memory to ST(0)	ESCAPE MF 0	MOD 0 1 1 R/M	DISP	63–73 + EA	80–93 + EA	67–77 + EA	74–88 + EA
ST(i) to ST(0)	ESCAPE 0 0 0	1 1 0 1 1 ST(i)		45–52			
FCOMPP = Compare ST(1) to ST(0) and Pop Twice	ESCAPE 1 1 0	1 1 0 1 1 0 0 1		45–55			
FTST = Test ST(0)	ESCAPE 0 0 1	1 1 1 0 0 1 0 0		38–48			
FXAM = Examine ST(0)	ESCAPE 0 0 1	1 1 1 0 0 1 0 1		12–23			

Table 5. 8087 Extensions to the 86/186 Instruction Sets (cont.)

Constants			Optional 8,16 Bit Displacement	Clock Count Range 32 Bit Real	32 Bit Integer	64 Bit Real	16 Bit Integer
	MF	=		00	01	10	11
FLDZ = LOAD + 0.0 into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 1 1 0		11–17			
FLD1 = LOAD + 1.0 into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 0 0 0		15–21			
FLDPI = LOAD π into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 0 1 1		16–22			
FLDL2T = LOAD $\log_2 10$ into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 0 0 1		16–22			
FLDL2E = LOAD $\log_2 e$ into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 0 1 0		15–21			
FLDLG2 = LOAD $\log_{10} 2$ into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 1 0 0		18–24			
FLDLN2 = LOAD $\log_e 2$ into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 1 0 1		17–23			
Arithmetic							
FADD = Addition							
Integer/Real Memory with ST(0)	ESCAPE MF 0	MOD 0 0 0 R/M	DISP	90–120 + EA	108–143 + EA	95–125 + EA	102–137 + EA
ST(i) and ST(0)	ESCAPE d P 0	1 1 0 0 0 ST(i)		70–100 (Note 1)			
FSUB = Subtraction							
Integer/Real Memory with ST(0)	ESCAPE MF 0	MOD 1 0 R R/M	DISP	90–120 + EA	108–143 + EA	95–125 + EA	102–137 + EA
ST(i) and ST(0)	ESCAPE d P 0	1 1 1 0 R R/M		70–100 (Note 1)			
FMUL = Multiplication							
Integer/Real Memory with ST(0)	ESCAPE MF 0	MOD 0 0 1 R/M	DISP	110–125 + EA	130–144 + EA	112–168 + EA	124–138 + EA
ST(i) and ST(0)	ESCAPE d P 0	1 1 0 0 1 R/M		90–145 (Note 1)			
FDIV = Division							
Integer/Real Memory with ST(0)	ESCAPE MF 0	MOD 1 1 R R/M	DISP	215–225 + EA	230–243 + EA	220–230 + EA	224–238 + EA
ST(i) and ST(0)	ESCAPE d P 0	1 1 1 1 R R/M		193–203 (Note 1)			
FSQRT = Square Root of ST(0)	ESCAPE 0 0 1	1 1 1 1 1 0 1 0		180–186			
FSCALE = Scale ST(0) by ST(1)	ESCAPE 0 0 1	1 1 1 1 1 1 0 1		32–38			
FPREM = Partial Remainder of ST(0) ÷ ST(1)	ESCAPE 0 0 1	1 1 1 1 1 0 0 0		15–190			
FRNDINT = Round ST(0) to Integer	ESCAPE 0 0 1	1 1 1 1 1 1 0 0		16–50			

NOTE:
1. If P=1 then add 5 clocks.

Table 5. 8087 Extensions to the 86/186 Instructions Sets (cont.)

Instruction	Byte 1	Byte 2	Optional 8,16 Bit Displacement	Clock Count Range
FXTRACT = Extract Components of St(0)	ESCAPE 0 0 1	1 1 1 1 0 1 0 0		27–55
FABS = Absolute Value of ST(0)	ESCAPE 0 0 1	1 1 1 0 0 0 0 1		10–17
FCHS = Change Sign of ST(0)	ESCAPE 0 0 1	1 1 1 0 0 0 0 0		10–17
Transcendental				
FPTAN = Partial Tangent of ST(0)	ESCAPE 0 0 1	1 1 1 1 0 0 1 0		30–540
FPATAN = Partial Arctangent of ST(0) ÷ ST(1)	ESCAPE 0 0 1	1 1 1 1 0 0 1 1		250–800
F2XM1 = $2^{ST(0)} - 1$	ESCAPE 0 0 1	1 1 1 1 0 0 0 0		310–630
FYL2X = ST(1) • Log_2 [ST(0)]	ESCAPE 0 0 1	1 1 1 1 0 0 0 1		900–1100
FYL2XP1 = ST(1) • Log_2 [ST(0) + 1]	ESCAPE 0 0 1	1 1 1 1 1 0 0 1		700–1000
Processor Control				
FINIT = Initialized 8087	ESCAPE 0 1 1	1 1 1 0 0 0 1 1		2–8
FENI = Enable Interrupts	ESCAPE 0 1 1	1 1 1 0 0 0 0 0		2–8
FDISI = Disable Interrupts	ESCAPE 0 1 1	1 1 1 0 0 0 0 1		2–8
FLDCW = Load Control Word	ESCAPE 0 0 1	MOD 1 0 1 R/M	DISP	7–14 + EA
FSTCW = Store Control Word	ESCAPE 0 0 1	MOD 1 1 1 R/M	DISP	12–18 + EA
FSTSW = Store Status Word	ESCAPE 1 0 1	MOD 1 1 1 R/M	DISP	12–18 + EA
FCLEX = Clear Exceptions	ESCAPE 0 1 1	1 1 1 0 0 0 1 0		2–8
FSTENV = Store Environment	ESCAPE 0 0 1	MOD 1 1 0 R/M	DISP	40–50 + EA
FLDENV = Load Environment	ESCAPE 0 0 1	MOD 1 0 0 R/M	DISP	35–45 + EA
FSAVE = Save State	ESCAPE 1 0 1	MOD 1 1 0 R/M	DISP	197 – 207 + EA
FRSTOR = Restore State	ESCAPE 1 0 1	MOD 1 0 0 R/M	DISP	197 – 207 + EA
FINCSTP = Increment Stack Pointer	ESCAPE 0 0 1	1 1 1 1 0 1 1 1		6–12
FDECSTP = Decrement Stack Pointer	ESCAPE 0 0 1	1 1 1 1 0 1 1 0		6–12

Table 5. 8087 Extensions to the 86/186 Instructions Sets (cont.)

			Clock Count Range
FFREE = Free ST(i)	ESCAPE 1 0 1	1 1 0 0 0 ST(i)	9–16
FNOP = No Operation	ESCAPE 0 0 1	1 1 0 1 0 0 0 0	10–16
FWAIT = CPU Wait for 8087	1 0 0 1 1 0 1 1		3+5n*

*n = number of times CPU examines TEST line before 8087 lowers BUSY.

NOTES:

1. if mod=00 then DISP=0*, disp-low and disp-high are absent
   if mod=01 then DISP=disp-low sign-extended to 16-bits, disp-high is absent
   if mod=10 then DISP=disp-high; disp-low
   if mod=11 then r/m is treated as an ST(i) field
2. if r/m=000 then EA=(BX) + (SI) +DISP
   if r/m=001 then EA=(BX) + (DI) +DISP
   if r/m=010 then EA=(BP) + (SI) +DISP
   if r/m=011 then EA=(BP) + (DI) +DISP
   if r/m=100 then EA=(SI) + DISP
   if r/m=101 then EA=(DI) + DISP
   if r/m=110 then EA=(BP) + DISP
   if r/m=111 then EA=(BX) + DISP

   *except if mod=000 and r/m=110 then EA =disp-high; disp-low.
3. MF= Memory Format
   00—32-bit Real
   01—32-bit Integer
   10—64-bit Real
   11—16-bit Integer
4. ST(0)= Current stack top
   ST(i) i^{th} register below stack top
5. d= Destination
   0—Destination is ST(0)
   1—Destination is ST(i)
6. P= Pop
   0—No pop
   1—Pop ST(0)
7. R= Reverse: When d=1 reverse the sense of R
   0—Destination (op) Source
   1—Source (op) Destination
8. For **FSQRT**: $-0 \leq ST(0) \leq +\infty$
   For **FSCALE**: $-2^{15} \leq ST(1) < +2^{15}$ and ST(1) integer
   For **F2XM1**: $0 \leq ST(0) \leq 2^{-1}$
   For **FYL2X**: $0 < ST(0) < \infty$
   $-\infty < ST(1) < +\infty$
   For **FYL2XP1**: $0 \leq |ST(0)| < (2 - \sqrt{2})/2$
   $-\infty < ST(1) < \infty$
   For **FPTAN**: $0 \leq ST(0) \leq \pi/4$
   For **FPATAN**: $0 \leq ST(0) < ST(1) < +\infty$

intel®

8282/8283 OCTAL LATCH

- Address Latch for iAPX 86, 88, 186, 188, MCS-80®, MCS-85®, MCS-48® Famlies
- High Output Drive Capability for Driving System Data Bus
- Fully Parallel 8-Bit Data Register and Buffer
- Transparent during Active Strobe
- 3-State Outputs
- 20-Pin Package with 0.3" Center
- No Output Low Noise when Entering or Leaving High Impedance State
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range

The 8282 and 8283 are 8-bit bipolar latches with 3-state output buffers. They can be used to implement latches, buffers, or multiplexers. The 8283 inverts the input data at its outputs while the 8282 does not. Thus, all of the principal peripheral and input/output functions of a microcomputer system can be implemented with these devices.

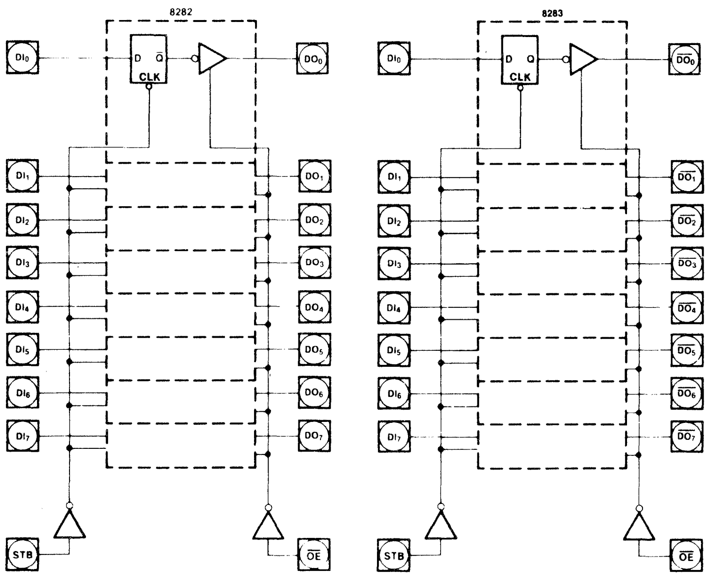


Figure 1. Logic Diagrams

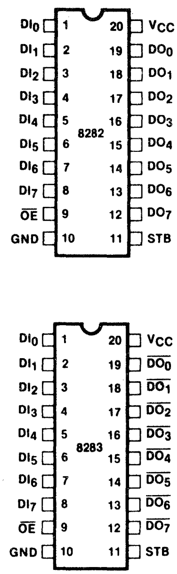


Figure 2. Pin Configurations

Table 1. Pin Description

Pin	Description
STB	STROBE (Input). STB is an input control pulse used to strobe data at the data input pins (A_0-A_7) into the data latches. This signal is active HIGH to admit input data. The data is latched at the HIGH to LOW transition of STB.
$\overline{OE}$	OUTPUT ENABLE (Input). $\overline{OE}$ is an input control signal which when active LOW enables the contents of the data latches onto the data output pin (B_0-B_7). OE being inactive HIGH forces the output buffers to their high impedance state.
DI_0-DI_7	DATA INPUT PINS (Input). Data presented at these pins satisfying setup time requirements when STB is strobed and latched into the data input latches.
DO_0-DO_7 (8282) $\overline{DO_0}$-$\overline{DO_7}$ (8283)	DATA OUTPUT PINS (Output). When $\overline{OE}$ is true, the data in the data latches is presented as inverted (8283) or non-inverted (8282) data onto the data output pins.

FUNCTIONAL DESCRIPTION

The 8282 and 8283 octal latches are 8-bit latches with 3-state output buffers. Data having satisfied the setup time requirements is latched into the data latches by strobing the STB line HIGH to LOW. Holding the STB line in its active HIGH state makes the latches appear transparent. Data is presented to the data output pins by activating the $\overline{OE}$ input line. When $\overline{OE}$ is inactive HIGH the output buffers are in their high impedance state. Enabling or disabling the output buffers will not cause negative-going transients to appear on the data output bus.

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... 0°C to 70°C
Storage Temperature .... −65°C to +150°C
All Output and Supply Voltages .... −0.5V to +7V
All Input Voltages .... −1.0V to +5.5V
Power Dissipation .... 1 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (V_{CC} = 5V ±10%, T_A = 0°C to 70°C)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_C	Input Clamp Voltage		−1	V	I_C = −5 mA
I_{CC}	Power Supply Current		160	mA	
I_F	Forward Input Current		−0.2	mA	V_F = 0.45V
I_R	Reverse Input Current		50	μA	V_R = 5.25V
V_{OL}	Output Low Voltage		.45	V	I_{OL} = 32 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −5 mA
I_{OFF}	Output Off Current		±50	μA	V_{OFF} = 0.45 to 5.25V
V_{IL}	Input Low Voltage		0.8	V	V_{CC} = 5.0V See Note 1
V_{IH}	Input High Voltage	2.0		V	V_{CC} = 5.0V See Note 1
C_{IN}	Input Capacitance		12	pF	F = 1 MHz V_{BIAS} = 2.5V, V_{CC} = 5V T_A = 25°C

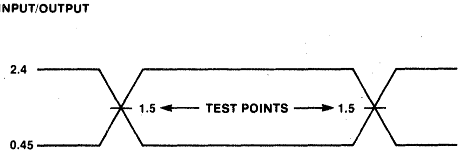
NOTE:
1. Output Loading I_{OL} = 32 mA, I_{OH} = −5 mA, C_L = 300 pF.*

A.C. CHARACTERISTICS (V_{CC} = 5V ±10%, T_A = 0°C to 70°C (See Note 2) Loading: Outputs—I_{OL} = 32 mA, I_{OH} = −5 mA, C_L = 300 pF*)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TIVOV	Input to Output Delay —Inverting —Non-Inverting	 5 5	 22 30	 ns ns	(See Note 1)
TSHOV	STB to Output Delay —Inverting —Non-Inverting	 10 10	 40 45	 ns ns	
TEHOZ	Output Disable Time	5	18	ns	
TELOV	Output Enable Time	10	30	ns	
TIVSL	Input to STB Setup Time	0		ns	
TSLIX	Input to STB Hold Time	25		ns	
TSHSL	STB High Time	15		ns	
TOLOH	Input, Output Rise Time		20	ns	From 0.8V to 2.0V
TOHOL	Input, Output Fall Time		12	ns	From 2.0V to 0.8V

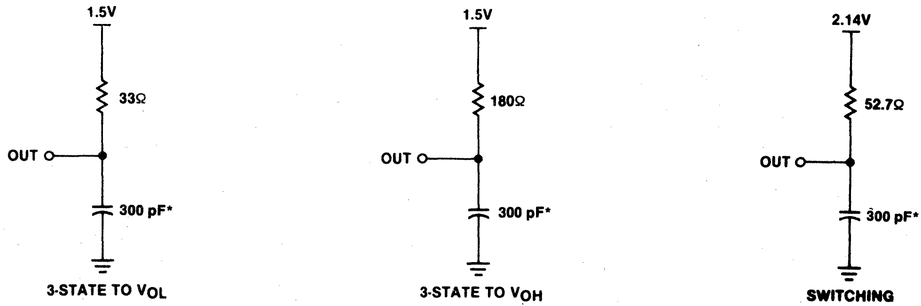
NOTE:
*C_L = 200 pF for plastic 8282/8283.
1. See waveforms and test load circuit on following page.
2. For Extended Temperature EXPRESS the Preliminary Maximum Values are TIVOV = 25 vs 22, 35 vs 30; TSHOV = 45, 55; TEHOZ = 25; TELOV = 50.

A.C. TESTING INPUT, OUTPUT WAVEFORM



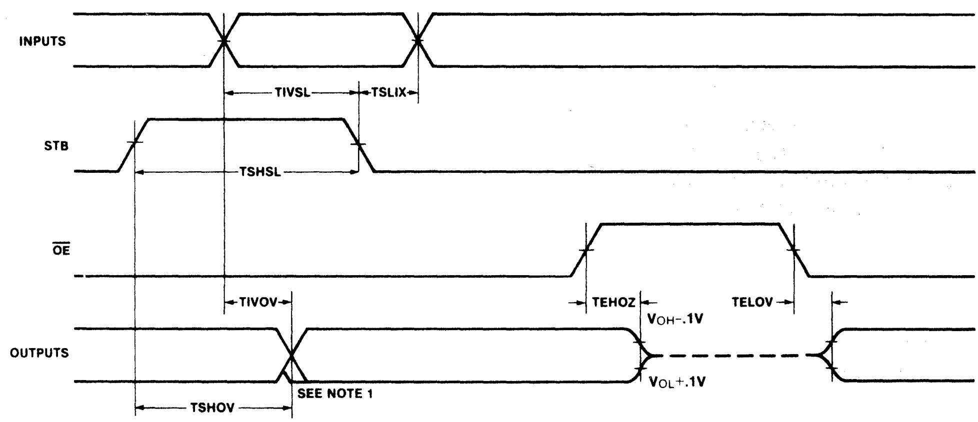
A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 1.5V FOR BOTH A LOGIC "1" AND "0." INPUT RISE AND FALL TIMES ARE MEASURED FROM 0.8V TO 2.0V AND ARE DRIVEN AT 5ns ± 2ns.

OUTPUT TEST LOAD CIRCUITS

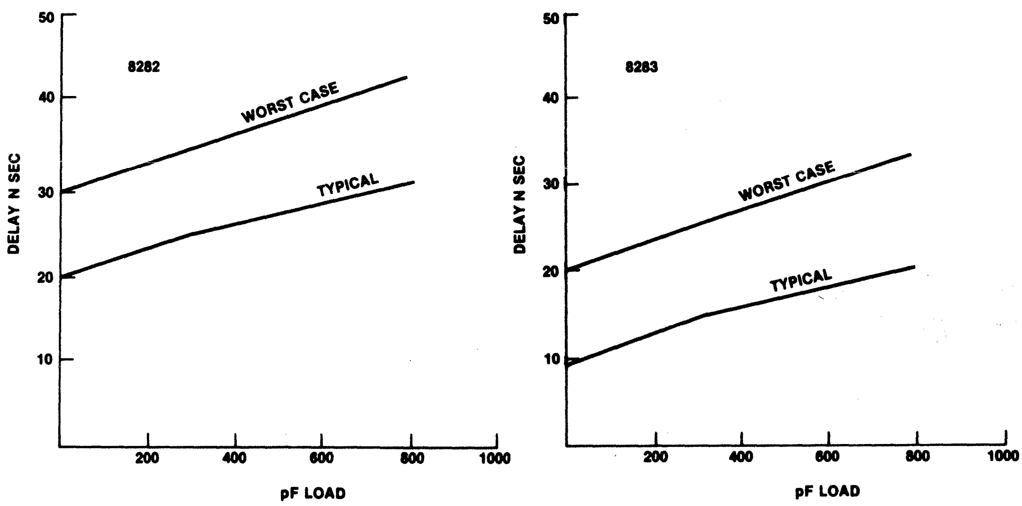


*200 pF for plastic 8282/8283.

WAVEFORMS



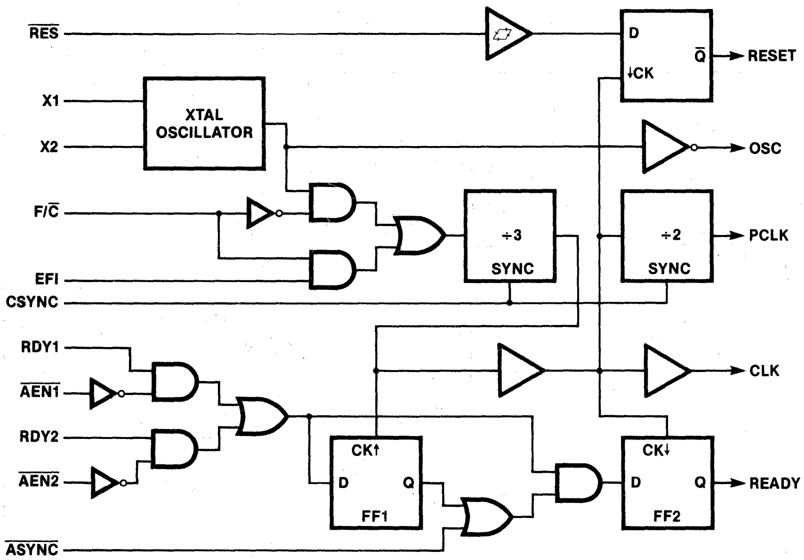
NOTE: 1. OUTPUT MAY BE MOMENTARILY INVALID FOLLOWING THE HIGH GOING STB TRANSITION.
2. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS OTHERWISE NOTED.



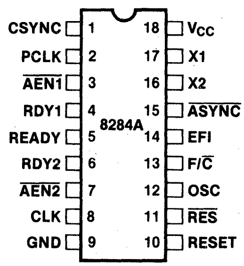
Output Delay vs. Capacitance

8284A/8284A-1 CLOCK GENERATOR AND DRIVER FOR iAPX 86, 88 PROCESSORS

- Generates the System Clock for the iAPX 86, 88 Processors:
  5 MHz, 8 MHz with 8284A
  10 MHz with 8284A-1
- Uses a Crystal or a TTL Signal for Frequency Source
- Provides Local READY and MULTIBUS® READY Synchronization
- 18-Pin Package
- Single +5V Power Supply
- Generates System Reset Output from Schmitt Trigger Input
- Capable of Clock Synchronization with Other 8284As
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range



8284A/8284A-1 Block Diagram



8284A/8284A-1 Pin Configuration

Table 1. Pin Description

Symbol	Type	Name and Function
$\overline{AEN1}$, $\overline{AEN2}$	I	**Address Enable:** $\overline{AEN}$ is an active LOW signal. AEN serves to qualify its respective Bus Ready Signal (RDY1 or RDY2). $\overline{AEN1}$ validates RDY1 while $\overline{AEN2}$ validates RDY2. Two AEN signal inputs are useful in system configurations which permit the processor to access two Multi-Master System Busses. In non Multi-Master configurations the $\overline{AEN}$ signal inputs are tied true (LOW).
RDY1, RDY2	I	**Bus Ready:** (Transfer Complete). RDY is an active HIGH signal which is an indication from a device located on the system data bus that data has been received, or is available. RDY1 is qualified by $\overline{AEN1}$ while RDY2 is qualified by $\overline{AEN2}$.
$\overline{ASYNC}$	I	**Ready Synchronization Select:** $\overline{ASYNC}$ is an input which defines the synchronization mode of the READY logic. When $\overline{ASYNC}$ is low, two stages of READY synchronization are provided. When $\overline{ASYNC}$ is left open (internal pull-up resistor is provided) or HIGH a single stage of READY synchronization is provided.
READY	O	**Ready:** READY is an active HIGH signal which is the synchronized RDY signal input. READY is cleared after the guaranteed hold time to the processor has been met.
X1, X2	I	**Crystal In:** X1 and X2 are the pins to which a crystal is attached. The crystal frequency is 3 times the desired processor clock frequency.
F/$\overline{C}$	I	**Frequency/Crystal Select:** F/$\overline{C}$ is a strapping option. When strapped LOW, F/$\overline{C}$ permits the processor's clock to be generated by the crystal. When F/$\overline{C}$ is strapped HIGH, CLK is generated from the EFI input.
EFI	I	**External Frequency:** When F/$\overline{C}$ is strapped HIGH, CLK is generated from the input frequency appearing on this pin. The input signal is a square wave 3 times the frequency of the desired CLK output.
CLK	O	**Processor Clock:** CLK is the clock output used by the processor and all devices which directly connect to the processor's local bus (i.e., the bipolar support chips and other MOS devices). CLK has an output frequency which is ⅓ of the crystal or EFI input frequency and a ⅓ duty cycle. An output HIGH of 4.5 volts (V_{CC}= 5V) is provided on this pin to drive MOS devices.
PCLK	O	**Peripheral Clock:** PCLK is a TTL level peripheral clock signal whose output frequency is ½ that of CLK and has a 50% duty cycle.
OSC	O	**Oscillator Output:** OSC is the TTL level output of the internal oscillator circuitry. Its frequency is equal to that of the crystal.
$\overline{RES}$	I	**Reset In:** $\overline{RES}$ is an active LOW signal which is used to generate RESET. The 8284A provides a Schmitt trigger input so that an RC connection can be used to establish the power-up reset of proper duration.
RESET	O	**Reset:** RESET is an active HIGH signal which is used to reset the 8086 family processors. Its timing characteristics are determined by $\overline{RES}$.
CSYNC	I	**Clock Synchronization:** CSYNC is an active HIGH signal which allows multiple 8284As to be synchronized to provide clocks that are in phase. When CSYNC is HIGH the internal counters are reset. When CSYNC goes LOW the internal counters are allowed to resume counting. CSYNC needs to be externally synchronized to EFI. When using the internal oscillator CSYNC should be hardwired to ground.
GND		**Ground.**
V_{CC}		**Power:** +5V supply.

FUNCTIONAL DESCRIPTION

General

The 8284A is a single chip clock generator/driver for the iAPX 86, 88 processors. The chip contains a crystal-controlled oscillator, a divide-by-three counter, complete MULTIBUS "Ready" synchronization and reset logic. Refer to Figure 1 for Block Diagram and Figure 2 for Pin Configuration.

Oscillator

The oscillator circuit of the 8284A is designed primarily for use with an external series resonant, fundamental mode, crystal from which the basic operating frequency is derived.

The crystal frequency should be selected at three times the required CPU clock. X1 and X2 are the two crystal input crystal connections. For the most stable operation of the oscillator (OSC) output circuit, two series resistors ($R_1 = R_2 = 510\ \Omega$) as shown in the waveform figures are recommended. The output of the oscillator is buffered and brought out on OSC so that other system timing signals can be derived from this stable, crystal-controlled source.

For systems which have a V_{CC} ramp time $\geq$ 1V/ms and/or have inherent board capacitance between X1 or X2, exceeding 10 pF (not including 8284A pin capacitance), the two 510Ω resistors should be used. This circuit provides optimum stability for the oscillator in such extreme conditions. It is advisable to limit stray capacitances to less than 10 pF on X1 and X2 to minimize deviation from operating at the fundamental frequency.

If EFI is used and no crystal is connected, it is recommended that X_1 or X_2 should be tied to V_{CC} through a 510Ω resistor to prevent the oscillator from free running which might produce HF noise and additional I_{CC} current.

Clock Generator

The clock generator consists of a synchronous divide-by-three counter with a special clear input that inhibits the counting. This clear input (CSYNC) allows the output clock to be synchronized with an external event (such as another 8284A clock). It is necessary to synchronize the CSYNC input to the EFI clock external to the 8284A. This is accomplished with two Schottky flip-flops. The counter output is a 33% duty cycle clock at one-third the input frequency.

The F/$\overline{C}$ input is a strapping pin that selects either the crystal oscillator or the EFI input as the clock for the ÷3 counter. If the EFI input is selected as the clock source, the oscillator section can be used independently for another clock source. Output is taken from OSC.

Clock Outputs

The CLK output is a 33% duty cycle MOS clock driver designed to drive the iAPX 86, 88 processors directly. PCLK is a TTL level peripheral clock signal whose output frequency is ½ that of CLK. PCLK has a 50% duty cycle.

Reset Logic

The reset logic provides a Schmitt trigger input ($\overline{RES}$) and a synchronizing flip-flop to generate the reset timing. The reset signal is synchronized to the falling edge of CLK. A simple RC network can be used to provide power-on reset by utilizing this function of the 8284A.

READY Synchronization

Two READY inputs (RDY1, RDY2) are provided to accommodate two Multi-Master system busses. Each input has a qualifier ($\overline{AEN1}$ and $\overline{AEN2}$, respectively). The $\overline{AEN}$ signals validate their respective RDY signals. If a Multi-Master system is not being used the $\overline{AEN}$ pin should be tied LOW.

Synchronization is required for all asynchronous active-going edges of either RDY input to guarantee that the RDY setup and hold times are met. Inactive-going edges of RDY in normally ready systems do not require synchronization but must satisfy RDY setup and hold as a matter of proper system design.

The $\overline{ASYNC}$ input defines two modes of READY synchronization operation.

When $\overline{ASYNC}$ is LOW, two stages of synchronization are provided for active READY input signals. Positive-going asynchronous READY inputs will first be synchronized to flip-flop one at the rising edge of CLK and then synchronized to flip-flop two at the next falling edge of CLK, after which time the READY output will go active (HIGH). Negative-going asynchronous READY inputs will be synchronized directly to flip-flop two at the falling edge of CLK, after which time the READY output will go inactive. This mode of operation is intended for use by asynchronous (normally not ready) devices in the system which cannot be guaranteed by design to meet the required RDY setup timing, T_{R1VCL}, on each bus cycle.

When $\overline{ASYNC}$ is high or left open, the first READY flip-flop is bypassed in the READY synchronization logic. READY inputs are synchronized by flip-flop two on the falling edge of CLK before they are presented to the processor. This mode is available for synchronous devices that can be guaranteed to meet the required RDY setup time.

$\overline{ASYNC}$ can be changed on every bus cycle to select the appropriate mode of synchronization for each device in the system.

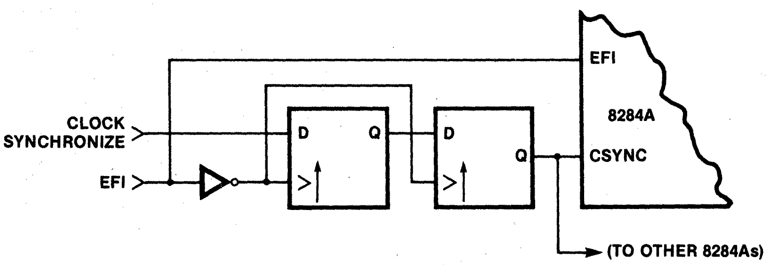


Figure 3. CSYNC Synchronization

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias . . . . 0°C to 70°C
Storage Temperature . . . . −65°C to +150°C
All Output and Supply Voltages . . . . −0.5V to +7V
All Input Voltages . . . . −1.0V to +5.5V
Power Dissipation . . . . 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS ($T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
I_F	Forward Input Current ($\overline{ASYNC}$) Other Inputs		−1.3 −0.5	mA mA	$V_F = 0.45V$ $V_F = 0.45V$
I_R	Reverse Input Current ($\overline{ASYNC}$) Other Inputs		50 50	μA μA	$V_R = V_{CC}$ $V_R = 5.25V$
V_C	Input Forward Clamp Voltage		−1.0	V	$I_C = -5mA$
I_{CC}	Power Supply Current		170	mA	
V_{IL}	Input LOW Voltage		0.8	V	
V_{IH}	Input HIGH Voltage	2.0		V	
V_{IHR}	Reset Input HIGH Voltage	2.6		V	
V_{OL}	Output LOW Voltage		0.45	V	5mA
V_{OH}	Output HIGH Voltage CLK Other Outputs	4 2.4		V V	−1mA −1mA
$V_{IHR} - V_{ILR}$	$\overline{RES}$ Input Hysteresis	0.25		V	

A.C. CHARACTERISTICS ($T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$)

TIMING REQUIREMENTS

Symbol	Parameter	Min.	Max.	Units	Test Conditions
t_{EHEL}	External Frequency HIGH Time	13		ns	90%−90% V_{IN}
t_{ELEH}	External Frequency LOW Time	13		ns	10%−10% V_{IN}
t_{ELEL}	EFI Period	33		ns	(Note 1)
	XTAL Frequency	12	30	MHz	
t_{R1VCL}	RDY1, RDY2 Active Setup to CLK	35		ns	$\overline{ASYNC}$ = HIGH
t_{R1VCH}	RDY1, RDY2 Active Setup to CLK	35		ns	$\overline{ASYNC}$ = LOW
t_{R1VCL}	RDY1, RDY2 Inactive Setup to CLK	35		ns	
t_{CLR1X}	RDY1, RDY2 Hold to CLK	0		ns	
t_{AYVCL}	$\overline{ASYNC}$ Setup to CLK	50		ns	
t_{CLAYX}	$\overline{ASYNC}$ Hold to CLK	0		ns	
t_{A1VR1V}	$\overline{AEN1}$, $\overline{AEN2}$ Setup to RDY1, RDY2	15		ns	
t_{CLA1X}	$\overline{AEN1}$, $\overline{AEN2}$ Hold to CLK	0		ns	
t_{YHEH}	CSYNC Setup to EFI	20		ns	
t_{EHYL}	CSYNC Hold to EFI	10		ns	
t_{YHYL}	CSYNC Width	$2 \cdot t_{ELEL}$		ns	
t_{I1HCL}	$\overline{RES}$ Setup to CLK	65		ns	(Note 1)
t_{CLI1H}	$\overline{RES}$ Hold to CLK	20		ns	(Note 1)

A.C. CHARACTERISTICS (Continued)

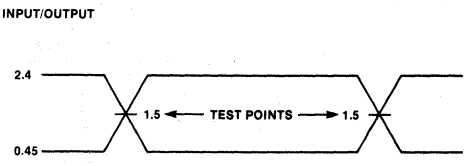
TIMING RESPONSES

Symbol	Parameter	Min. 8284A	Min. 8284A-1	Max.	Units	Test Conditions
t_{CLCL}	CLK Cycle Period	125	100		ns	
t_{CHCL}	CLK HIGH Time	$(1/3\ t_{CLCL})+2$	39		ns	
t_{CLCH}	CLK LOW Time	$(2/3\ t_{CLCL})-15$	53		ns	
t_{CH1CH2} t_{CL2CL1}	CLK Rise or Fall Time			10	ns	1.0V to 3.5V
t_{PHPL}	PCLK HIGH Time	$t_{CLCL}-20$	$t_{CLCL}-20$		ns	
t_{PLPH}	PCLK LOW Time	$t_{CLCL}-20$	$t_{CLCL}-20$		ns	
t_{RYLCL}	Ready Inactive to CLK (See Note 3)	−8	−8		ns	
t_{RYHCH}	Ready Active to CLK (See Note 2)	$(2/3\ t_{CLCL})-15$	53		ns	
t_{CLIL}	CLK to Reset Delay			40	ns	
t_{CLPH}	CLK to PCLK HIGH DELAY			22	ns	
t_{CLPL}	CLK to PCLK LOW Delay			22	ns	
t_{OLCH}	OSC to CLK HIGH Delay	−5	−5	22	ns	
t_{OLCL}	OSC to CLK LOW Delay	2	2	35	ns	
t_{OLOH}	Output Rise Time (except CLK)			20	ns	From 0.8V to 2.0V
t_{OHOL}	Output Fall Time (except CLK)			12	ns	From 2.0V to 0.8V

NOTES:

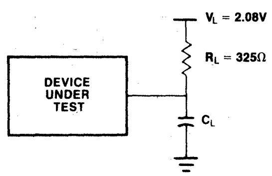
1. Setup and hold necessary only to guarantee recognition at next clock.
2. Applies only to T3 and TW states.
3. Applies only to T2 states.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 1.5V FOR BOTH A LOGIC "1" AND "0." INPUT RISE AND FALL TIMES (MEASURED BETWEEN 0.8V AND 2.0V) ARE 5 ± 2 NS.

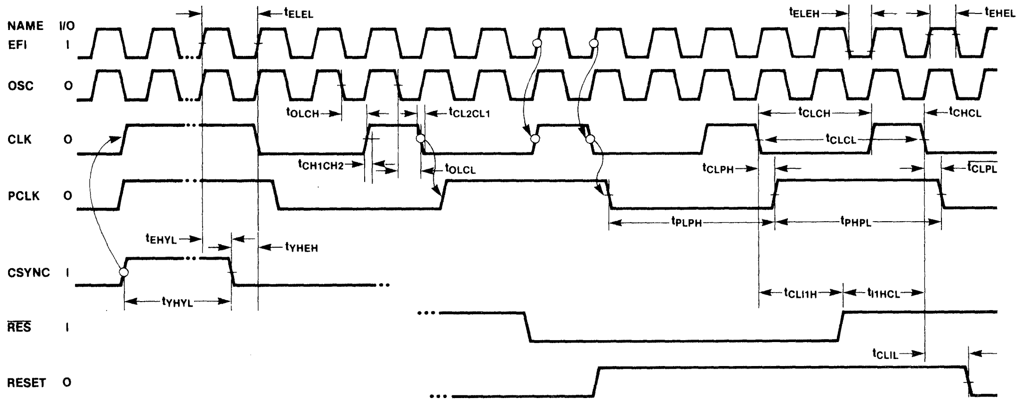
A.C. TESTING LOAD CIRCUIT



C_L = 100pF FOR CLK
C_L = 30pF FOR READY

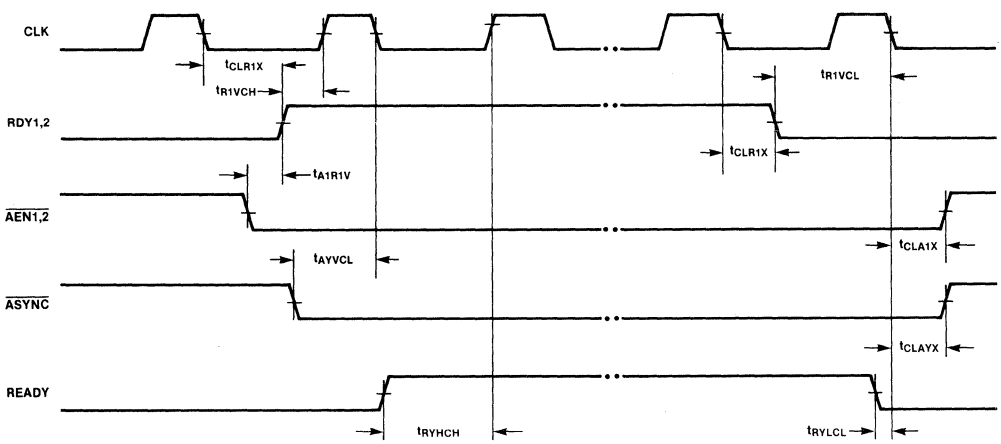
WAVEFORMS

CLOCKS AND RESET SIGNALS



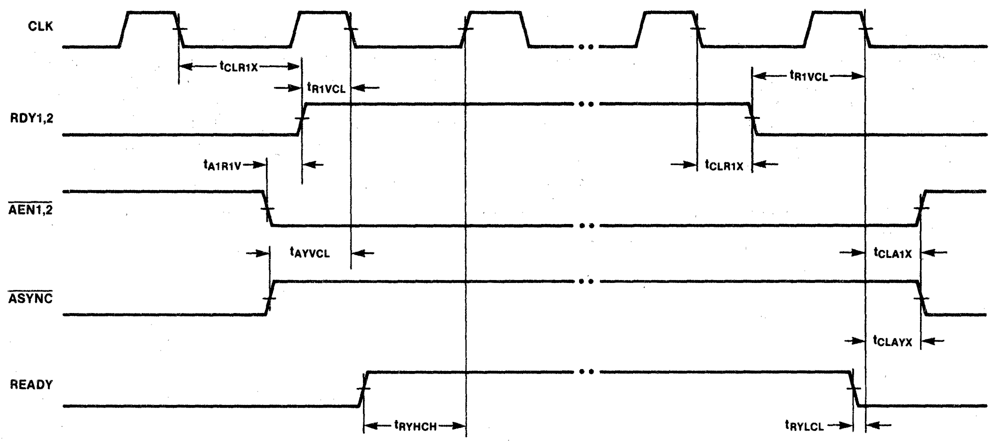
NOTE: ALL TIMING MEASUREMENTS ARE MADE AT 1.5 VOLTS, UNLESS OTHERWISE NOTED.

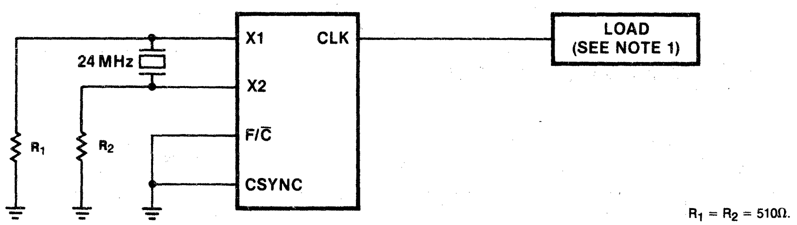
READY SIGNALS (FOR ASYNCHRONOUS DEVICES)



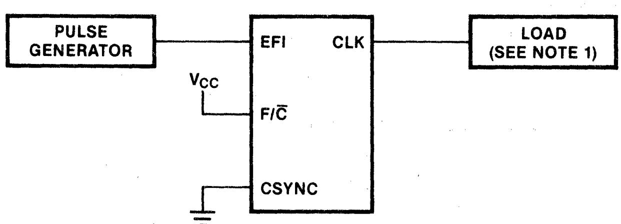
WAVEFORMS (Continued)

READY SIGNALS (FOR SYNCHRONOUS DEVICES)

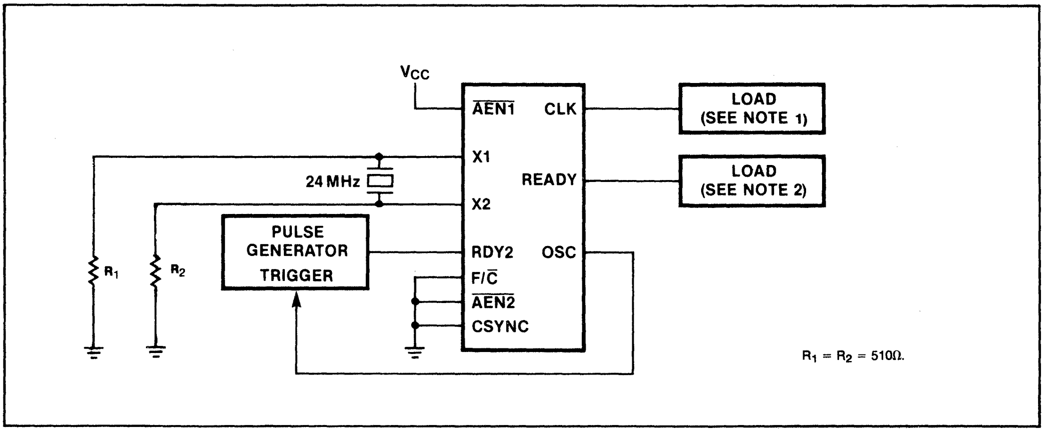




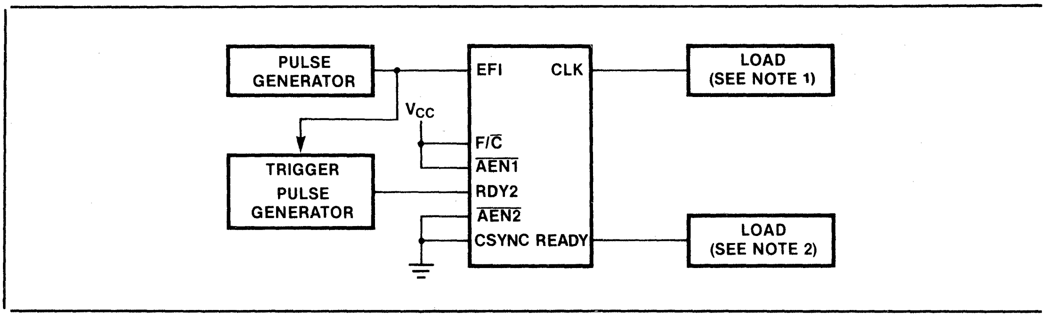
Clock High and Low Time (Using X1, X2)



Clock High and Low Time (Using EFI)



Ready to Clock (Using X1, X2)



Ready to Clock (Using EFI)

NOTES:
1. $C_L = 100$ pF
2. $C_L = 30$ pF

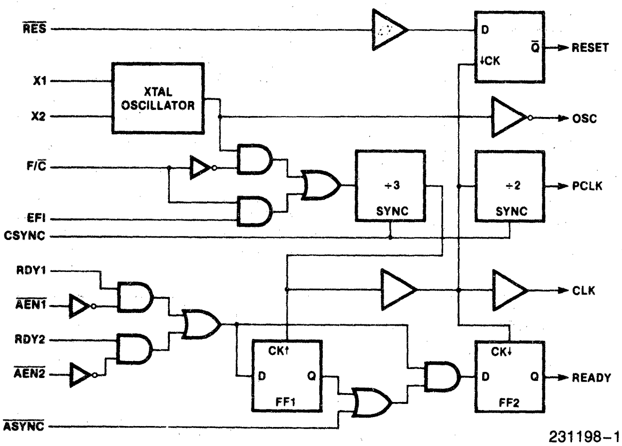
intel®

PRELIMINARY

82C84A/82C84A-5
CHMOS CLOCK GENERATOR AND DRIVER FOR 80C86, 80C88 PROCESSORS

- **Generates the System Clock for the 80C86, 80C88 Processors:**
  82C84A-5 for 5 MHz
  82C84A for 8 MHz
- **Pin Compatible with Bipolar 8284A***
- **Uses a Crystal or an External Frequency Source**
- **Provides Local READY and MULTIBUS® READY Synchronization**
- **Generates System Reset Output from Schmitt Trigger Input**
- **Capable of Clock Synchronization with other 82C84As**
- **Low Power Consumption**
- **Single 5V Power Supply**
- **TTL Compatible Inputs/Outputs**
- **Will Be Available in 18-Lead Plastic DIP and 20-Lead PLCC Packages**
  (See Packaging Spec., Order #231369)

The Intel 82C84A is a high performance CHMOS clock generator-driver designed to service the requirements of the 80C86/88 and 8086/88. Power consumption is a fraction of that of equivalent bipolar circuits. The chip contains a crystal controlled oscillator, a divide-by-three counter and complete READY synchronization and reset logic. Crystal controlled operation up to 15, 25 MHz utilizes a parallel, fundamental mode crystal and two small load capacitors. *The Bipolar 8284A requires two load resistors and a resonant crystal.

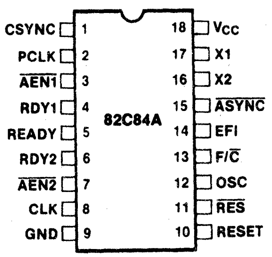


231198–1

82C84A/82C84A-5 Block Diagram

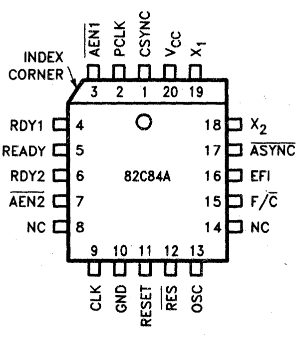
Control Pin	Logical 1	Logical 0
F/C̄	External Clock	Crystal Drive
R̄ĒS̄	Normal	Reset
RDY 1 RDY 2	Bus Ready	Bus not ready
ĀĒN̄ 1 ĀĒN̄ 2	Address Disabled	Address Enabled
ĀS̄ȲN̄C̄	1 Stage Ready Synchronization	2 Stage Ready Synchronization

82C84A/82C84A-5 Pin Description



231198–2

82C84A/82C84A-5 18-Lead DIP Configuration



231198–13

8284A/8284A-5 20-Lead PLCC Configuration

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel. **November 1985**

Order Number: 231198-002

Table 1. Pin Description

Symbol	Type	Name and Function
$\overline{AEN1}$, $\overline{AEN2}$	I	**ADDRESS ENABLE:** $\overline{AEN}$ is an active LOW signal. AEN serves to qualify its repective Bus Ready Signal (RDY1 or RDY2). $\overline{AEN1}$ validates RDY1 while $\overline{AEN2}$ validates RDY2. Two AEN signal inputs are useful in system configurations which permit the processor to access two Multi-Master System Busses. In non Multi-Master configurations the $\overline{AEN}$ signal inputs are tied true (LOW).
RDY1, RDY2	I	**BUS READY:** (Transfer Complete). RDY is an active HIGH signal which is an indication from a device located on the system data bus that data has been received, or is available. RDY1 is qualified by $\overline{AEN1}$ while RDY2 is qualified by $\overline{AEN2}$.
$\overline{ASYNC}$	I	**READY SYNCHRONIZATION SELECT:** $\overline{ASYNC}$ is an input which defines the synchronization mode of the READY logic. When $\overline{ASYNC}$ is LOW, two stages of READY synchronization are provided. When $\overline{ASYNC}$ is left open (an internal pull-up is provided) or HIGH a single stage of READY synchronization is provided.
READY	O	**READY:** READY is an active HIGH signal which is the synchronized RDY signal input. READY is cleared after the guaranteed hold time to the processor has been met.
X1, X2	I	**CRYSTAL IN:** X1 and X2 are the pins to which a crystal is attached. The crystal frequency is 3 times the desired processor clock frequency. (If no crystal is attached, then X1 should be tied to V_{CC} or GND and X2 should be left open.)
F/$\overline{C}$	I	**FREQUENCY/CRYSTAL SELECT:** F/$\overline{C}$ is a strapping option. When strapped LOW, F/$\overline{C}$ permits the processor's clock to be generated by the crystal. When F/$\overline{C}$ is strapped HIGH, CLK is generated from the EFI input.
EFI	I	**EXTERNAL FREQUENCY:** When F/$\overline{C}$ is strapped HIGH, CLK is generated from the input frequency appearing on this pin. The input signal is a square wave 3 times the frequency of the desired CLK output. When F/$\overline{C}$ is strapped LOW, EFI should be tied HIGH or LOW.
CLK	O	**PROCESSOR CLOCK:** CLK is the clock output used by the processor and all devices which directly connect to the processor's local bus (i.e., the bipolar support chips and other MOS devices). CLK has an output frequency which is ⅓ of the crystal or EFI input frequency and a ⅓ duty cycle.
PCLK	O	**PERIPHERAL CLOCK:** PCLK is a TTL level peripheral clock signal whose output frequency is ½ that of CLK and has a 50% duty cycle.
OSC	O	**OSCILLATOR OUTPUT:** OSC is the TTL level output of the internal oscillator circuitry. Its frequency is equal to that of the crystal.
$\overline{RES}$	I	**RESET IN:** $\overline{RES}$ is an active LOW signal which is used to generate RESET. The 82C84A provides a Schmitt trigger input so that an RC connection can be used to establish the power-up reset of proper duration.

Table 1. Pin Description (Continued)

Symbol	Type	Name and Function
RESET	O	**RESET:** RESET is an active HIGH signal which is used to reset the 80C86/88 family processors. Its timing characteristics are determined by $\overline{RES}$.
CSYNC	I	**CLOCK SYNCHRONIZATION:** CSYNC is an active HIGH signal which allows multiple 82C84A's to be synchronized to provide clocks that are in phase. When CSYNC is HIGH the internal counters are reset. When CSYNC goes LOW the internal counters are allowed to resume counting. CSYNC needs to be externally synchronized to EFI. When using the internal oscillator CSYNC should be hardwired to ground.
GND		**GROUND.**
V_{CC}		**POWER:** +5V supply.

FUNCTIONAL DESCRIPTION

Oscillator

The oscillator circuit of the 82C84A is designed primarily for use with an external parallel resonant, fundamental mode crystal from which the basic operating frequency is derived.

The crystal frequency should be selected at three times the required CPU clock. X1 and X2 are the two crystal input crystal connections. For the most stable operation of the oscillator (OSC) output circuit, two capacitors (C1 = C2) as shown in the waveform figures are recommended. The output of the oscillator is buffered and brought out on OSC so that other system timing signals can be derived from this stable, crystal-controlled source.

Capacitors C1, C2 are chosen such that their combined capacitance:

$$CT = \frac{C1 \bullet C2}{C1 + C2} \quad \text{(Including stray capacitance)}$$

matches the load capacitance as specified by the crystal manufacturer. This insures operation within the frequency tolerance specified by the crystal manufacturer.

Clock Generator

The clock generator consists of a synchronous divide-by-three counter with a special clear input that inhibits the counting. This clear input (CSYNC) allows the output clock to be synchronized with an external event (such as another 82C84A clock). It is necessary to synchronize the CSYNC input to the EFI clock external to the 82C84A. This is accomplished with two Schottky flip-flops. The counter output is a 33% duty cycle clock at one-third the input frequency.

The F/$\overline{C}$ input is a strapping pin that selects either the crystal oscillator or the EFI input as the clock for the ÷3 counter. If the EFI input is selected as the clock source, the oscillator section can be used independently for another clock source. Output is taken from OSC.

Clock Outputs

The CLK ouput is a 33% duty cycle MOS clock driver designed to drive the 80C86/88 processors directly. PCLK is a TTL level peripheral clock signal whose output frequency is ½ that of CLK. PCLK has a 50% duty cycle.

Reset Logic

The reset logic provides a Schmitt trigger input ($\overline{RES}$) and a synchronizing flip-flop to generate the reset timing. The reset signal is synchronized to the falling edge of CLK. A simple RC network can be used to provide power-on reset by utilizing this function of the 82C84A.

READY Synchronization

Two READY inputs (RDY1, RDY2) are provided to accommodate two Multi-Master system busses. Each input has a qualifier ($\overline{AEN1}$ and $\overline{AEN2}$, respectively). The $\overline{AEN}$ signals validate their respective RDY signals. If a Multi-Master system is not being used the $\overline{AEN}$ pin should be tied LOW.

Synchronization is required for all asynchronous active-going edges of either RDY input to guarantee that the RDY setup and hold times are met. Inactive-going edges of RDY in normally ready systems do not require synchronization but must satisfy RDY setup and hold as a matter of proper system design.

The $\overline{ASYNC}$ input defines two modes of READY synchronization operation.

When $\overline{ASYNC}$ is LOW, two stages of synchronization are provided for active READY input signals. Positive-going asynchronous READY inputs will first be synchronized to flip-flop one at the rising edge of CLK and then synchronized to flip-flop two at the next falling edge of CLK, after which time the READY output will go active (HIGH). Negative-going asynchronous READY inputs will be synchronized directly to flip-flop two at the falling edge of CLK, after which time the READY output will go inactive. This mode of operation is intended for use by asynchronous (normally not ready) devices in the system which cannot be guaranteed by design to meet the required RDY setup timing, T_{R1VCL}, on each bus cycle.

When $\overline{ASYNC}$ is HIGH, the first READY flip-flop is bypassed in the READY synchronization logic. READY inputs are synchronized by flip-flop two on the falling edge of CLK before they are presented to the processor. This mode is available for synchronous devices that can be guaranteed to meet the required RDY setup time.

$\overline{ASYNC}$ can be changed on every bus cycle to select the appropriate mode of synchronization for each device in the system.

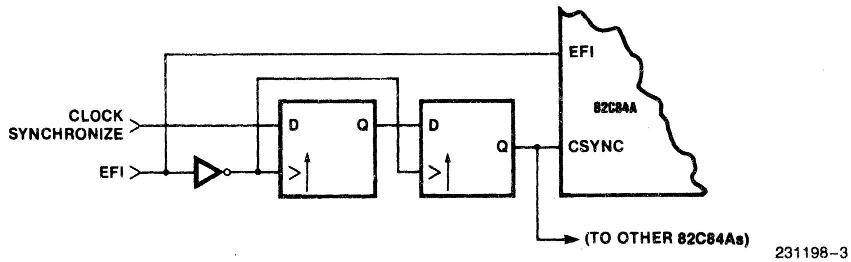


Figure 3. CSYNC Synchronization

ABSOLUTE MAXIMUM RATINGS*

Supply Voltage .... −0.5V to 7.0V
Input Voltage Applied .... −0.5V to V_{CC} +0.5V
Output Voltage Applied ...... −0.5V to V_{CC} +0.5V
Storage Temperature .... −65°C to +150°C
Ambient Temp. Under Bias .... 0°C to +70°C
Power Dissipation .... 1.0 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = 5V ±10%)

Symbol	Parameter		Min	Max	Units	Test Conditions
I_{CC}	Operating Supply Current: 82C84A 82C84A-5			40 10	mA	25 MHz xtal, C_L = 0 15 MHz xtal, C_L = 0
I_{CCS}	Stand By Supply Current (Note 1)			100	μA	
I_{LI}	Input Leakage Current (Note 2)	$\overline{ASYNC}$ Only		10	μA	$\overline{ASYNC}$ = V_{CC}
				−130	μA	$\overline{ASYNC}$ = GND
		All Other Pins		±1.0	μA	0V ≤ V_{IN} ≤ V_{CC}

D.C. CHARACTERISTICS (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input LOW Voltage		0.8	V	
V_{IH}	Input HIGH Voltage	2.2	V_{CC} + 0.5	V	
V_{IHR}	Reset Input HIGH Voltage	0.6 V_{CC}		V	
V_{OL}	Output LOW Voltage		0.4	V	CLK: I_{OL} = 4 mA Others: I_{OL} = 2.5 mA
V_{OH}	Output HIGH Voltage	V_{CC} − 0.4		V V	CLK: I_{OH} = −4 mA Others: I_{OH} = −2.5 mA
V_{IHR}-V_{ILR}	$\overline{RES}$ Input Hysteresis	0.25		V	
C_{IN}	Input Capacitance		7	pF	freq = 1 MHz

NOTES:
1. V_{IH}, F/$\overline{C}$, X1 ≥ V_{CC} − 0.2V; V_{IL}, X2 ≤ 0.2V; ASYNC = V_{CC} or ASYNC = OPEN.
2. An internal pull-up resistor is implemented on the $\overline{ASYNC}$ input.

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = 5V ±10%)

TIMING REQUIREMENTS

Symbol	Parameter	82C84A		82C84A-5		Units	Test Conditions
		Min	Max	Min	Max		
t_{EHEL}	External Frequency HIGH Time	13		20		ns	90%–90% V_{IN}
t_{ELEH}	External Frequency LOW Time	13		20		ns	10%–10% V_{IN}
t_{ELEL}	EFI Period	36		66		ns	(Note 1)
	XTAL Frequency	2.4	25	6.0	15	MHz	
t_{R1VCL}	RDY1, RDY2 Active Setup to CLK	35		35		ns	$\overline{ASYNC}$ = HIGH
t_{R1VCH}	RDY1, RDY2 Active Setup to CLK	35		35		ns	$\overline{ASYNC}$ = LOW
t_{R1VCL}	RDY1, RDY2 Inactive Setup to CLK	35		35		ns	
t_{CLR1X}	RDY1, RDY2 Hold to CLK	0		0		ns	
t_{AYVCL}	$\overline{ASYNC}$ Setup to CLK	50		50		ns	
t_{CLAYX}	$\overline{ASYNC}$ Hold to CLK	0		0		ns	
t_{A1VR1V}	$\overline{AEN1}$, $\overline{AEN2}$ Setup to RDY1, RDY2	15		15		ns	
t_{CLA1X}	$\overline{AEN1}$, $\overline{AEN2}$ Hold to CLK	0		0		ns	
t_{YHEH}	CSYNC Setup to EFI	20		20		ns	
t_{EHYL}	CSYNC Hold to EFI	20		20		ns	
t_{YHYL}	CSYNC Width	2 • t_{ELEL}		2 • t_{ELEL}		ns	
t_{I1HCL}	$\overline{RES}$ Setup to CLK	65		65		ns	(Note 2)
t_{CLI1H}	$\overline{RES}$ Hold to CLK	20		20		ns	(Note 2)
t_{ILIH}	Input Rise Time		15		15	ns	(Note 1)
t_{IHIL}	Input Fall Time		15		15	ns	(Note 1)

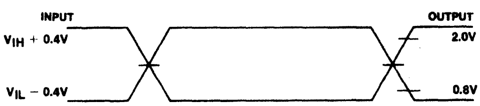
A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES

Symbol	Parameter	Min 82C84A	Min 82C84A-5	Max	Units	Test Conditions
t_{CLCL}	CLK Cycle Period	125	200		ns	
t_{CHCL}	CLK HIGH Time	$(1/3\ t_{CLCL}) + 2$	$(1/3\ t_{CLCL}) + 2$		ns	
t_{CLCH}	CLK LOW Time	$(2/3\ t_{CLCL}) - 15$	$(2/3\ t_{CLCL}) - 15$		ns	
t_{CH1CH2} t_{CL2CL1}	CLK Rise or Fall Time			10	ns	1.0V to 3.5V
t_{PHPL}	PCLK HIGH Time	$t_{CLCL} - 20$	$t_{CLCL} - 20$		ns	
t_{PLPH}	PCLK LOW Time	$t_{CLCL} - 20$	$t_{CLCL} - 20$		ns	
t_{RYLCL}	Ready Inactive to CLK (See Note 4)	−8	−8		ns	
t_{RYHCH}	Ready Active to CLK (See Note 3)	$(2/3\ t_{CLCL}) - 15$	$(2/3\ t_{CLCL}) - 15$		ns	
t_{CLIL}	CLK to Reset Delay			40	ns	
t_{CLPH}	CLK to PCLK HIGH DELAY			22	ns	
t_{CLPL}	CLK to PCLK LOW Delay			22	ns	
t_{OLCH}	OSC to CLK HIGH Delay	−5	−5	22	ns	
t_{OLCL}	OSC to CLK LOW Delay	2	2	35	ns	
t_{OLOH}	Output Rise Time (except CLK)			15	ns	From 0.8V to 2.0V
t_{OHOL}	Output Fall Time (except CLK)			15	ns	From 2.0V to 0.8V

NOTES:
1. Transition between V_{IL}(max) − 0.4V and V_{IH}(min) + 0.4V.
2. Setup and hold necessary only to guarantee recognition at next clock.
3. Applies only to T3 and TW states.
4. Applies only to T2 states.

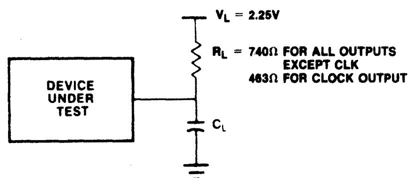
A.C. TESTING INPUT, OUTPUT WAVEFORM



231198-4

A.C. TESTING: ALL INPUT SIGNALS MUST SWITCH BETWEEN 0.45V AND 2.4V. T_{RISE} AND T_{FALL} MUST BE ≤ 15 ns.
ALL TIMING MEASUREMENTS ARE MADE AT 0.8V AND 2.0V.

A.C. TESTING LOAD CIRCUIT

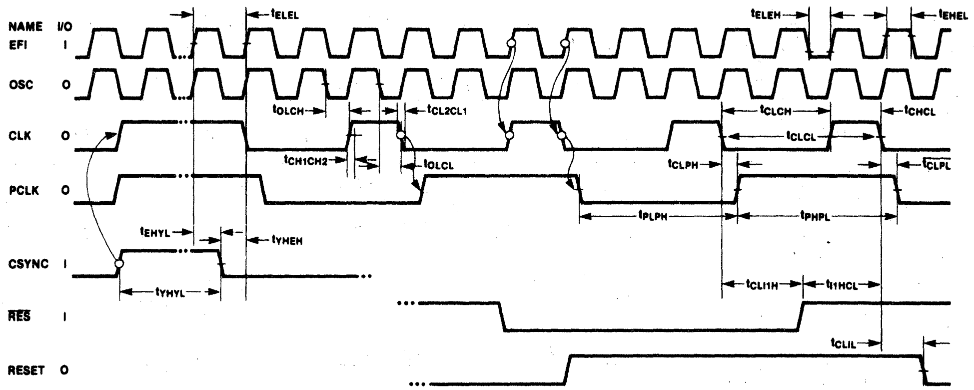


231198-5

C_L = 100 pF FOR CLK
C_L = 30 pF FOR READY
C_L INCLUDES PROBE AND JIG CAPACITANCE

WAVEFORMS

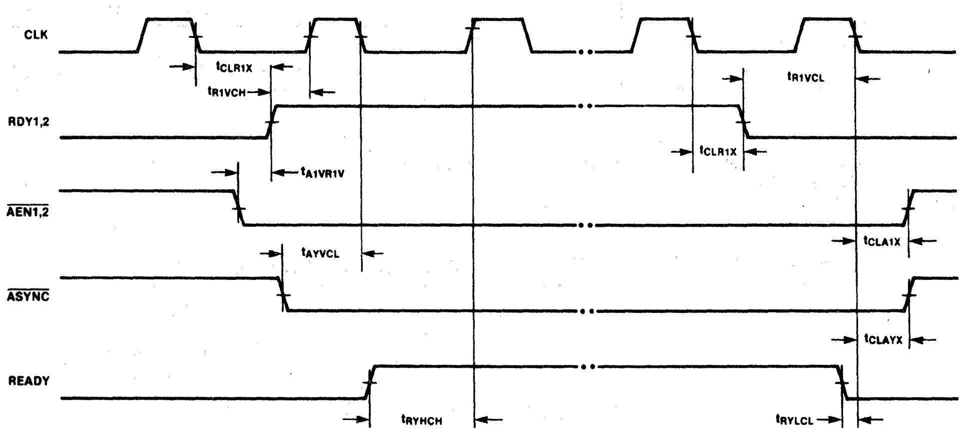
CLOCKS AND RESET SIGNALS



231198-6

NOTE:
ALL TIMING MEASUREMENTS ARE MADE AT 0.8V AND 2.0V, UNLESS OTHERWISE NOTED.

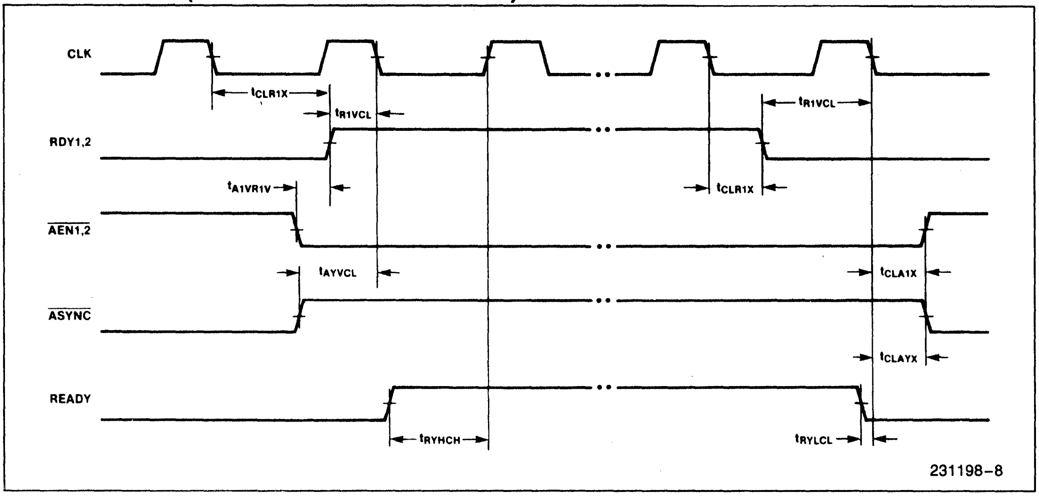
READY SIGNALS (FOR ASYNCHRONOUS DEVICES)

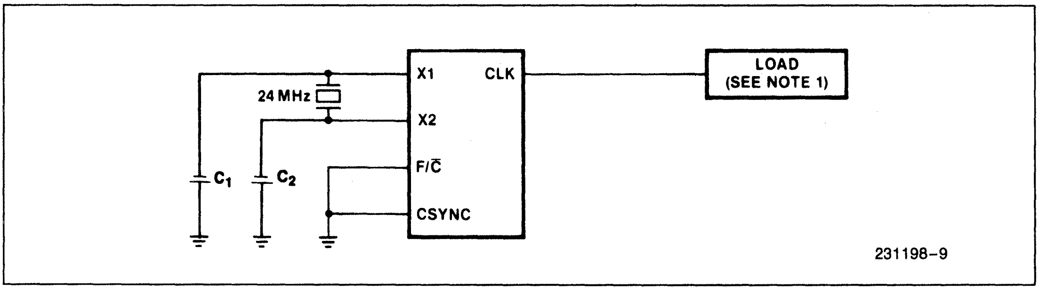


231198-7

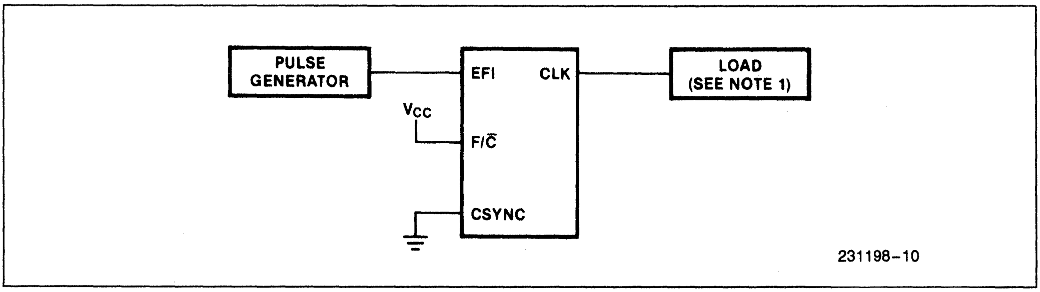
WAVEFORMS (Continued)

READY SIGNALS (FOR SYNCHRONOUS DEVICES)

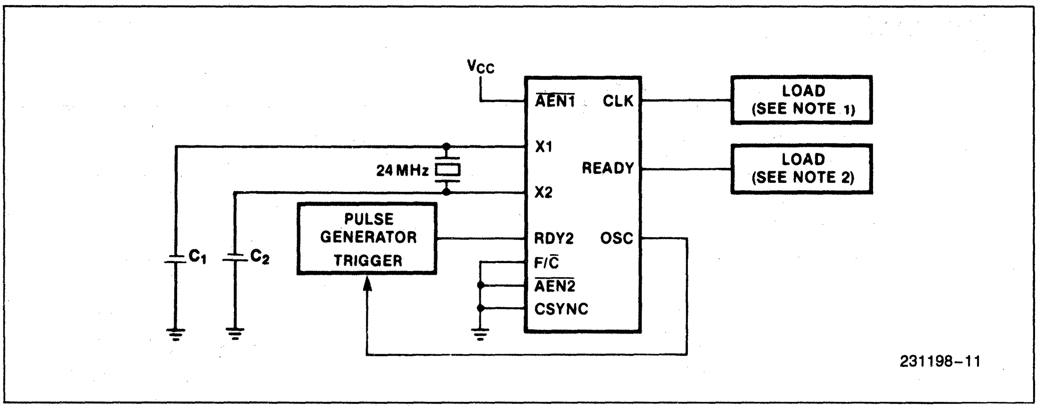




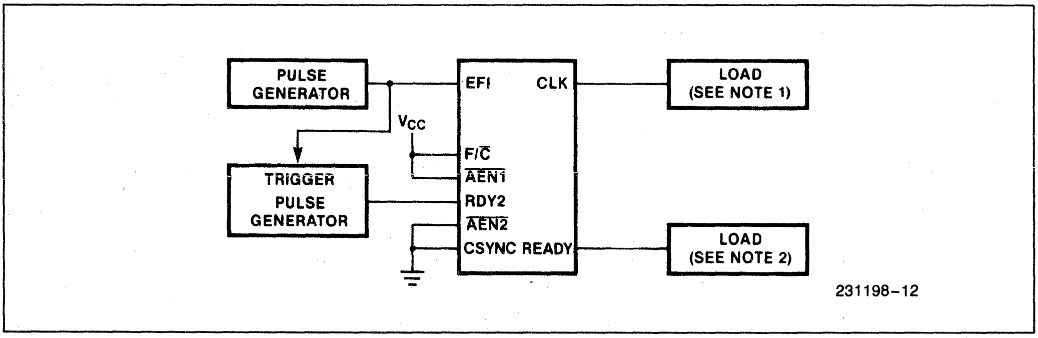
Clock High and Low Time (Using X1, X2)



Clock High and Low Time (Using EFI)



Ready to Clock (Using X1, X2)



Ready to Clock (Using EFI)

NOTES:
1. C_L = 100 pF
2. C_L = 30 pF

8286/8287
OCTAL BUS TRANSCEIVER

- Data Bus Buffer Driver for iAPX 86,88,186,188, MCS-80™, MCS-85™, and MCS-48™ Families
- High Output Drive Capability for Driving System Data Bus
- Fully Parallel 8-Bit Transceivers
- 3-State Outputs
- 20-Pin Package with 0.3" Center
- No Output Low Noise when Entering or Leaving High Impedance State
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range

The 8286 and 8287 are 8-bit bipolar transceivers with 3-state outputs. The 8287 inverts the input data at its outputs while the 8286 does not. Thus, a wide variety of applications for buffering in microcomputer systems can be met.

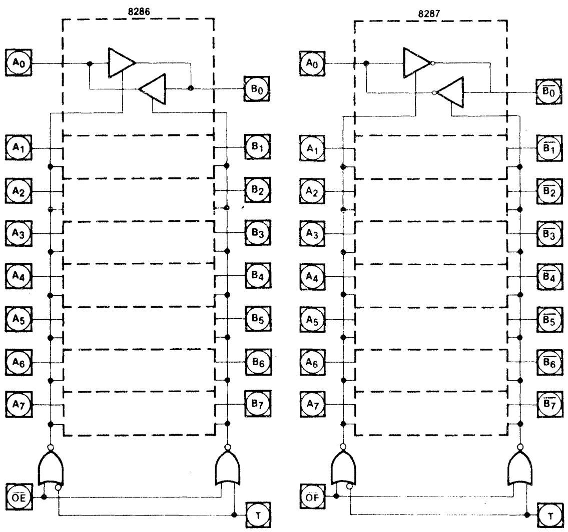


Figure 1. Logic Diagrams

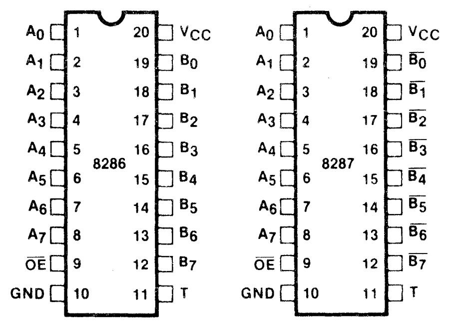


Figure 2. Pin Configurations

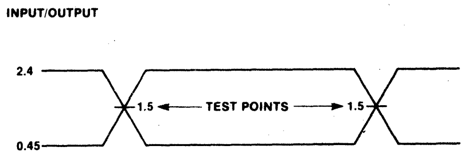
Table 1. Pin Description

Symbol	Type	Name and Function
T	I	**Transmit:** T is an input control signal used to control the direction of the transceivers. When HIGH, it configures the transceiver's B_0–B_7 as outputs with A_0–A_7 as inputs. T LOW configures A_0–A_7 as the outputs with B_0–B_7 serving as the inputs.
$\overline{OE}$	I	**Output Enable:** $\overline{OE}$ is an input control signal used to enable the appropriate output driver (as selected by T) onto its respective bus. This signal is active LOW.
A_0–A_7	I/O	**Local Bus Data Pins:** These pins serve to either present data to or accept data from the processor's local bus depending upon the state of the T pin.
B_0–B_7(8286) $\overline{B_0}$–$\overline{B_7}$(8287)	I/O	**System Bus Data Pins:** These pins serve to either present data to or accept data from the system bus depending upon the state of the T pin.

FUNCTIONAL DESCRIPTION

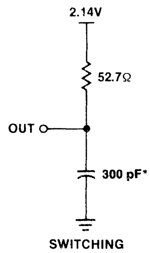
The 8286 and 8287 transceivers are 8-bit transceivers with high impedance outputs. With T active HIGH and $\overline{OE}$ active LOW, data at the A_0–A_7 pins is driven onto the B_0–B_7 pins. With T inactive LOW and $\overline{OE}$ active LOW, data at the B_0–B_7 pins is driven onto the A_0–A_7 pins. No output low glitching will occur whenever the transceivers are entering or leaving the high impedance state.

A.C. TESTING INPUT, OUTPUT WAVEFORM

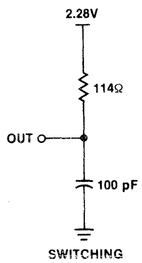


A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." THE CLOCK IS DRIVEN AT 4.3V and 0.25V. TIMING MEASUREMENTS ARE MADE AT 1.5V FOR BOTH A LOGIC "1" AND "0." INPUT RISE AND FALL TIMES ARE 5 ± 2 NS., MEASURED BETWEEN 0.8V AND 2.0V.

TEST LOAD CIRCUITS



B OUTPUT



A OUTPUT

*200 pF for plastic 8286/8287

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias.................0°C to 70°C
Storage Temperature.............–65°C to +150°C
All Output and Supply Voltages........–0.5V to +7V
All Input Voltages..................–1.0V to +5.5V
Power Dissipation..........................1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (V_{CC} = +5V ±10%, T_A = 0°C to 70°C)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_C	Input Clamp Voltage		-1	V	I_C = -5 mA
I_{CC}	Power Supply Current—8287 —8286		130 160	mA mA	
I_F	Forward Input Current		-0.2	mA	V_F = 0.45V
I_R	Reverse Input Current		50	μA	V_R = 5.25V
V_{OL}	Output Low Voltage —B Outputs —A Outputs		.45 .45	V V	I_{OL} = 32 mA I_{OL} = 16 mA
V_{OH}	Output High Voltage —B Outputs —A Outputs	2.4 2.4		V V	I_{OH} = -5 mA I_{OH} = -1 mA
I_{OFF} I_{OFF}	Output Off Current Output Off Current		I_F I_R		V_{OFF} = 0.45V V_{OFF} = 5.25V
V_{IL}	Input Low Voltage —A Side —B Side		0.8 0.9	V V	V_{CC} = 5.0V, See Note 1 V_{CC} = 5.0V, See Note 1
V_{IH}	Input High Voltage	2.0		V	V_{CC} = 5.0V, See Note 1
C_{IN}	Input Capacitance		12	pF	F = 1 MHz V_{BIAS} = 2.5V, V_{CC} = 5V T_A = 25°C

NOTE:
1. B Outputs—I_{OL} = 32 mA, I_{OH} = −5 mA, C_L = 300 pF*: A Outputs—I_{OL} = 16 mA, I_{OH} = −1 mA, C_L = 100 pF.

A.C. CHARACTERISTICS (V_{CC} = +5V ±10%, T_A = 0°C to 70°C) (See Note 2)

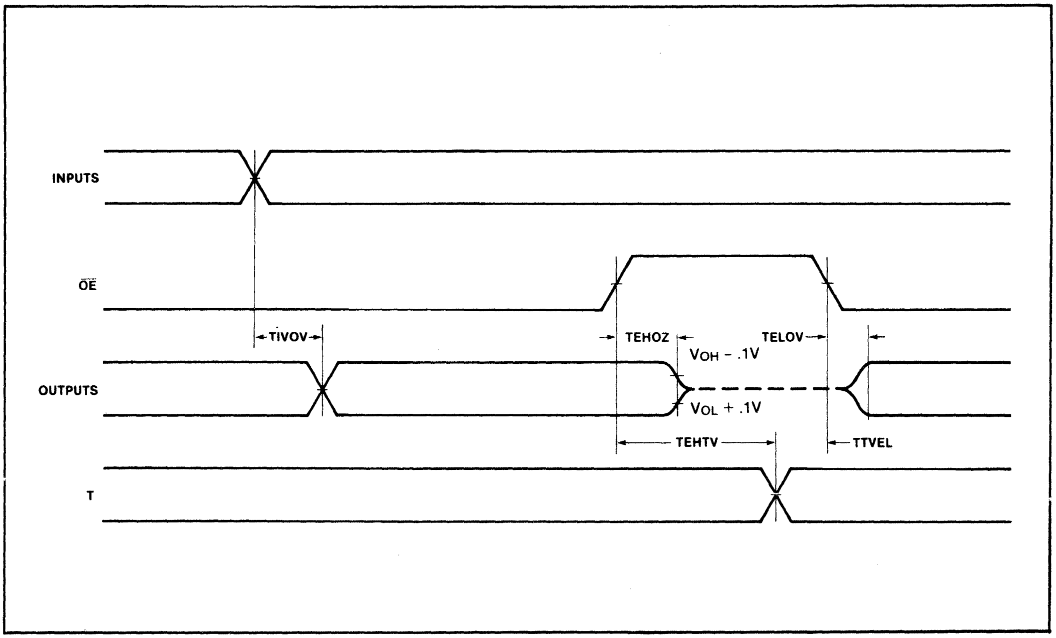
Loading: B Outputs—I_{OL} = 32 mA, I_{OH} = −5 mA, C_L = 300 pF*
A Outputs—I_{OL} = 16 mA, I_{OH} = −1 mA, C_L = 100 pF

Symbol	Parameter	Min	Max	Units	Test Conditions
TIVOV	Input to Output Delay Inverting Non-Inverting	 5 5	 22 30	 ns ns	(See Note 1)
TEHTV	Transmit/Receive Hold Time	5		ns	
TTVEL	Transmit/Receive Setup	10		ns	
TEHOZ	Output Disable Time	5	18	ns	
TELOV	Output Enable Time	10	30	ns	
TOLOH	Input, Output Rise Time		20	ns	From 0.8 V to 2.0V
TOHOL	Input, Output Fall Time		12	ns	From 2.0V to 8.0V

*C_L = 200 pF for plastic 8286/8287

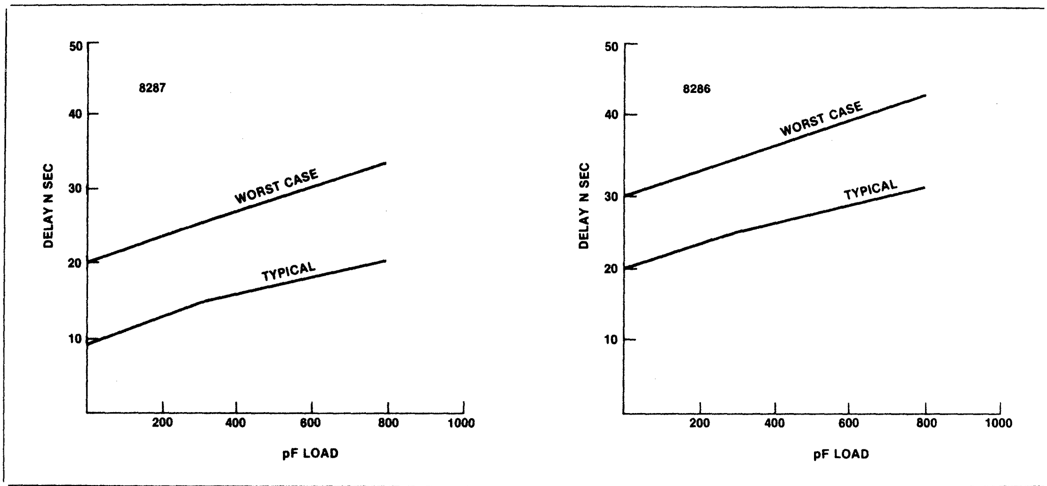
NOTE:
1. See waveforms and test load circuit on following page.
2. For Extended Temperature EXPRESS the Preliminary Maximum Values are TIVOV = 25 vs 22, 35 vs 30; TEHOZ = 25; TELOV = 50.

WAVEFORMS



NOTE:

1. All timing measurements are made at 1.5V unless otherwise noted.



Output Delay versus Capacitance

intel®

8288
BUS CONTROLLER
FOR iAPX 86, 88 PROCESSORS

- **Bipolar Drive Capability**
- **Provides Advanced Commands**
- **Provides Wide Flexibility in System Configurations**
- **Compatible with 10 MHz iAPX 86 and 8 MHz iAPX 186 based systems.**
- **3-State Command Output Drivers**
- **Configurable for Use with an I/O Bus**
- **Facilitates Interface to One or Two Multi-Master Busses**
- **Available in EXPRESS**
  - **Standard Temperature Range**
  - **Extended Temperature Range**

The Intel® 8288 Bus Controller is a 20-pin bipolar component for use with medium-to-large iAPX 86, 88 processing systems. The bus controller provides command and control timing generation as well as bipolar bus drive capability while optimizing system performance.

A strapping option on the bus controller configures it for use with a multi-master system bus and separate I/O bus.

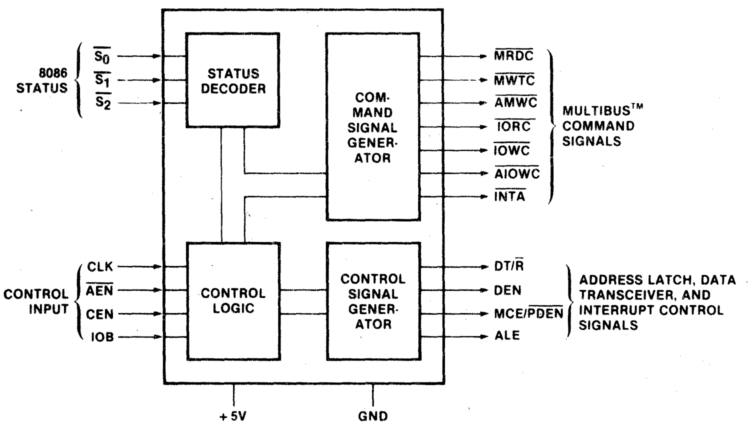


Figure 1. Block Diagram

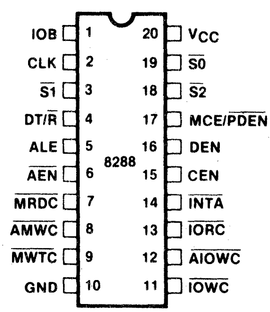


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Type	Name and Function
V_{CC}		**Power:** +5V supply.
GND		**Ground.**
$\overline{S_0}, \overline{S_1}, \overline{S_2}$	I	**Status Input Pins:** These pins are the status input pins from the 8086, 8088 or 8089 processors. The 8288 decodes these inputs to generate command and control signals at the appropriate time. When these pins are not in use (passive) they are all HIGH. (See chart under Command and Control Logic.)
CLK	I	**Clock:** This is a clock signal from the 8284 clock generator and serves to establish when command and control signals are generated.
ALE	O	**Address Latch Enable:** This signal serves to strobe an address into the address latches. This signal is active HIGH and latching occurs on the falling (HIGH to LOW) transition. ALE is intended for use with transparent D type latches.
DEN	O	**Data Enable:** This signal serves to enable data transceivers onto either the local or system data bus. This signal is active HIGH.
DT/$\overline{\text{R}}$	O	**Data Transmit/Receive:** This signal establishes the direction of data flow through the transceivers. A HIGH on this line indicates Transmit (write to I/O or memory) and a LOW indicates Receive (Read).
$\overline{\text{AEN}}$	I	**Address Enable:** $\overline{\text{AEN}}$ enables command outputs of the 8288 Bus Controller at least 115 ns after it becomes active (LOW). $\overline{\text{AEN}}$ going inactive immediately 3-states the command output drivers. $\overline{\text{AEN}}$ does not affect the I/O command lines if the 8288 is in the I/O Bus mode (IOB tied HIGH).
CEN	I	**Command Enable:** When this signal is LOW all 8288 command outputs and the DEN and $\overline{\text{PDEN}}$ control outputs are forced to their inactive state. When this signal is HIGH, these same outputs are enabled.
IOB	I	**Input/Output Bus Mode:** When the IOB is strapped HIGH the 8288 functions in the I/O Bus mode. When it is strapped LOW, the 8288 functions in the System Bus mode. (See sections on I/O Bus and System Bus modes).
$\overline{\text{AIOWC}}$	O	**Advanced I/O Write Command:** The $\overline{\text{AIOWC}}$ issues an I/O Write Command earlier in the machine cycle to give I/O devices an early indication of a write instruction. Its timing is the same as a read command signal. $\overline{\text{AIOWC}}$ is active LOW.
$\overline{\text{IOWC}}$	O	**I/O Write Command:** This command line instructs an I/O device to read the data on the data bus. This signal is active LOW.
$\overline{\text{IORC}}$	O	**I/O Read Command:** This command line instructs an I/O device to drive its data onto the data bus. This signal is active LOW.
$\overline{\text{AMWC}}$	O	**Advanced Memory Write Command:** The $\overline{\text{AMWC}}$ issues a memory write command earlier in the machine cycle to give memory devices an early indication of a write instruction. Its timing is the same as a read command signal. $\overline{\text{AMWC}}$ is active LOW.
$\overline{\text{MWTC}}$	O	**Memory Write Command:** This command line instructs the memory to record the data present on the data bus. This signal is active LOW.
$\overline{\text{MRDC}}$	O	**Memory Read Command:** This command line instructs the memory to drive its data onto the data bus. This signal is active LOW.
$\overline{\text{INTA}}$	O	**Interrupt Acknowledge:** This command line tells an interrupting device that its interrupt has been acknowledged and that it should drive vectoring information onto the data bus. This signal is active LOW.
MCE/$\overline{\text{PDEN}}$	O	This is a dual function pin. **MCE (IOB is tied LOW):** Master Cascade Enable occurs during an interrupt sequence and serves to read a Cascade Address from a master PIC (Priority Interrupt Controller) onto the data bus. The MCE signal is active HIGH. **$\overline{\text{PDEN}}$ (IOB is tied HIGH):** Peripheral Data Enable enables the data bus transceiver for the I/O bus that DEN performs for the system bus. $\overline{\text{PDEN}}$ is active LOW.

FUNCTIONAL DESCRIPTION

Command and Control Logic

The command logic decodes the three 8086, 8088 or 8089 CPU status lines ($\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$) to determine what command is to be issued.

This chart shows the meaning of each status "word".

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Processor State	8288 Command
0	0	0	Interrupt Acknowledge	$\overline{INTA}$
0	0	1	Read I/O Port	$\overline{IORC}$
0	1	0	Write I/O Port	$\overline{IOWC}$, $\overline{AIOWC}$
0	1	1	Halt	None
1	0	0	Code Access	$\overline{MRDC}$
1	0	1	Read Memory	$\overline{MRDC}$
1	1	0	Write Memory	$\overline{MWTC}$, $\overline{AMWC}$
1	1	1	Passive	None

The command is issued in one of two ways dependent on the mode of the 8288 Bus Controller.

I/O Bus Mode—The 8288 is in the I/O Bus mode if the IOB pin is strapped HIGH. In the I/O Bus mode all I/O command lines ($\overline{IORC}$, $\overline{IOWC}$, $\overline{AIOWC}$, $\overline{INTA}$) are always enabled (i.e., not dependent on $\overline{AEN}$). When an I/O command is initiated by the processor, the 8288 immediately activates the command lines using $\overline{PDEN}$ and DT/$\overline{R}$ to control the I/O bus transceiver. The I/O command lines should not be used to control the system bus in this configuration because no arbitration is present. This mode allows one 8288 Bus Controller to handle two external busses. No waiting is involved when the CPU wants to gain access to the I/O bus. Normal memory access requires a "Bus Ready" signal ($\overline{AEN}$ LOW) before it will proceed. It is advantageous to use the IOB mode if I/O or peripherals dedicated to one processor exist in a multi-processor system.

System Bus Mode—The 8288 is in the System Bus mode if the IOB pin is strapped LOW. In this mode no command is issued until 155 ns after the $\overline{AEN}$ Line is activated (LOW). This mode assumes bus arbitration logic will inform the bus controller (on the $\overline{AEN}$ line) when the bus is free for use. Both memory and I/O commands wait for bus arbitration. This mode is used when only one bus exists. Here, both I/O and memory are shared by more than one processor.

COMMAND OUTPUTS

The advanced write commands are made available to initiate write procedures early in the machine cycle. This signal can be used to prevent the processor from entering an unnecessary wait state.

The command outputs are:

$\overline{MRDC}$ —Memory Read Command
$\overline{MWTC}$ —Memory Write Command
$\overline{IORC}$ —I/O Read Command
$\overline{IOWC}$ —I/O Write Command
$\overline{AMWC}$ —Advanced Memory Write Command
$\overline{AIOWC}$ —Advanced I/O Write Command
$\overline{INTA}$ —Interrupt Acknowledge

$\overline{INTA}$ (Interrupt Acknowledge) acts as an I/O read during an interrupt ccle. Its purpose is to inform an interrupting device that its interrupt is being acknowledged and that it should place vectoring information onto the data bus.

CONTROL OUTPUTS

The control outputs of the 8288 are Data Enable (DEN), Data Transmit/Receive (DT/$\overline{R}$) and Master Cascade Enable/Peripheral Data Enable (MCE/$\overline{PDEN}$). The DEN signal determines when the external bus should be enabled onto the local bus and the DT/$\overline{R}$ determines the direction of data transfer. These two signals usually go to the chip select and direction pins of a transceiver.

The MCE/$\overline{PDEN}$ pin changes function with the two modes of the 8288. When the 8288 is in the IOB mode (IOB HIGH) the $\overline{PDEN}$ signal serves as a dedicated data enable signal for the I/O or Peripheral System bus.

INTERRUPT ACKNOWLEDGE AND MCE

The MCE signal is used during an interrupt acknowledge cycle if the 8288 is in the System Bus mode (IOB LOW). During any interrupt sequence there are two interrupt acknowledge cycles that occur back to back. During the first interrupt cycle no data or address transfers take place. Logic should be provided to mask off MCE during this cycle. Just before the second cycle begins the MCE signal gates a master Priority Interrupt Controller's (PIC) cascade address onto the processor's local bus where ALE (Address Latch Enable) strobes it into the address latches. On the leading edge of the second interrupt cycle the addressed slave PIC gates an interrupt vector onto the system data bus where it is read by the processor.

If the system contains only one PIC, the MCE signal is not used. In this case the second Interrupt Acknowledge signal gates the interrupt vector onto the processor bus.

ADDRESS LATCH ENABLE AND HALT

Address Latch Enable (ALE) occurs during each machine cycle and serves to strobe the current address into the address latches. ALE also serves to strobe the status ($\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$) into a latch for halt state decoding.

COMMAND ENABLE

The Command Enable (CEN) input acts as a command qualifier for the 8288. If the CEN pin is high the 8288 functions normally. If the CEN pin is pulled LOW, all command lines are held in their inactive state (not 3-state). This feature can be used to implement memory partitioning and to eliminate address conflicts between system bus devices and resident bus devices.

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... 0°C to 70°C
Storage Temperature .... −65°C to +150°C
All Output and Supply Voltages .... −0.5V to +7V
All Input Voltages .... −1.0V to +5.5V
Power Dissipation .... 1.5 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (V_{CC} = 5V ± 10%, T_A = 0°C to 70°C)

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_C	Input Clamp Voltage		−1	V	$I_C = -5$ mA
I_{CC}	Power Supply Current		230	mA	
I_F	Forward Input Current		−0.7	mA	$V_F = 0.45V$
I_R	Reverse Input Current		50	μA	$V_R = V_{CC}$
V_{OL}	Output Low Voltage Command Outputs Control Outputs		 0.5 0.5	 V V	 $I_{OL} = 32$ mA $I_{OL} = 16$ mA
V_{OH}	Output High Voltage Command Outputs Control Outputs	 2.4 2.4		 V V	 $I_{OH} = -5$ mA $I_{OH} = -1$ mA
V_{IL}	Input Low Voltage		0.8	V	
V_{IH}	Input High Voltage	2.0		V	
I_{OFF}	Output Off Current		100	μA	V_{OFF} = 0.4 to 5.25V

A.C. CHARACTERISTICS (V_{CC} = 5V ± 10%, T_A = 0°C to 70°C)*

TIMING REQUIREMENTS

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
TCLCL	CLK Cycle Period	100		ns	
TCLCH	CLK Low Time	50		ns	
TCHCL	CLK High Time	30		ns	
TSVCH	Status Active Setup Time	35		ns	
TCHSV	Status Inactive Hold Time	10		ns	
TSHCL	Status Inactive Setup Time	35		ns	
TCLSH	Status Active Hold Time	10		ns	

*Note: For Extended Temperature EXPRESS the Preliminary Values are TCLCL = 125; TCLCH = 50; TCHCL = 30; TCVNX = 50; TCLLH, TCLMCH = 25; TSVLH, TSVMCH = 25.

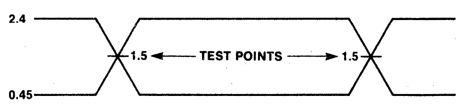
A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
TCVNV	Control Active Delay	5	45	ns	
TCVNX	Control Inactive Delay	10	45	ns	
TCLLH, TCLMCH	ALE MCE Active Delay (from CLK)		20	ns	
TMHNL	Command to DEN Delay (NOTE 1)	TCLCH-5		ns	
TSVLH, TSVMCH	ALE MCE Active Delay (from Status)		20	ns	
TCHLL	ALE Inactive Delay	4	15	ns	$\overline{MRDC}$, $\overline{IORC}$, $\overline{MWTC}$, $\overline{IOWC}$, $\overline{INTA}$, $\overline{AMWC}$, $\overline{AIOWC}$: I_{OL} = 32 mA, I_{OH} = −5 mA, C_L = 300 pF
TCLML	Command Active Delay	10	35	ns	
TCLMH	Command Inactive Delay	10	35	ns	
TCHDTL	Direction Control Active Delay		50	ns	
TCHDTH	Direction Control Inactive Delay		30	ns	
TAELCH	Command Enable Time		40	ns	
TAEHCZ	Command Disable Time		40	ns	
TAELCV	Enable Delay Time	115	200	ns	
TAEVNV	AEN to DEN		20	ns	Other: I_{OL} = 16 mA, I_{OH} = −1 mA, C_L = 80 pF
TCEVNV	CEN to DEN, PDEN		25	ns	
TCELRH	CEN to Command		TCLML	ns	
TOLOH	Output, Rise Time		20	ns	From 0.8V to 2.0V
TOHOL	Output, Fall Time		12	ns	From 2.0V to 0.8V

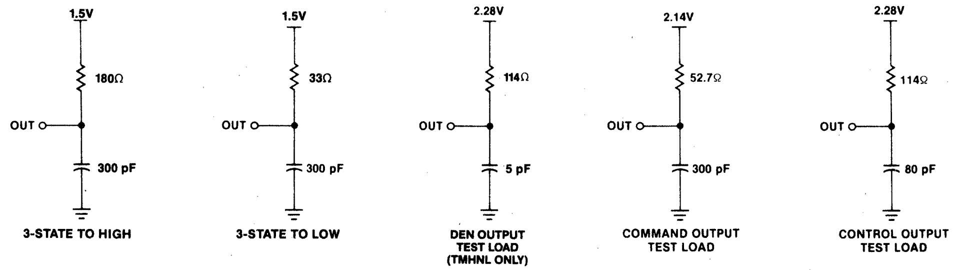
Note 1. TMHNL is tested with DEN C_L = 5 pF

A.C. TESTING INPUT, OUTPUT WAVEFORM

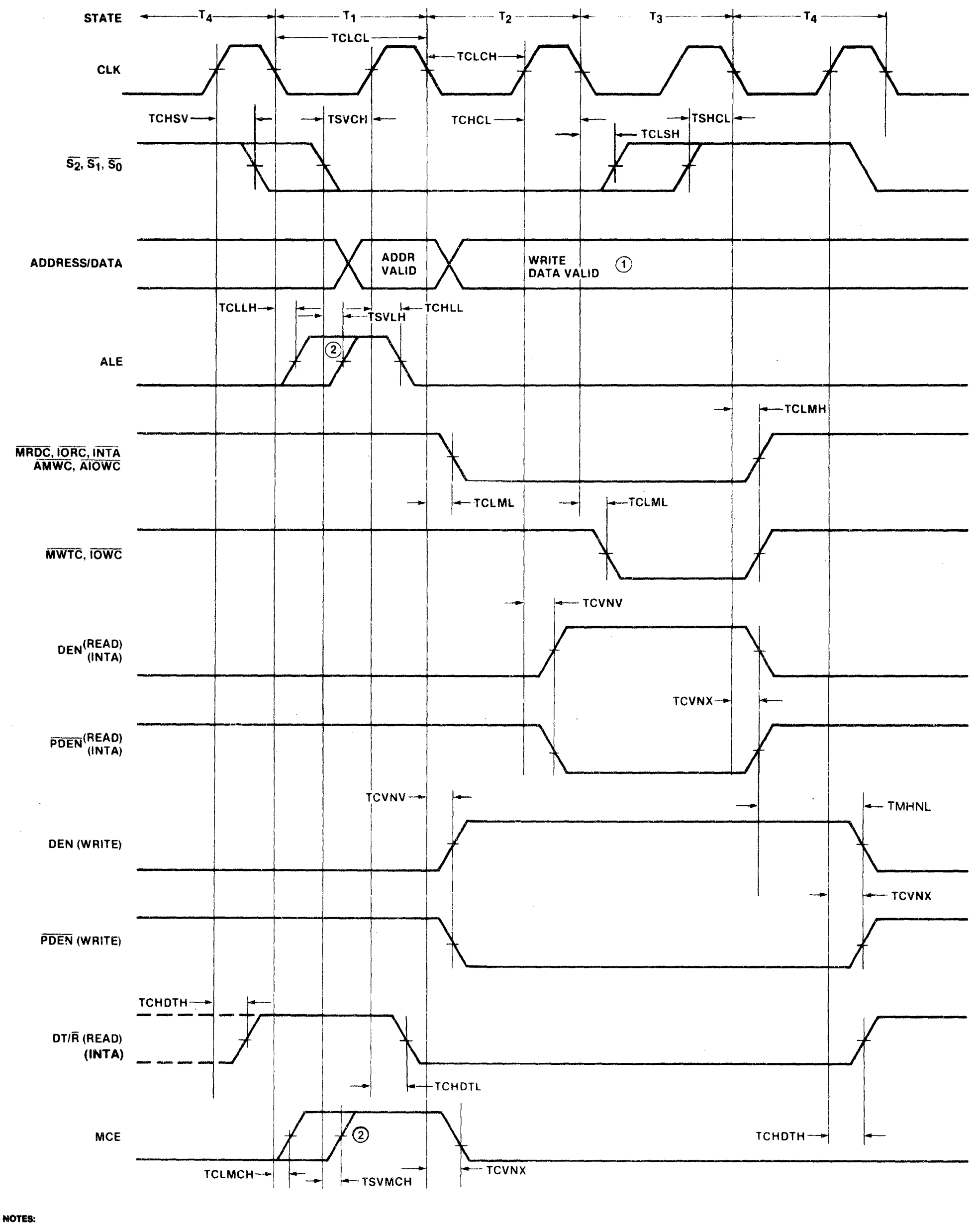


A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." THE CLOCK IS DRIVEN AT 4.3V AND 0.25V. TIMING MEASUREMENTS ARE MADE AT 1.5V FOR BOTH A LOGIC "1" AND "0." INPUT RISE AND FALL TIMES ARE 5 ± 2 NS., MEASURED BETWEEN 0.8V AND 2.0V

TEST LOAD CIRCUITS—3-STATE COMMAND OUTPUT TEST LOAD



WAVEFORMS

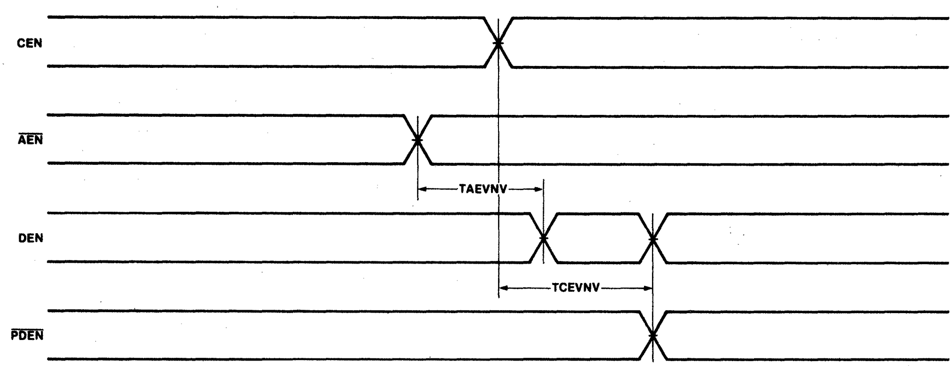


NOTES:

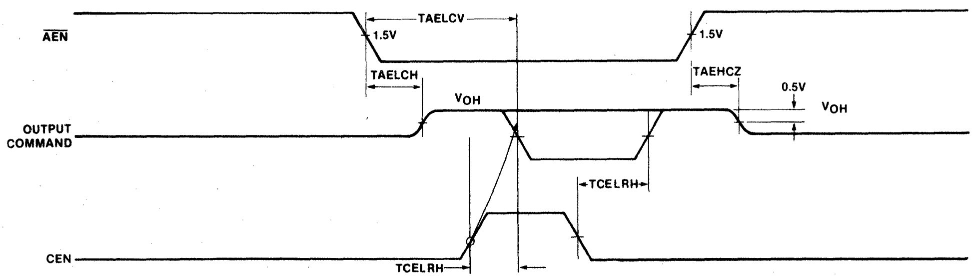
1. ADDRESS/DATA BUS IS SHOWN ONLY FOR REFERENCE PURPOSES.
2. LEADING EDGE OF ALE AND MCE IS DETERMINED BY THE FALLING EDGE OF CLK OR STATUS GOING ACTIVE, WHICHEVER OCCURS LAST.
3. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS SPECIFIED OTHERWISE.

WAVEFORMS (Continued)

DEN, PDEN QUALIFICATION TIMING



ADDRESS ENABLE (AEN) TIMING (3-STATE ENABLE/DISABLE)



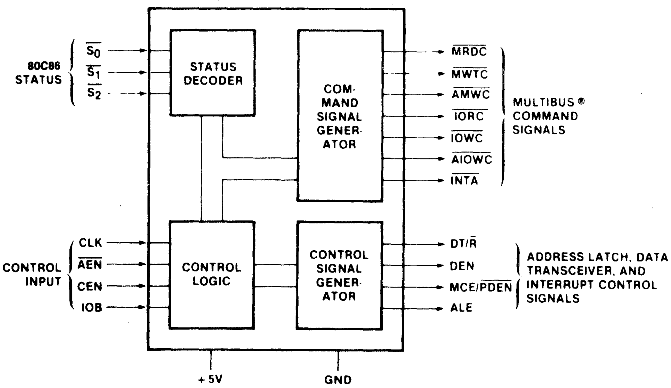
NOTE: CEN must be low or valid prior to T2 to prevent the command from being generated

PRELIMINARY

82C88
CHMOS BUS CONTROLLER FOR 80C86, 80C88 PROCESSORS

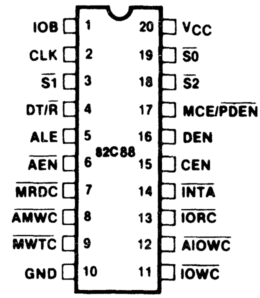
- **Pin Compatible with Bipolar 8288**
- **Low Power Operation**
  - **— I_{CCS} = 10 μA**
  - **— I_{CC} = 1 mA/MHz**
- **Provides Advanced Commands for Multi-Master Busses**
- **3-State Command Output Drivers**
- **High Drive Capability**
- **Configurable for Use with an I/O Bus**
- **Single 5V Power Supply**
- **Will Be Available in 20-Lead Plastic DIP and 20-Lead PLCC Packages**
  (See Packaging Spec., Order #231369)

The Intel 82C88 is a high performance CHMOS version of the 8288 bipolar bus controller. The 82C88 provides command and control timing generation for 80C86/88, 8086/88 and iAPX 186 systems. Static CHMOS circuit design insures low operating power. 8 MHz speed optimizes system performance and the 82C88 high output drive capability eliminates the need for additional bus drivers.



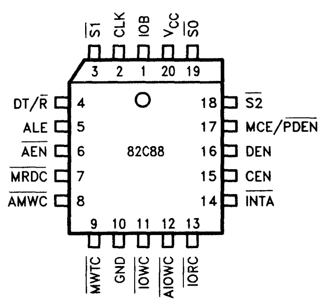
231199–1

Figure 1. Block Diagram



231199–2

Figure 2a. 82C88 20-Lead DIP Configuration



231199–8

Figure 2b. 82C88 20-Lead PLCC Configuration

Order Number: 231199-002

Table 1. Pin Description

Symbol	Type	Name and Function
V_{CC}		**POWER:** +5V supply.
GND		**GROUND.**
$\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$	I	**STATUS INPUT PINS:** These pins are the status input pins from the 80C86, 80C88 or 8089 processors. The 82C88 decodes these inputs to generate command and control signals at the appropriate time. When these pins are not in use (passive) they are all HIGH. (See chart under Command and Control Logic.) Active "Bus Hold" circuits hold these lines HIGH when no other driving source is present.
CLK	I	**CLOCK:** This is a clock signal from the 82C84 clock generator and serves to establish when command and control signals are generated.
ALE	O	**ADDRESS LATCH ENABLE:** This signal serves to strobe an address into the address latches. This signal is active HIGH and latching occurs on the falling (HIGH to LOW) transition. ALE is intended for use with transparent D type latches.
DEN	O	**DATA ENABLE:** This signal serves to enable data transceivers onto either the local or system data bus. This signal is active HIGH.
DT/$\overline{R}$	O	**DATA TRANSMIT/RECEIVE:** This signal establishes the direction of data flow through the transceivers. A HIGH on this line indicates Transmit (write to I/O or memory) and a LOW indicates Receive (Read).
$\overline{AEN}$	I	**ADDRESS ENABLE:** $\overline{AEN}$ enables command outputs of the 82C88 Bus Controller at least 110 ns after it becomes active (LOW). $\overline{AEN}$ going inactive immediately 3-states the command output drivers. $\overline{AEN}$ does not affect the I/O command lines if the 82C88 is in the I/O Bus mode (IOB tied HIGH).
CEN	I	**COMMAND ENABLE:** When this signal is LOW all 82C88 command outputs and the DEN and $\overline{PDEN}$ control outputs are forced to their inactive state. When this signal is HIGH, these same outputs are enabled.
IOB	I	**INPUT/OUTPUT BUS MODE:** When the IOB is strapped HIGH the 82C88 functions in the I/O Bus mode. When it is strapped LOW, the 82C88 functions in the System Bus mode. (See sections on I/O Bus and System Bus modes).
$\overline{AIOWC}$	O	**ADVANCED I/O WRITE COMMAND:** The $\overline{AIOWC}$ issues an I/O Write Command earlier in the machine cycle to give I/O devices an early indication of a write instruction. Its timing is the same as a read command signal. $\overline{AIOWC}$ is active LOW.
$\overline{IOWC}$	O	**I/O WRITE COMMAND:** This command line instructs an I/O device to read the data on the data bus. This signal is active LOW.
$\overline{IORC}$	O	**I/O READ COMMAND:** This command line instructs an I/O device to drive its data onto the data bus. This signal is active LOW.
$\overline{AMWC}$	O	**ADVANCED MEMORY WRITE COMMAND:** The $\overline{AMWC}$ issues a memory write command earlier in the machine cycle to give memory devices an early indication of a write instruction. Its timing is the same as read command signal. $\overline{AMWC}$ is active LOW.
$\overline{MWTC}$	O	**MEMORY WRITE COMMAND:** This command line instructs the memory to record the data present on the data bus. This signal is active LOW.
$\overline{MRDC}$	O	**MEMORY READ COMMAND:** This command line instructs the memory to drive its data onto the data bus. This signal is active LOW.
$\overline{INTA}$	O	**INTERRUPT ACKNOWLEDGE:** This command line tells an interrupting device that its interrupt has been acknowledged and that it should drive vectoring information onto the data bus. This signal is active LOW.
MCE/$\overline{PDEN}$	O	This is a dual function pin. **MCE (IOB IS TIED LOW):** Master Cascade Enable occurs during an interrupt sequence and serves to read a Cascade Address from a master PIC (Priority Interrupt Controller) onto the data bus. The MCE signal is active HIGH. **$\overline{PDEN}$ (IOB IS TIED HIGH):** Peripheral Data Enable enables the data bus transceiver for the I/O bus that DEN performs for the system bus. $\overline{PDEN}$ is active LOW.

FUNCTIONAL DESCRIPTION

Command and Control Logic

The command logic decodes the three 80C86, 80C88 or 8089 CPU status lines ($\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$) to determine what command is to be issued.

This chart shows the meaning of each status "word".

$\overline{S_2}$ $\overline{S_1}$ $\overline{S_0}$	Processor State	82C88 Command
0 0 0	Interrupt Acknowledge	$\overline{\text{INTA}}$
0 0 1	Read I/O Port	$\overline{\text{IORC}}$
0 1 0	Write I/O Port	$\overline{\text{IOWC}}$, $\overline{\text{AIOWC}}$
0 1 1	Halt	None
1 0 0	Code Access	$\overline{\text{MRDC}}$
1 0 1	Read Memory	$\overline{\text{MRDC}}$
1 1 0	Write Memory	$\overline{\text{MWTC}}$, $\overline{\text{AMWC}}$
1 1 1	Passive	None

The command is issued in one of two ways dependent on the mode of the 82C88 Bus Controller.

I/O Bus Mode — The 82C88 is in the I/O Bus mode if the IOB pin is strapped HIGH. In the I/O Bus mode all I/O command lines ($\overline{\text{IORC}}$, $\overline{\text{IOWC}}$, $\overline{\text{AIOWC}}$, $\overline{\text{INTA}}$) are always enabled (i.e., not dependent on $\overline{\text{AEN}}$). When an I/O command is initiated by the processor, the 82C88 immediately activates the command lines, using $\overline{\text{PDEN}}$ and DT/$\overline{\text{R}}$ to control the I/O bus transceiver. The I/O command lines should not be used to control the system bus in this configuration because no arbitration is present. This mode allows one 82C88 Bus Controller to handle two external busses. No waiting is involved when the CPU wants to gain access to the I/O bus. Normal memory access requires a "Bus Ready" signal ($\overline{\text{AEN}}$ LOW) before it will proceed. It is advantageous to use the IOB mode if I/O or peripherals dedicated to one processor exist in a multi-processor system.

System Bus Mode — The 82C88 in the System Bus mode if the IOB pin is strapped LOW. In this mode no command is issued until 110 ns after the $\overline{\text{AEN}}$ Line is activated (LOW). This mode assumes bus arbitration logic will inform the bus controller (on the $\overline{\text{AEN}}$ line) when the bus is free for use. Both memory and I/O commands wait for bus arbitration. This mode is used when only one bus exists. Here, both I/O and memory are shared by more than one processor.

COMMAND OUTPUTS

The advanced write commands are made available to initiate write procedures early in the machine cycle. This signal can be used to prevent the processor from entering an unnecessary wait state.

The command outputs are:

- $\overline{\text{MRDC}}$ — Memory Read Command
- $\overline{\text{MWTC}}$ — Memory Write Command
- $\overline{\text{IORC}}$ — I/O Read Command
- $\overline{\text{IOWC}}$ — I/O Write Command
- $\overline{\text{AMWC}}$ — Advanced Memory Write Command
- $\overline{\text{AIOWC}}$ — Advanced I/O Write Command
- $\overline{\text{INTA}}$ — Interrupt Acknowledge

$\overline{\text{INTA}}$ (Interrupt Acknowledge) acts as an I/O read during an interrupt cycle. Its purpose is to inform an interrupting device that its interrupt is being acknowledged and that it should place vectoring information onto the data bus.

CONTROL OUTPUTS

The control outputs of the 82C88 are Data Enable (DEN), Data Transmit/Receive (DT/$\overline{\text{R}}$) and Master Cascade Enable/Peripheral Data Enable (MCE/$\overline{\text{PDEN}}$). The DEN signal determines when the external bus should be enabled onto the local bus and the DT/$\overline{\text{R}}$ determines the direction of data transfer. These two signals usually go to the chip select and direction pins of a transceiver.

The MCE/$\overline{\text{PDEN}}$ pin changes function with the two modes of the 82C88. When the 82C88 is in the IOB mode (IOB HIGH) the $\overline{\text{PDEN}}$ signal serves as a dedicated data enable signal for the I/O or Peripheral System bus.

INTERRUPT ACKNOWLEDGE AND MCE

The MCE signal is used during an interrupt acknowledge cycle if the 82C88 is in the System Bus mode (IOB LOW). During any interrupt sequence there are two interrupt acknowledge cycles that occur back to back. During the first interrupt cycle no data or address transfers take place. Logic should be provided to mask off MCE during this cycle. Just before the second cycle begins the MCE signal gates a master Priority Interrupt Controller's (PIC) cascade address onto the processor's local bus where ALE (Address Latch Enable) strobes it into the address latches. On the leading edge of the second interrupt cycle the addressed slave PIC gates an interrupt vector onto the system data bus where it is read by the processor.

If the system contains only one PIC, the MCE signal is not used. In this case the second interrupt Ac-

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... 0°C to 70°C
Storage Temperature .... −65°C to +150°C
Supply Voltage
 (with Respect to GND............ −0.5V to 8.0V
All Input Voltages
 (with Respect to GND ..... −2.0V to V_{CC} + 0.5V
All Output Voltages
 (with Respect to GND ..... −0.5V to V_{CC} + 0.5V
Power Dissipation .... 1.0 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS (V_{CC} = 5V ±10%, T_A = 0°C to 70°C)

Symbol	Parameter	Min	Max	Unit	Test Conditions
I_{CC}	Operating Supply Current		8	mA	V_{IN} = V_{CC} or GND V_{CC} = 5.5V Outputs Unloaded
I_{CCS}	Standby Supply Current		10	μA	V_{IN} = V_{CC} or GND V_{CC} = 5.5V Outputs Unloaded
V_{IH}	Input High Voltage	2.0		V	
V_{IL}	Input Low Voltage		0.8	V	
V_{CH}	V_{IH} for Clock	0.7 V_{CC}		V	
V_{CL}	V_{IL} for Clock		0.2 V_{CC}	V	
I_{LI}	Input Leakage Current		±1.0	μA	0V ≤ V_{IN} ≤ V_{CC} (Notes 1, 2)
I_{BHH}	Input Leakage Current (Bus Hold High)	−50	−300	μA	V_{IN} = 2.0V (Notes 3, 4)
I_{BHHO}	Bus Hold High Overdrive	−600		μA	(Notes 3, 5)
I_{LO}	Output Leakage Current		±10	μA	0V ≤ V_{OUT} ≤ V_{CC}
V_{OL}	Output Low Voltage: Command Outputs Control Outputs		0.5 0.4	V	I_{OL} = 20 mA I_{OL} = 8 mA
V_{OH}	Output High Voltage: Command Outputs Control Outputs	3.0 V_{CC} − 0.4 3.0 V_{CC} − 0.4		V	I_{OH} = −8 mA I_{OH} = −2.5 mA I_{OH} = −4 mA I_{OH} = −2.5 mA
C_{IN}	Input Capacitance		5	pF	Freq. = 1 MHz Unmeasured pins at GND
C_{OUT}	Output Capacitance		15	pF	Freq. = 1 MHz Unmeasured pins at GND

NOTES:
1. Except $\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$.
2. During input leakage test, maximum input rise and fall time should be 15 ns between V_{CC} and GND.
3. $\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$ only.
4. Raise inputs to V_{CC}, then lower to 2.0V.
5. An external driver must sink at least I_{BHHO} to toggle a status line from HIGH to LOW.

knowledge signal gates the interrupt vector onto the processor bus.

ADDRESS LATCH ENABLE AND HALT

Address Latch Enable (ALE) occurs during each machine cycle and serves to strobe the current address into the address latches. ALE also serves to strobe the status ($\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$) into a latch for halt state decoding.

COMMAND ENABLE

The Command Enable (CEN) input acts as a command qualifier for the 82C88. If the CEN pin is high the 82C88 functions normally. If the CEN pin is pulled LOW, all command lines are held in their inactive state (not 3-state). This feature can be used to implement memory partitioning and to eliminate address conflicts between system bus devices and resident bus devices.

A.C. CHARACTERISTICS (V_{CC} = 5V ±10%, T_A = 0°C to 70°C)*

TIMING REQUIREMENTS

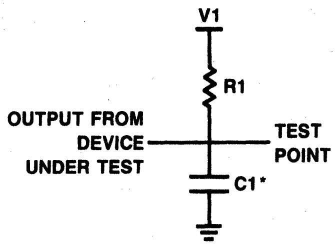
Symbol	Parameter	Min	Max	Units	Test Conditions
fc	CLK Frequency		8	MHz	
TCLCL	CLK Cycle Period	125		ns	
TCLCH	CLK Low Time	66		ns	
TCHCL	CLK High Time	40		ns	
TSVCH	Status Active Setup Time	35		ns	
TCHSV	Status Inactive Hold Time	10		ns	
TSHCL	Status Inactive Setup Time	35		ns	
TCLSH	Status Active Hold Time	10		ns	

TIMING RESPONSES

Symbol	Parameter	Min	Max	Units	Test Conditions**
TCVNV	Control Active Delay	5	45	ns	a
TCVNX	Control Inactive Delay	10	45	ns	a
TCLLH	ALE Active Delay (from CLK)		20	ns	a
TCLMCH	MCE Active Delay (from CLK)		25	ns	a
TSVLH	ALE Active Delay (from Status)		20	ns	a
TSVMCH	MCE Active Delay (from Status)		30	ns	a
TCHLL	ALE Inactive Delay	4	25	ns	a(Note 3)
TMHNL	Command Inactive to DEN Low Delay	TCLCH-5		ns	Command: b, DEN: e
TCLML	Command Active Delay	5	35	ns	b
TCLMH	Command Inactive Delay	5	35	ns	b
TCHDTL	Direction Control Active Delay		50	ns	a
TCHDTH	Direction Control Inactive Delay		30	ns	a
TAELCH	Command Enable Time		40	ns	c(Note 1)
TAEHCZ	Command Disable Time		40	ns	d(Note 2)
TAELCV	Enable Delay Time	110	250	ns	b
TAEVNV	$\overline{\text{AEN}}$ to DEN		25	ns	a
TCEVNV	CEN to DEN, $\overline{\text{PDEN}}$		25	ns	a
TCELRH	CEN to Command		TCLML +10	ns	b
TOLOH	Output, Rise Time		15	ns	From 0.8V to 2.0V
TOHOL	Output, Fall Time		15	ns	From 2.0V to 0.8V

NOTES:
1. TAELCH measurement is between 1.5V and 2.5V.
2. TAEHCZ measured at 0.5V change in V_{OUT}.
3. In 5 MHz 80C86/88 systems, minimum ALE HIGH time = TCLCL − (TCHSV(max) + TSVLH) + TCHLL(min) = 74 ns.

TEST LOAD CIRCUITS—3-STATE COMMAND OUTPUT TEST LOAD



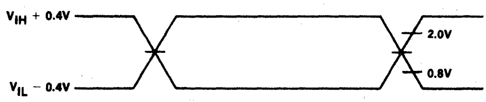
231199-3

*Includes stray and jig capacitance

**Test Condition Definition Table

Test Condition	I_{OH}	I_{OL}	V1	R1	C1
a	−4.0 mA	+8.0 mA	2.13V	220Ω	80 pf
b	−8.0 mA	+20.0 mA	2.29V	91Ω	300 pf
c	−8.0 mA		1.5V	187Ω	300 pf
d	−8.0 mA		1.5V	187Ω	50 pf
e	−1.0 μA	+1.0 μA	2.13V	870 kΩ	30 pf

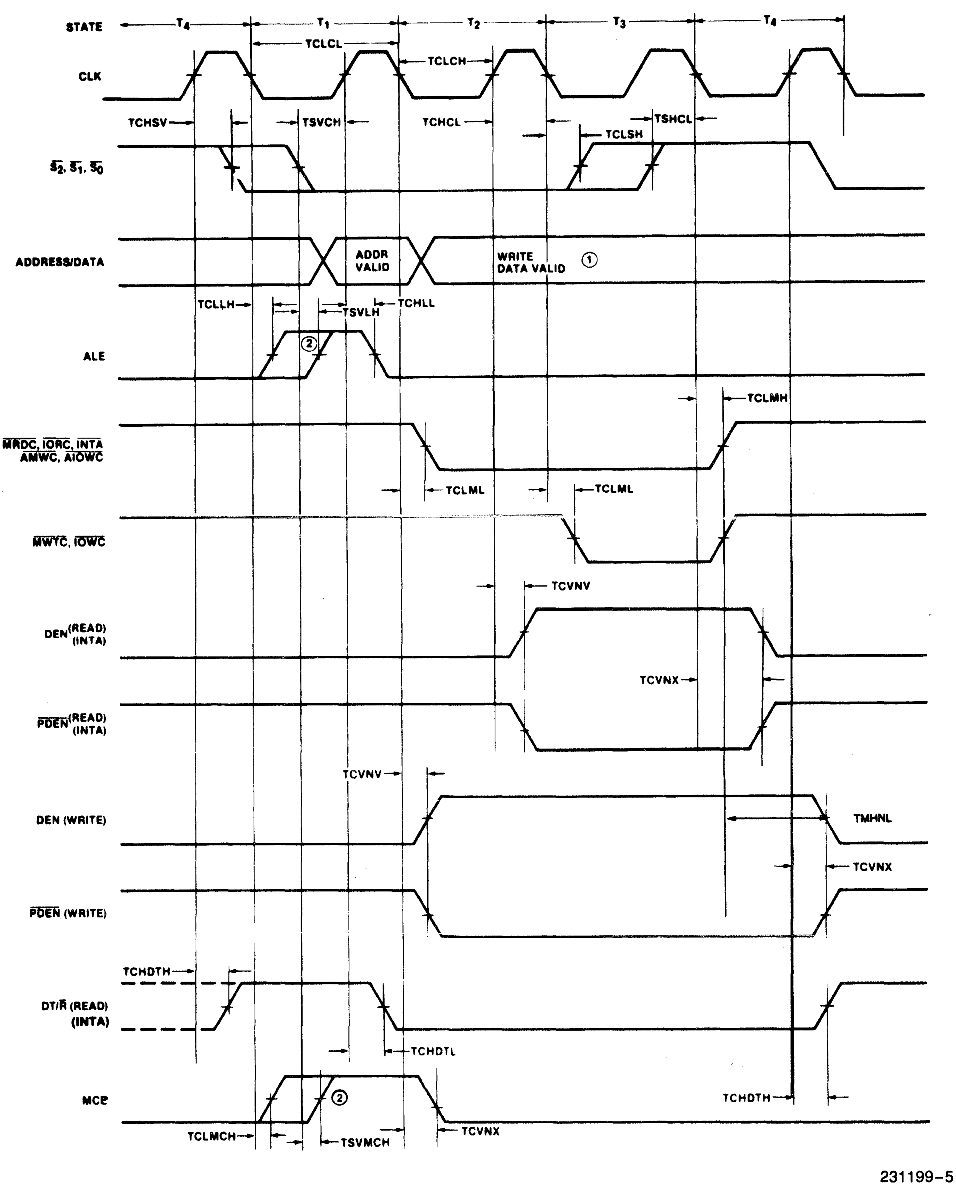
A.C. TESTING INPUT, OUTPUT WAVEFORM



231199-4

A.C. Testing: Inputs are driven at V_{IH} + 0.4V for a logic "1" and V_{IL} − 0.4V for a logic "0". The clock is driven at 3.9V and 0.4V. Timing measurements are made at 2.0V and 0.8V. Input rise and fall times should be ≤ 15 ns measured between 0.8V and 2.0V.

WAVEFORMS

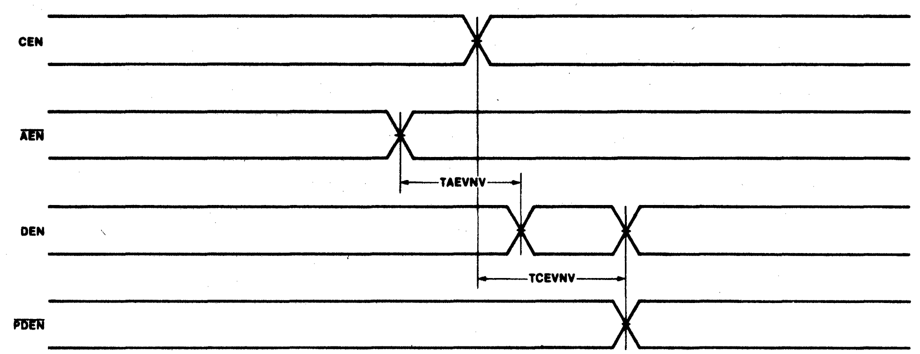


NOTES:

1. Address/Data Bus is shown only for reference purposes.
2. Leading edge of ALE and MCE is determined by the falling edge of CLK or Status going active, whichever occurs last.

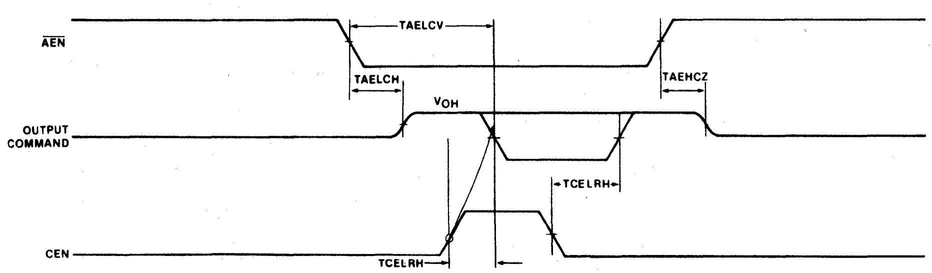
WAVEFORMS (Continued)

DEN, PDEN QUALIFICATION TIMING



231199-6

ADDRESS ENABLE (AEN) TIMING (3-STATE ENABLE/DISABLE)



NOTE:
CEN must be low or valid prior to T2 to prevent the command from being generated.

231199-7

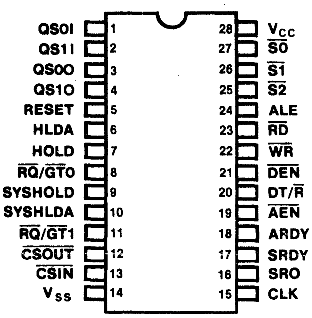
intel®

PRELIMINARY

82188 INTEGRATED BUS CONTROLLER FOR 8086, 8088, 80186, 80188 PROCESSORS

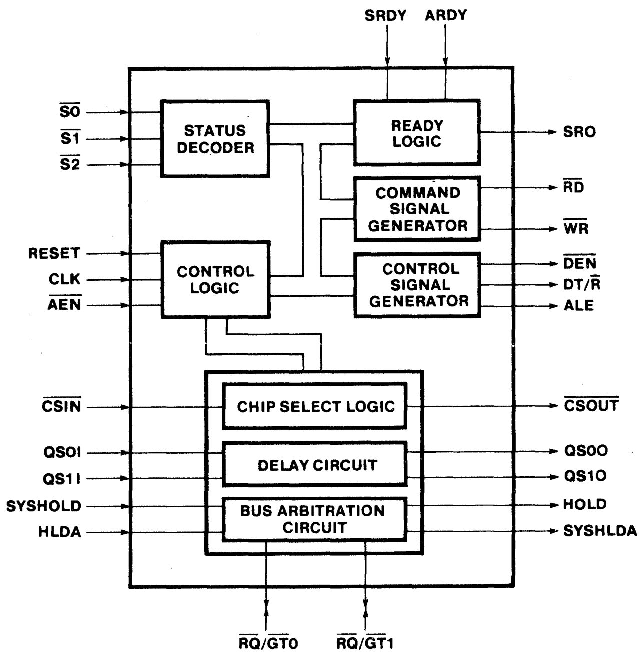
- **Provides Flexibility in System Configurations**
  - **—Supports 8087 Numerics Coprocessor in 8 MHz 80186 and 80188 Systems**
  - **—Provides a Low-cost Interface for 8086, 8088 Systems to an 82586 LAN Coprocessor or 82730 Text Coprocessor**
- **Facilitates Interface to one or more Multimaster Busses**
- **Supports Multiprocessor, Local Bus Systems**
- **Allows use of 80186, 80188 High-Integration Features**
- **3-State, Command Output Drivers**
- **Available in EXPRESS**
  - **—Standard Temperature Range**
  - **—Extended Temperature Range**
- **Available in Plastic DIP or Cerdip Package**
  (See Packaging Spec., Order #231369)

The 82188 Integrated Bus Controller (IBC) is a 28-pin HMOS III component for use with 80186, 80188, 8086 and 8088 systems. The IBC provides command and control timing signals plus a configurable $\overline{RQ}/\overline{GT}$ ⟷ HOLD-HLDA converter. The device may be used to interface an 8087 Numerics Coprocessor with an 80186 or 80188 Processor. Also, an 82586 Local Area Network (LAN) Coprocessor or 82730 Text Coprocessor may be interfaced to an 8086 or 8088 with the IBC.



231051-1

Figure 1. 82188 Pin Configuration



231051-2

Figure 2. 82188 Block Diagram

Order Number: 231051-003

PIN DESCRIPTIONS

Symbol	Pin No.	Type	Name and Function
$\overline{S0}$ $\overline{S1}$ $\overline{S2}$	27 26 25	I	**Status Input Pins** $\overline{S0}$–$\overline{S2}$ correspond to the status pins of the CPU. The 82188 uses the status lines to detect and identify the processor bus cycles. The 82188 decodes $\overline{S0}$–$\overline{S2}$ to generate the command and control signals. $\overline{S0}$–$\overline{S2}$ are also used to insert 3 wait states into the SRO line during the first 256 80186 bus cycles after RESET. A HIGH input on all three lines indicates that no bus activity is taking place. The status input lines contain weak internal pull-up devices.

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Bus Cycle Initiated
0	0	0	interrupt acknowledge
0	0	1	read I/O
0	1	0	write I/O
0	1	1	halt
1	0	0	instruction fetch
1	0	1	read data from memory
1	1	0	write data to memory
1	1	1	passive (no bus cycle)

Symbol	Pin No.	Type	Name and Function
CLK	15	I	**CLOCK** CLK is the clock signal generated by the CPU or clock generator device. CLK edges establish when signals are sampled and generated.
RESET	5	I	**RESET** RESET is a level triggered signal that corresponds to the system reset signal. The signal initializes an internal bus cycle counter, thus enabling the 82188 to insert internally generated wait states into the SRO signal during system initialization. The 82188 mode is also determined during RESET. $\overline{RD}$, $\overline{WR}$, and $\overline{DEN}$ are driven HIGH during RESET regardless of $\overline{AEN}$. RESET is active HIGH.
$\overline{AEN}$	19	I	**Address Enable** This signal enables the system command lines when active. If $\overline{AEN}$ is inactive (HIGH), $\overline{RD}$, $\overline{WR}$, and $\overline{DEN}$ will be tri-stated and ALE will be driven LOW (DT/$\overline{R}$ will not be effected). $\overline{AEN}$ is an asynchronous signal and is active LOW.
ALE	24	O	**Address Latch Enable** This signal is used to strobe an address into address latches. ALE is active HIGH and latch should occur on the HIGH to LOW transition. ALE is intended for use with transparent D-type latches.
$\overline{DEN}$	21	O	**Data Enable** This signal is used to enable data transceivers located on either the local or system data bus. The signal is active LOW. $\overline{DEN}$ is tri-stated when $\overline{AEN}$ is inactive.
DT/$\overline{R}$	20	O	**Data TRANSMIT/RECEIVE** This signal establishes the direction of data flow through the data transceivers. A HIGH on this line indicates TRANSMIT (write to I/O or memory) and a LOW indicates RECEIVE (Read from I/O or memory).

PIN DESCRIPTIONS (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{RD}$	23	O	**READ** This signal instructs an I/O or memory device to drive its data onto the data bus. The $\overline{RD}$ signal is similiar to the $\overline{RD}$ signal of the 80186(80188) in Non-Queue-Status Mode. $\overline{RD}$ is active LOW and is tri-stated when $\overline{AEN}$ is inactive.
$\overline{WR}$	22	O	**WRITE** This signal instructs an I/O or memory device to record the data presented on the data bus. The $\overline{WR}$ signal is similiar to the $\overline{WR}$ signal of the 80186(80188) in Non-Queue-Status Mode. $\overline{WR}$ is active LOW and is tri-stated when $\overline{AEN}$ is inactive.
HOLD	7	O	**HOLD** The HOLD signal is used to request bus control from the 80186 or 80188. The request can come from either the 8087 ($\overline{RQ}/\overline{GT0}$) or from the third processor (SYSHOLD). The signal is active HIGH.
HLDA	6	I	**HOLD Acknowledge** 80186 MODE–This line serves to translate the HLDA output of the 80186(80188) to the appropriate signal of the device requesting the bus. HLDA going active (HIGH) indicates that the 80186 has relinquished the bus. If the requesting device is the 8087, HLDA will be translated into the grant pulse of the $\overline{RQ}/\overline{GT0}$ line. If the requesting device is the optional third processor, HLDA will be routed into the SYSHLDA line. This pin also determines the mode in which the 82188 will operate. If this line is HIGH during the falling edge of RESET, the 82188 will enter the 8086 mode. If LOW, the 82188 will enter the 80186 mode. For 8086 mode, this pin should be strapped to V_{CC}.
$\overline{RQ}/\overline{GT0}$	8	I/O	**Request/Grant O** $\overline{RQ}/\overline{GT0}$ is connected to $\overline{RQ}/\overline{GT0}$ of the 8087 Numeric Coprocessor. When initiated by the 8087, $\overline{RQ}/\overline{GT0}$ will be translated to HOLD-HLDA to acquire the bus from the 80186(80188). This line is bidirectional, and is active LOW. $\overline{RQ}/\overline{GT0}$ has a weak internal pull-up device to prevent erroneous request/grant signals.
$\overline{RQ}/\overline{GT1}$	11	I/O	**Request/Grant 1** 80186 Mode–In 80186 Mode, $\overline{RQ}/\overline{GT1}$ allows a third processor to take control of the local bus when the 8087 has bus control. For a HOLD-HLDA type third processor, the 82188's $\overline{RQ}/\overline{GT1}$ line should be connected to the $\overline{RQ}/\overline{GT1}$ line of the 8087. 8086 MODE–In 8086 Mode, $\overline{RQ}/\overline{GT1}$ is connected to either $\overline{RQ}/\overline{GT0}$ or $\overline{RQ}/\overline{GT1}$ of the 8086. $\overline{RQ}/\overline{GT1}$ will start its request/grant sequence when the SYSHOLD line goes active. In 8086 Mode, $\overline{RQ}/\overline{GT1}$ is used to gain bus control from the 8086 or 8088. $\overline{RQ}/\overline{GT1}$ is a bidirectional line and is active LOW. This line has a weak internal pull-up device to prevent erroneous request/grant signals.

PIN DESCRIPTIONS (Continued)

Symbol	Pin No.	Type	Name and Function
SYSHOLD	9	I	**System Hold** 80186 MODE–SYSHOLD serves as a hold input for an optional third processor in an 80186(80188)-8087 system. If the 80186(80188) has bus control, SYSHOLD will be routed to HOLD to gain control of the bus. If the 8087 has bus control, SYSHOLD will be translated to $\overline{RQ}/\overline{GT}1$ to gain control of the bus. 8086 MODE–SYSHOLD serves as a hold input for a coprocessor in an 8086 or 8088 system. SYSHOLD is translated to $\overline{RQ}/\overline{GT}1$ of the 82188 to allow the coprocessor to take control of the bus. SYSHOLD may be an asynchronous signal.
SYSHLDA	10	O	**System Hold Acknowledge** SYSHLDA serves as a hold acknowledge line to the processor or coprocessor connected to it. The device connected to the SYSHOLD-SYSHLDA lines is allowed the bus when SYSHLDA goes active (HIGH).
SRDY	17	I	**Synchronous Ready** The SRDY input serves the same function as SRDY of the 80186(80188). The 82188 combines SRDY with ARDY to form a synchronized ready output signal (SRO). SRDY must be synchronized external to the 82188 and is active HIGH. If tied to V_{CC}, SRO will remain active (HIGH) after the first 256 80186 cycles following RESET. If only ARDY is to be used, SRDY should be tied LOW.
ARDY	18	I	**Asynchronous Ready** The ARDY input serves the same function as ARDY of the 80186(80188). ARDY may be an asynchronous input, and is active HIGH. Only the rising edge of ARDY is synchronized by the 82188. The falling edge must be synchronized external to the 82188. If connected to V_{CC}, SRO will remain active (HIGH) after the first 256 80186 bus cycles following RESET. If only SRDY is to be used, ARDY should be connected LOW.
SRO	16	O	**Synchronous READY Output** SRO provides a synchronized READY signal which may be interfaced directly with the SRDY of the 80186(80188) and READY of the 8087. The SRO signal is an accumulation of the synchronized ARDY signal, the SRDY signal, and the internally generated wait state signal.
QS0I QS1I	1 2	I	**Queue-Status Inputs** QS0I, QS1I are connected to the Queue-Status lines of the 80186(80188) to allow synchronization of the queue-status signals to 8087 timing requirements.
QS0O QS1O	3 4	O	**Queue-Status Outputs** QS0O, QS1O are connected to the queue-status pins of the 8087. The signals produced meet 8087 Queue-Status input requirements.

PIN DESCRIPTIONS (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{\text{CSIN}}$	13	I	**Chip-Select Input** $\overline{\text{CSIN}}$ is connected to one of the chip-select lines of the 80186(80188). $\overline{\text{CSIN}}$ informs the 82188 that a bank select is taking place. The 82188 routes this signal to the chip-select output ($\overline{\text{CSOUT}}$). $\overline{\text{CSIN}}$ is active LOW. This line is not used when memory and I/O device addresses are decoded external to the 80186(80188).
$\overline{\text{CSOUT}}$	12	O	**Chip-Select Output** This signal is used as a chip-select line for a bank of memory devices. It is active when $\overline{\text{CSIN}}$ is active or when the 8087 has bus control. $\overline{\text{CSOUT}}$ is active LOW.

FUNCTIONAL DESCRIPTION

BUS CONTROLLER

The 82188 Integrated Bus Controller (IBC) generates system control and command signals. The signals generated are determined by the Status Decoding Logic. The bus controller logic interprets status lines $\overline{\text{S0}}-\overline{\text{S2}}$ to determine what type of bus cycle is taking place. The appropriate signals are then generated by the Command and Control Signal Generators.

The Address Enable ($\overline{\text{AEN}}$) line allows the command and control signals to be disabled. When $\overline{\text{AEN}}$ is inactive (HIGH), the command signals and $\overline{\text{DEN}}$ will be tri-stated, and ALE will be held low (DT/$\overline{\text{R}}$ will be uneffected). $\overline{\text{AEN}}$ inactive will allow other systems to take control of the bus. Control and command signals respond to a change in the $\overline{\text{AEN}}$ signal within 40 ns.

The command signals consist of $\overline{\text{RD}}$ and $\overline{\text{WR}}$. The 82188's $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals are similiar to $\overline{\text{RD}}$ and $\overline{\text{WR}}$ of the 80186(80188) in the non-Queue-Status Mode. These command signals do not differentiate between memory and I/O devices. $\overline{\text{RD}}$ and $\overline{\text{WR}}$ can be conditioned by $\overline{\text{S2}}$ of the 80186(80188) to obtain separate signals for I/O and memory devices.

The control commands consist of Data Enable ($\overline{\text{DEN}}$), Data Transmit/Receive (DT/$\overline{\text{R}}$), and Address Latch Enable (ALE). The control commands are similiar to those generated by the 80186(80188). $\overline{\text{DEN}}$ determines when the external bus should be enabled onto the local bus. DT/$\overline{\text{R}}$ determines the direction of the data transfer, and ALE determines when the address should be strobed into the latches (used for demultiplexing the address bus).

MODE SELECT

The 82188 Integrated Bus Controller (IBC) is configurable. The device has two modes: 80186 Mode and 8086 Mode. Selecting the mode of the device configures the Bus Arbitration Logic (see BUS ARBITRATION section for details). In 80186 Mode, the 82188 IBC may be used as a bus controller/interface device for an 80186(81088), 8087, and optional third processor system. In 8086 Mode, the 82188 IBC may be used as an interface device allowing a maximum mode 8086(8088) to interface with a coprocessor that uses a HOLD-HLDA bus exchange protocol.

The mode of the 82188 is determined during RESET. If the HLDA line is LOW at the falling edge of RESET (as in the case when tied to the HLDA line of the 80186 or 80188), the 82188 will enter into 80186 Mode. If the HLDA line is HIGH at the falling edge of RESET, the 82188 will enter 8086 Mode. In 8086 Mode, only the Bus Arbitration Logic is used. The eight pins used in 8086 Mode are: SYSHOLD, SYSHLDA, HLDA, CLK, RESET, $\overline{\text{RQ}}/\overline{\text{GT}}1$, V_{CC}, and V_{SS}. The other pins may be left unconnected.

BUS ARBITRATION

The Bus Exchange Logic interfaces up to three sets of bus exchange signals:

- HOLD-HLDA
- SYSHOLD-SYSHLDA
- $\overline{\text{RQ}}/\overline{\text{GT}}0$ ($\overline{\text{RQ}}/\overline{\text{GT}}1$)

This logic executes translating, routing, and arbitrating functions. The logic translates HOLD-HLDA signals to $\overline{\text{RQ}}/\overline{\text{GT}}$ signals and $\overline{\text{RQ}}/\overline{\text{GT}}$ signals to HOLD-HLDA signals. The logic also determines which set of bus exchange signals are to be interfaced. The mode of the 82188 and the priority of the devices requesting the bus determine the routing of the bus exchange signals.

80186 MODE

In 80186 Mode, a system may have three potential bus masters: the 80186 or 80188 CPU, the 8087 Numerics Coprocessor, and a third processor (such as the 82586 LAN or 82730 Text Coprocessor). The third processor may have either a HOLD-HLDA or $\overline{RQ}/\overline{GT}$ bus exchange protocol. The possible bus exchange signal connections and paths for 80186 Mode are shown in Figures 3 & 4 and Tables 1 & 2, respectively. If no HOLD-HLDA type third processor is used, SYSHOLD should be tied LOW to prevent an erroneous SYSHOLD signal. In 80186 mode, the bus priorities are:

Highest Priority .... Third Processor
Second Highest Priority 8087
Default Priority 80186

— THREE-PROCESSOR SYSTEM OPERATION (HOLD-HLDA TYPE THIRD PROCESSOR)

In the configuration shown in Figure 3, the third processor requests the bus by sending SYSHOLD HIGH. The 82188 will route (and translate if necessary) the request to the current bus master. This includes routing the request to HOLD if the 80186(80188) is the current bus master or routing and translating the request to $\overline{RQ}/\overline{GT}1$ if the 8087 is in control of the bus. The third processor's request is not passed through the 8087 if the 80186 is the bus master (see Table 1).

The 8087 requests the bus using $\overline{RQ}/\overline{GT}0$. The request pulse from the 8087 will be translated and routed to HOLD if the 80186 is the bus master. If the third processor has control of the bus, the grant pulse to the 8087 will be delayed until the third processor relinquishes the bus (sending SYSHOLD LOW). In this case, HOLD will remain HIGH during the third processor-to-8087 bus control transfer. The 80186 will not be granted the bus until both coprocessors have released it.

Table 1. Bus Exchange Paths (80186 Mode) (HOLD-HLDA Type 3rd Proc)

Requesting Device	Current Bus Master		
	80186	8087	3rd Proc
80186	n/a	n/a	n/a
8087	$\overline{RQ}/\overline{GT}0$ ⟷ HOLD / HLDA	n/a	n/a
3rd Proc	SYSHOLD / SYSHLDA ⟷ HOLD / HLDA	SYSHOLD / SYSHLDA ⟷ $\overline{RQ}/\overline{GT}1$	n/a

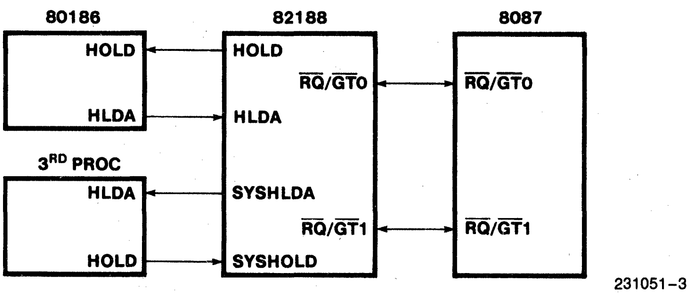


Figure 3.
Bus Exchange Signal Connections (80186 Mode) for a Three Local Processor System (HOLD-HLDA Type 3rd Proc)

Table 2. Bus Exchange Paths (80186 Mode) ($\overline{RQ}/\overline{GT}$ Type 3rd Proc)

Requesting Device	Current Bus Master		
	80186	8087	3rd Proc
80186	n/a	n/a	n/a
8087	$\overline{RQ}/\overline{GT}0 \longleftrightarrow \frac{HOLD}{HLDA}$	n/a	n/a
3rd Proc	$\overline{RQ}/\overline{GT}1 \longleftrightarrow \overline{RQ}/\overline{GT}0 \longleftrightarrow \frac{HOLD}{HLDA}$	$\overline{RQ}/\overline{GT}1$	n/a

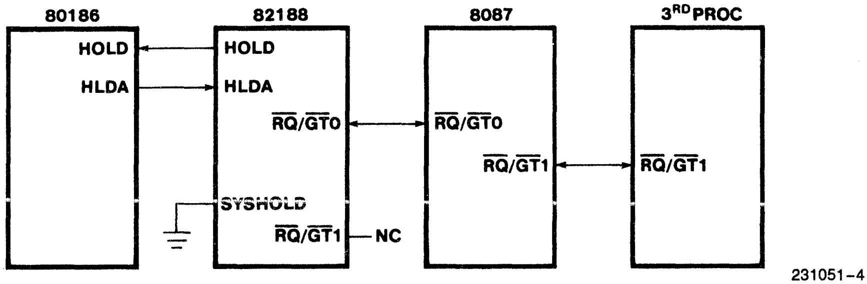


Figure 4.
Bus Exchange Signal Connections (80186 Mode) for a Three Local Processor System ($\overline{RQ}/\overline{GT}$ Type 3rd Proc)

When the bus is requested from the 80186(80188), a bus priority decision is made. This decision is made when the HLDA line goes active. Upon receipt of the HLDA signal, the highest-priority requesting device will be acknowledged the bus. For example, if the 8087 initially requested the bus, the bus will be granted to the third processor if SYSHOLD became active before HLDA was received by the 82188. In this case, the grant pulse to the 8087 will be delayed until the third processor relinquishes the bus.

— THREE-PROCESSOR SYSTEM OPERATION ($\overline{RQ}/\overline{GT}$ TYPE THIRD PROCESSOR)

In the configuration shown in Figure 4, the third processor requests the bus by initiating a request/grant sequence with the 8087's $\overline{RQ}/\overline{GT}1$ line. The 8087 will grant the bus if it is the current bus master or will pass the request on if the 80186 is the current bus master (see Table 2). In this configuration, the 82188's Bus Arbitration Logic translates $\overline{RQ}/\overline{GT}0$ to HOLD-HLDA. The 8087 provides the bus arbitration in this configuration.

8086 MODE

The 8086 Mode allows an 8086, 8088 system to contain both $\overline{RQ}/\overline{GT}$ and HOLD-HLDA type coprocessors simultaneously. In 8086 Mode, two possible bus masters may be interfaced by the 82188; an 8086 or 8088 CPU and a coprocessor which uses a HOLD-HLDA bus exchange protocol (typically an 82586 LAN Coprocessor or an 82730 Text Coprocessor). The bus exchange signal connections for 8086 Mode are shown in Figure 5. Bus arbitration signals used in the 8086 Mode are:

- $\overline{RQ}/\overline{GT}1$
- SYSHOLD
- SYSHLDA

In 8086 Mode, no arbitration is necessary since only two devices are interfaced. The coprocessor has bus priority over the 8086(8088). SYSHOLD-SYSHLDA are routed and translated directly to $\overline{RQ}/\overline{GT}1$. $\overline{RQ}/\overline{GT}1$ of the 82188 may be tied to either $\overline{RQ}/\overline{GT}0$ or $\overline{RQ}/\overline{GT}1$ of the 8086(8088).

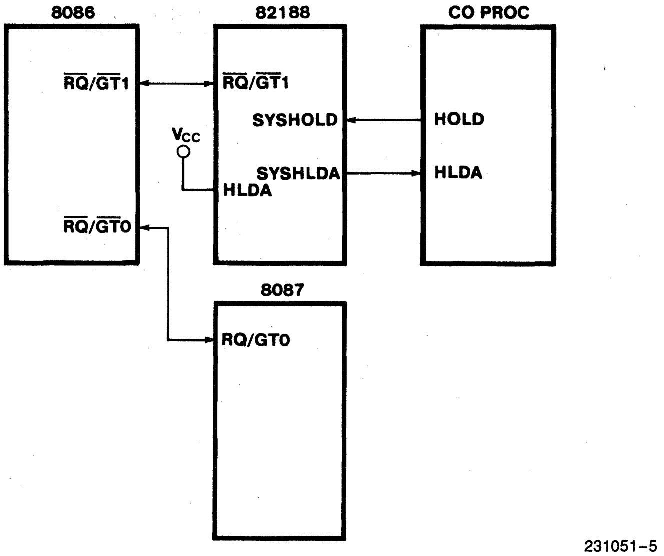


Figure 5. Bus Exchange Signal Connections (8086 Mode)

QUEUE-STATUS DELAY

The Queue-Status Delay logic is used to delay the queue-status signals from the 80186(80188) to meet 8087 queue-status timing requirements. QS0I, QS1I correspond to the queue-status lines of the 80186(80188). The 82188 delays these signals by one clock phase. The delayed signals are interfaced to the 8087 queue-status lines by QS0O, QS1O.

CHIP-SELECT

The Chip-Select Logic allows the utilization of the chip select circuitry of the 80186(80188). Normally, this circuitry could not be used in an 80186(80188)-8087 system since the 8087 contains no chip select circuitry. The Chip-Select Logic contains two external connections: Chip-Select Input ($\overline{CSIN}$) and Chip-Select Output ($\overline{CSOUT}$). $\overline{CSOUT}$ is active when either $\overline{CSIN}$ is active or when the 8087 has control of the bus.

By using $\overline{CSOUT}$ to select memory containing data structures, no external decoding is necessary. The 80186 may gain access to this memory bank through the $\overline{CSIN}$ line while the 8087 will automatically obtain access when it becomes the bus master. Note that this configuration limits the amount of memory accessible by the 8087 to the physical memory bank selected by $\overline{CSOUT}$. Systems where the 8087 must access the full 1 Megabyte address space must use an external decoding scheme.

READY

The Ready logic allows two types of Ready signals: a Synchronous Ready Signal (SRDY) and an Asynchronous Ready Signal (ARDY). These signals are similiar to SRDY and ARDY of the 80186. Wait states will be inserted when both SRDY and ARDY are LOW. Inserting wait states allows slower memory and I/O devices to be interfaced to the 80186(80188)-8087 system.

ARDY's LOW-to-HIGH transition is synchronized to the CPU clock by the 82188. The 82188 samples ARDY at the beginning of T2, T3 and Tw until sampled HIGH. Note that ARDY of the 82188 is sampled one phase earlier than ARDY of the 80186. ARDY's falling edge must be synchronous to the CPU clock. ARDY allows an easy interface with devices that emit an asynchronous ready signal.

The SRDY signal allows direct interface to devices that emit a synchronized ready signal. SRDY must be synchronized to the CPU clock for both of its transitions. SRDY is sampled in the middle of T2, T3 and in the middle of each Tw. An 82188-80186(80188)'s SRDY setup time is 30 ns longer than the 80186(80188)'s SRDY setup time. SRDY eliminates the half-clock cycle penalty necessary for ARDY to be internally sychronized.

The sychronized ready output (SRO) is the accumulation of SRDY, ARDY, and the internal wait-state generator. SRO should be connected to SRDY of the 80186(80188) (with 80186(80188)'s ARDY tied LOW), and READY of the 8087.

SRDY	ARDY	SRO
0	0	0
1	X	1
X	1	1

The internal wait state generator allows for synchronization between the 80186(80188) and 8087 in 80186 mode. Upon RESET, the 82188 automatically inserts 3 wait-states per 80186(80188) bus cycle, overlapped with any externally produced wait-states created by ARDY and SRDY.

Since the 8087 has no provision for internal wait-state generation, only externally created wait states will be effective. The 82188, upon RESET, will inject 3 wait states for each of the first 256 80186(80188) bus cycles onto the SRO line. This will allow the 8087 to match the 80186(80188)'s timing.

The internally-generated wait states are overlapped with those produced by the SRDY and ARDY lines. Overlapping the injected wait states insures a minimum of three wait states for the first 256 80186(80188) bus cycles after RESET. Systems with a greater number of wait states will not be effected. Internal wait state generation by the 82188 will stop on the 256th 80186(80188) bus cycle after RESET. To maintain sychronization between the 80186(80188) and 8087, the following conditions are necessary:

- The 80186(80188)'s control block must be mapped in I/O space before it is written to or read from.
- All memory chip-select lines must be set to 0 WAIT STATES, EXTERNAL READY ALSO USED within the first 256 80186(80188) bus cycles after RESET.

An equivalent READY logic diagram is shown in Figure 6.

SYSTEM CONSIDERATIONS

In any 82188 configuration, clock compatibility must be considered. Depending on the device, a 50% or a 33% duty-cycle clock is needed. For example, the 80186 and 80188 (as well as the 82188, 82586, and 82730) requires a 50% duty-cycle clock. The 8086, 8088 and their 'kit' devices' (8087, 8089, 8288, and 8289) clock requirements, on the other hand, require a 33% duty-cycle clock signal. The system designer must make sure clock requirements of all the devices in the system are met.

Figure 7 demonstrates the usage of the 82188 in 80186 Mode where it is used to interface an 8087 into an 80186 system.

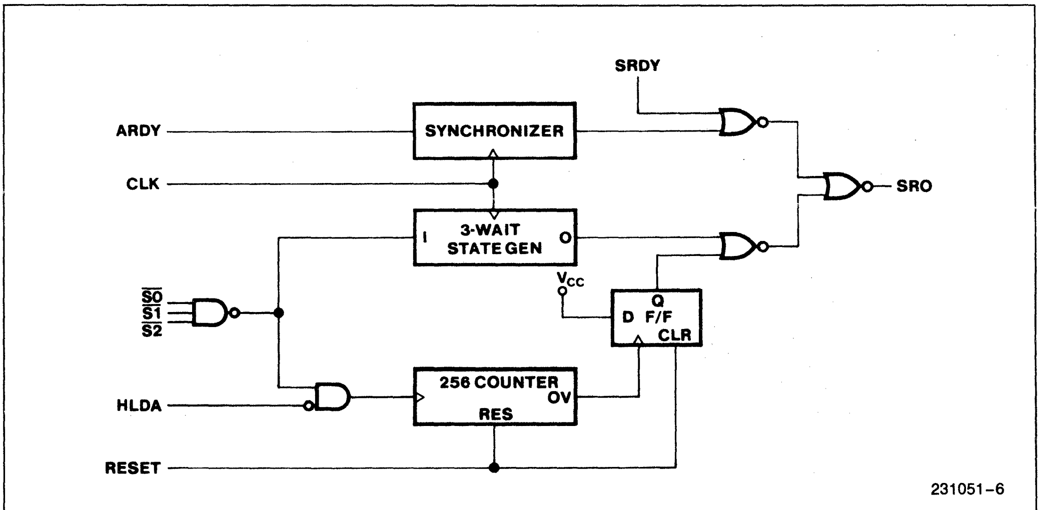


Figure 6.
Equivalent 82188 READY circuit

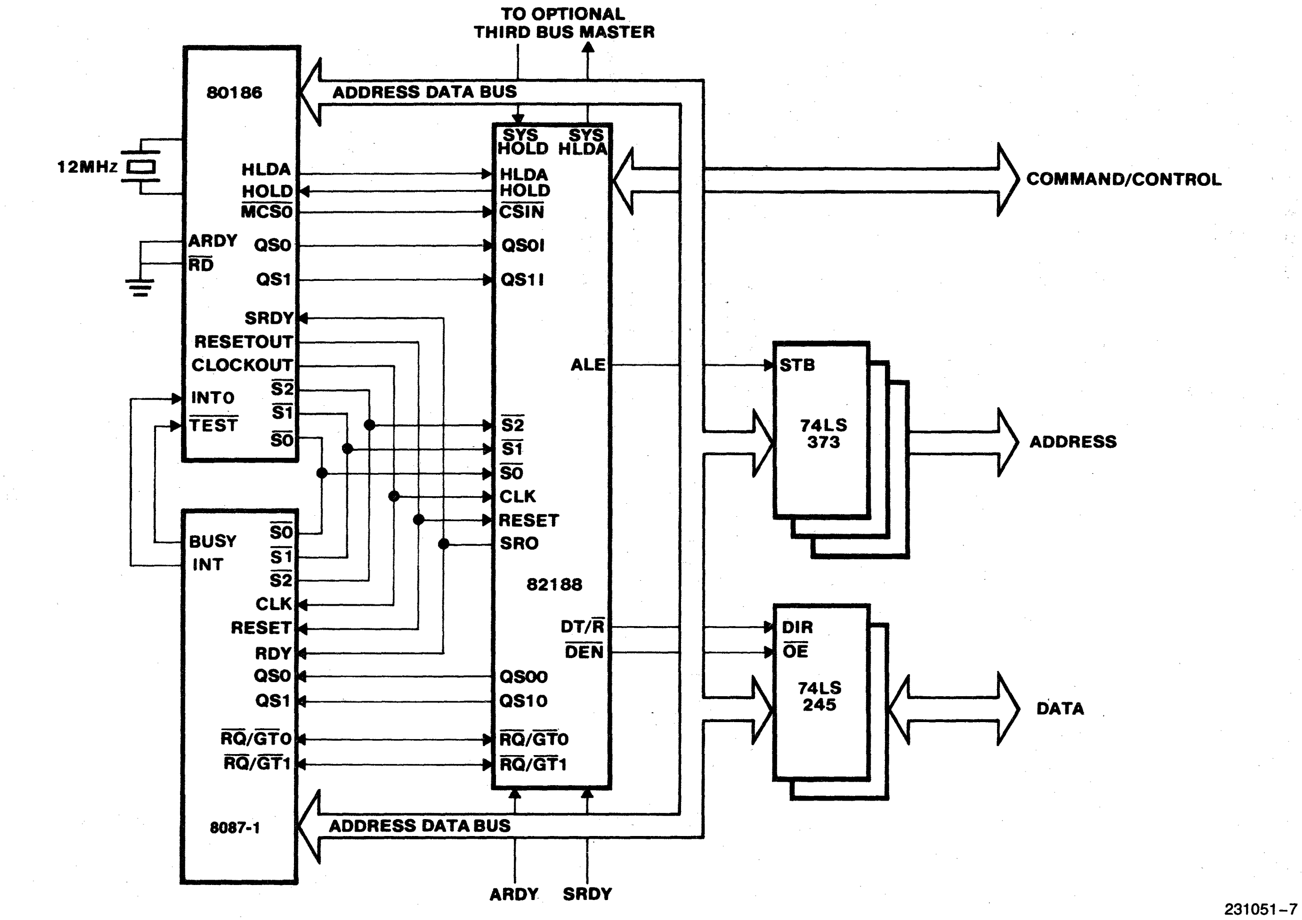


Figure 7.
80186-6/8087-2 System Using the 82188 in 80186 Mode

ABSOLUTE MAXIMUM RATINGS *

Temperature Under Bias .... 0°C to 70°C
Storage Temperature............. −65°C to 150°C
Case Temperature .... 0°C to +85°C
Voltage on any Pin with
Respect to GND .... −1.0V to 7.0V
Power Dissipation .... 0.7 Watts

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

DC CHARACTERISTICS

(V_{CC} = 5V ± 10%, T_A = 0°C to 70°C, T_{CASE} = 0°C to +85°C)

Symbol	Parameter	Min	Max	Units	Test Cond.
V_{IL}	Input Low Voltage	−0.5	+0.8	volts	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	volts	
V_{OL}	Output Low Voltage		0.45	volts	I_{OL} = 2 mA
V_{OH}	Output High Voltage	2.4		volts	I_{OH} = −400 μA
I_{CC}	Power Supply Current		100	mA	T_A = 25°C
I_{LI}	Input Leakage Current		±10	μA	$0V < V_{IN} < V_{CC}$
I_{LO}	Output Leakage Current		±10	μA	$0.45 < V_{OUT} < V_{CC}$
V_{CLI}	CLK Input Low Voltage	−0.5	+0.6	volts	
V_{CHI}	CLK Input High Voltage	3.9	V_{CC} + 1.0	volts	
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		20	pF	

AC CHARACTERISTICS

(V_{CC} = 5V ± 10%, T_A = 0°C to 70°C, T_{CASE} = 0°C to +85°C)

TIMING REQUIREMENTS

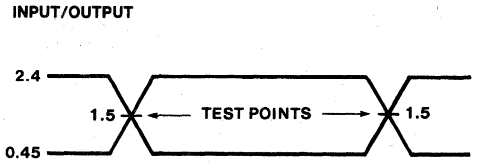
Symbol	Parameter	Min	Max	Units	Notes
TCLCL	Clock Period	125	500	ns	
TCLCH	Clock LOW Time	½TCLCL-7.5		ns	
TCHCL	Clock HIGH Time	½TCLCL-7.5		ns	
TARYHCL	ARDY Active Setup Time	20		ns	
TCHARYL	ARDY Hold Time	15		ns	8
TARYLCH	ARDY Inactive Setup Time	35		ns	
TSRYHCL	SRDY Input Setup Time	65,50		ns	1
TSVCH	STATUS Active Setup Time	55		ns	
TSXCL	STATUS Inactive Setup Time	50		ns	
TQIVCL	QS0I, QS1I Setup Time	15		ns	
THAVGV	HLDA Setup Time	50		ns	
TSHVCL	SYSHOLD Asynchronous Setup Time	25		ns	
TGVCH	$\overline{RQ}/\overline{GT}$ Input Setup Time	0		ns	6

TIMING RESPONSES

Symbol	Parameter	Min	Max	Units	Notes
TSVLH	STATUS Valid to ALE Delay		30	ns	4
TCHLL	ALE Inactive Delay		30	ns	
TCLML	$\overline{RD}$, $\overline{WR}$ Active Delay	10	70	ns	
TCLMH	$\overline{RD}$, $\overline{WR}$ Inactive Delay	10	55	ns	
TSVDTV	STATUS to DT/$\overline{R}$ Delay		30	ns	3
TCLDTV	DT/$\overline{R}$ Active Delay		55	ns	3
TCHDNV	$\overline{DEN}$ Active Delay	10	55	ns	
TCHDNX	$\overline{DEN}$ Inactive Delay	10	55	ns	
TCLQOV	QS0O, QS1O Delay	5	50	ns	
TCHHV	HOLD Delay		50	ns	2,6
TCLSAV	SYSHLDA Delay		50	ns	6
TCLGV	$\overline{RQ}$/$\overline{GT}$ Output Delay		40	ns	6
TGVHV	$\overline{RQ}$/$\overline{GT0}$ To HOLD Delay		50	ns	2,6
TCLLH	ALE Active Delay		30	ns	4
TAELCV	Command Enable Delay		40	ns	
TAEHCX	Command Disable Delay		40	ns	
TCHRO	SRO Output Delay	5	30	ns	5,6
TSRYHRO	SRDY To SRO Delay		30	ns	5
TCSICSO	$\overline{CSIN}$ To $\overline{CSOUT}$ Delay		30	ns	
TCLCSOV	CLK Low to $\overline{CSOUT}$ Delay	10		ns	
TCLCSOH	CLK Low to $\overline{CSOUT}$ Inactive Delay	10		ns	

NOTES (applicable to both spec listing and timing diagrams):
1. TSRYHOL = (80186's) TSRYCL + 30 ns = 65 ns for 6 MHz operation and 50 ns for 8 MHz operation.
2. Timing not tested.
3. DT/$\overline{R}$ will be asserted to the latest of TSVDTV & TCLDTV.
4. ALE will be asserted to the latest of TSVLH & TCLLH.
5. SRO will be asserted to the latest of TCHRO & TSRYHRO.
6. CL = 20–100 pF
7. Address/Data bus shown for reference only.
8. The falling edge of ARDY must be synchronized to CLK.

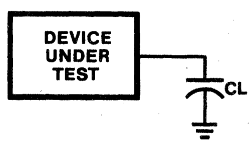
A.C. TESTING INPUT, OUTPUT WAVEFORM



231051-9

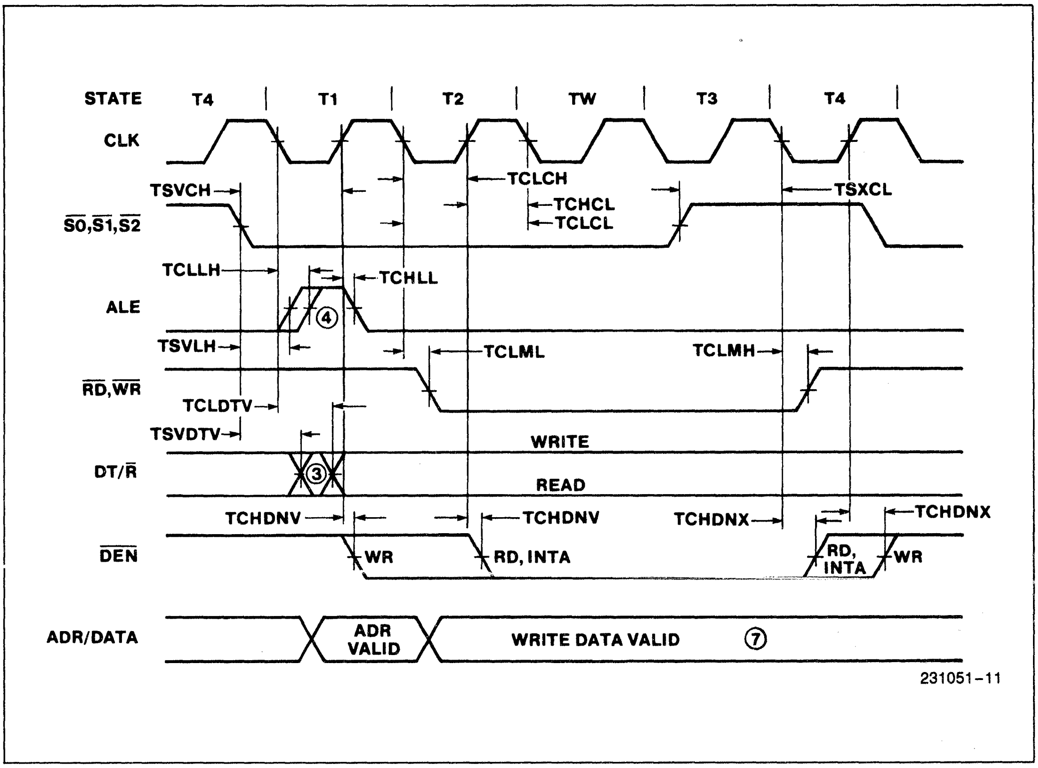
A.C. Testing: Inputs are driven at 2.4V for a Logic '1' and 0.45V for a Logic '0'.

A.C. TESTING LOAD CIRCUIT

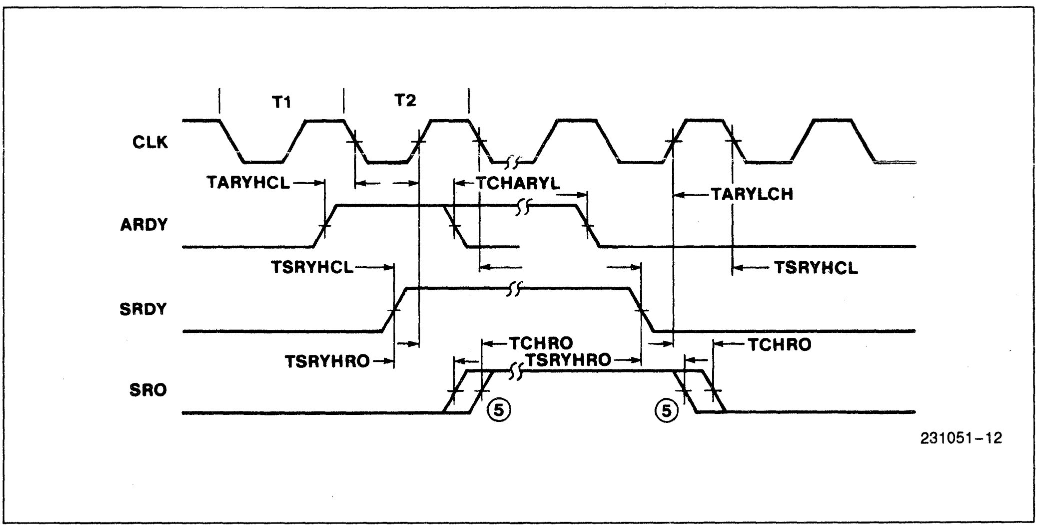


231051-10

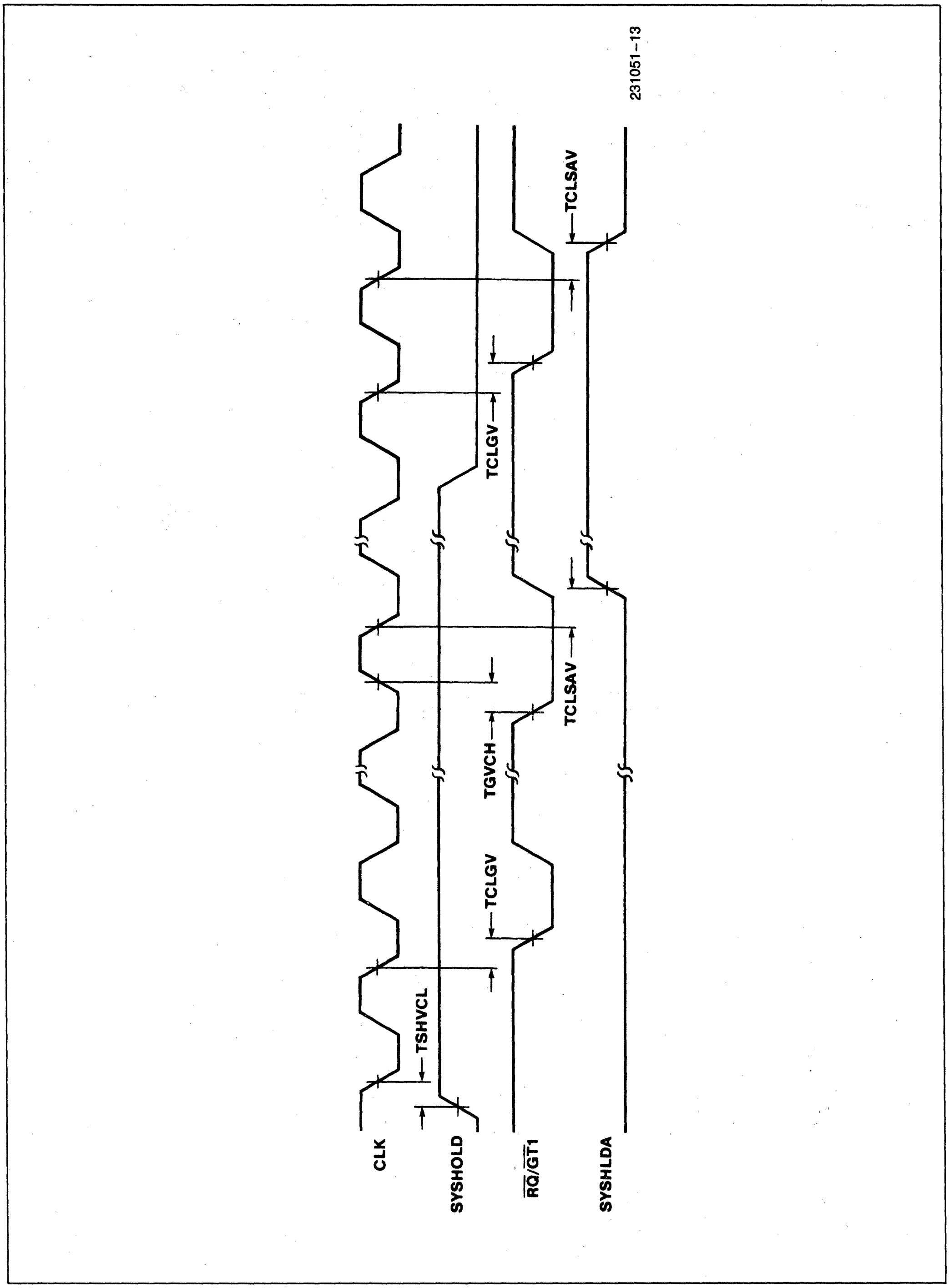
C_L includes Jig Capacitance
C_L = 20–200 pF unless otherwise noted



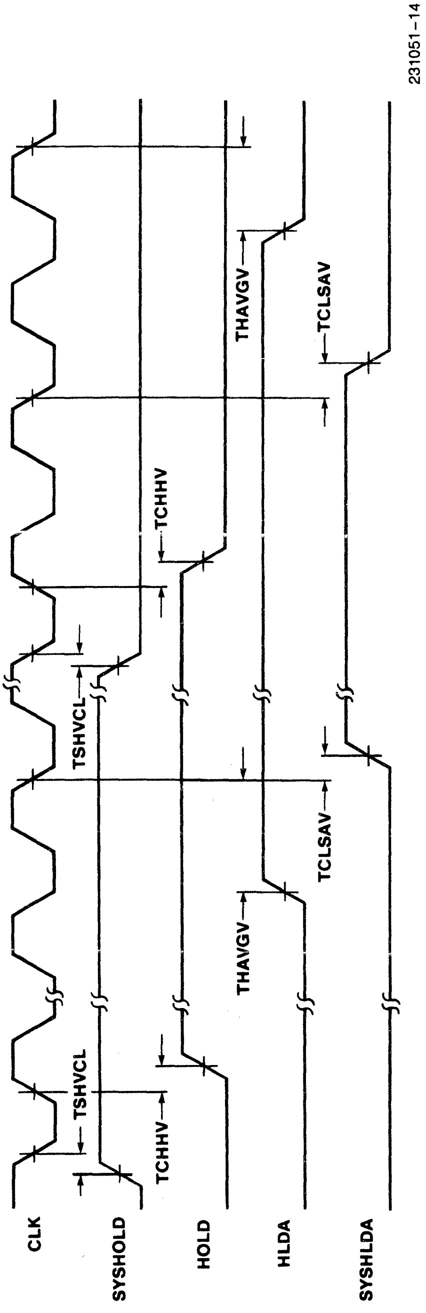
Command and Control Waveforms-80186 Mode



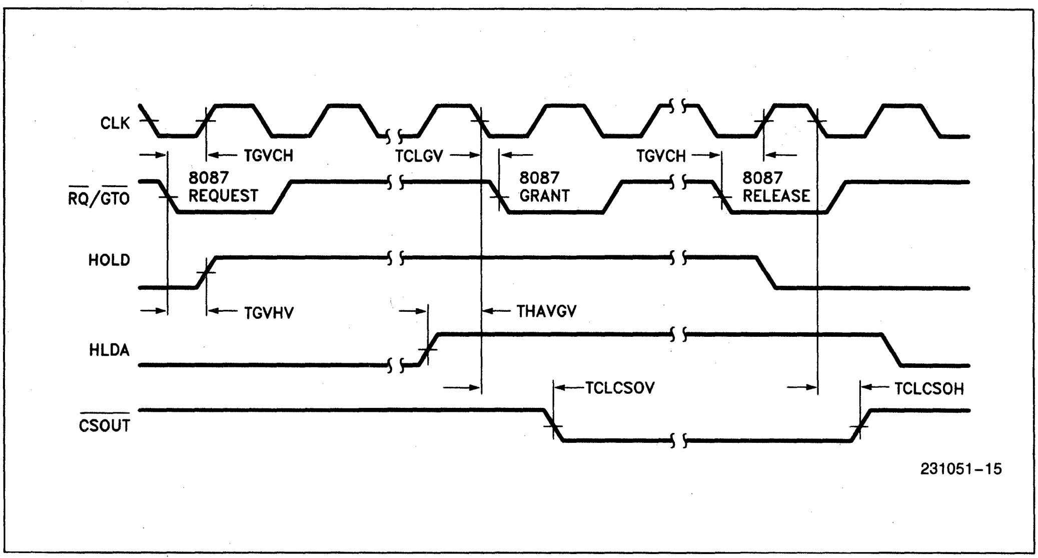
READY Timing-80186 Mode



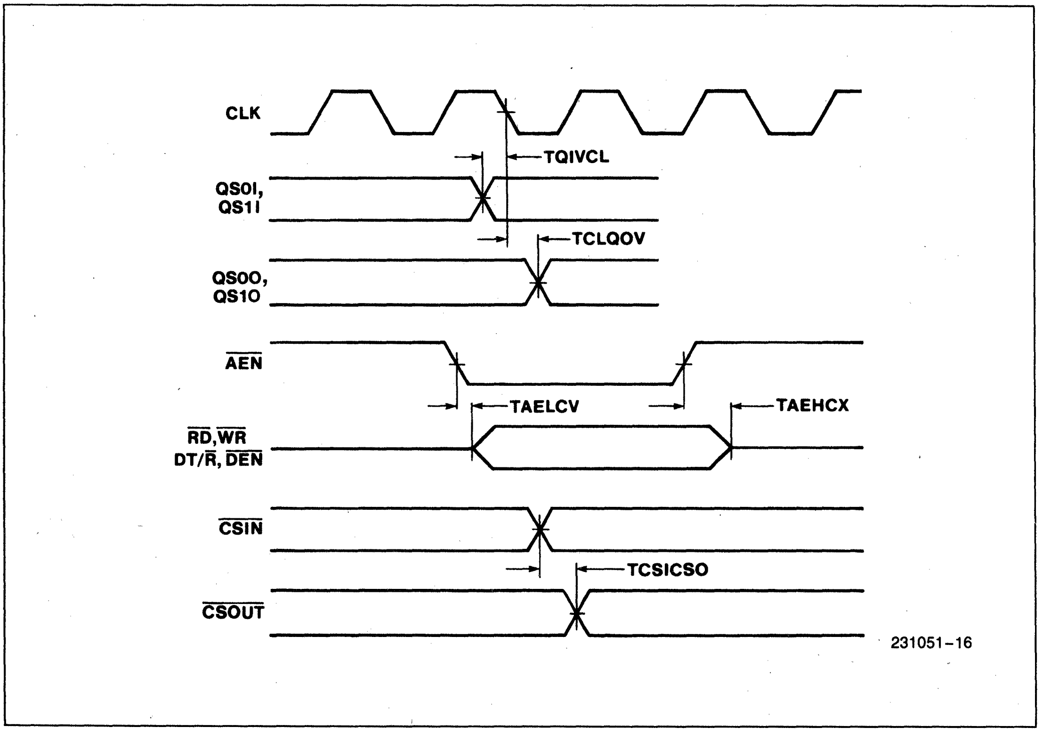
SYSHOLD-SYSHLDA to $\overline{RQ}/\overline{GT}1$ Timing—80186 Mode and 8086 Mode



SYSHOLD-SYSHLDA To HOLD-HLDA Timing—80186 Mode



RQ/GT0 to HOLD-HLDA Timing-80186 Mode



Queue Status, ALE, Chip Select Delay Timing-80186 Mode

intel®

8289/8289-1
BUS ARBITER

- Provides Multi-Master System Bus Protocol
- Synchronizes iAPX 86, 88 Processors with Multi-Master Bus
- 10MHz Version, 8289-1, Fully Compatible with 10MHz iAPX 86 or 8MHz iAPX 186 Based Systems
- Provides Simple Interface with 8288 Bus Controller
- Four Operating Modes for Flexible System Configuration
- Compatible with Intel Bus Standard MULTIBUS™
- Provides System Bus Arbitration for 8089 IOP in Remote Mode
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range

The Intel 8289 Bus Arbiter is a 20-pin, 5-volt-only bipolar component for use with medium to large iAPX 86, 88 multi-master/multiprocessing systems. The 8289 provides system bus arbitration for systems with multiple bus masters, such as an 8086 CPU with 8089 IOP in its REMOTE mode, while providing bipolar buffering and drive capability.

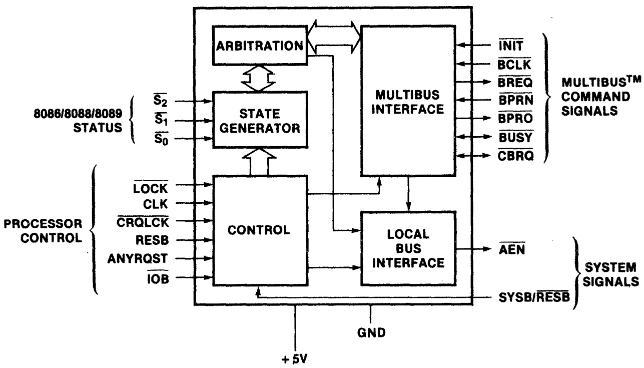


Figure 1. Block Diagram

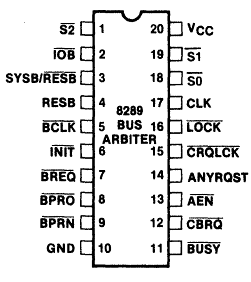


Figure 2. Pin Diagram

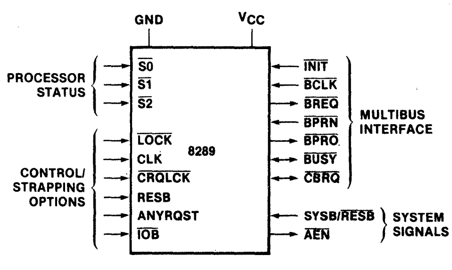


Figure 3. Functional Pinout

Table 1. Pin Description

Symbol	Type	Name and Function
V_{CC}		**Power:** +5V supply ±10%.
GND		**Ground.**
$\overline{S0}$,$\overline{S1}$,$\overline{S2}$	I	**Status Input Pins:** The status input pins from an 8086, 8088 or 8089 processor. The 8289 decodes these pins to initiate bus request and surrender actions. (See Table 2.)
CLK	I	**Clock:** From the 8284 clock chip and serves to establish when bus arbiter actions are initiated.
$\overline{LOCK}$	I	**Lock:** A processor generated signal which when activated (low) prevents the arbiter from surrendering the multi-master system bus to any other bus artiter, regardless of its priority.
$\overline{CRQLCK}$	I	**Common Request Lock:** An active low signal which prevents the arbiter from surrendering the multi-master system bus to any other bus arbiter requesting the bus through the $\overline{CBRQ}$ input pin.
RESB	I	**Resident Bus:** A strapping option to configure the arbiter to operate in systems having both a multi-master system bus and a Resident Bus. Strapped high, the multi-master system bus is requested or surrendered as a function of the SYSB/$\overline{RESB}$ input pin. Strapped low, the SYSB/$\overline{RESB}$ input is ignored.
ANYRQST	I	**Any Request:** A strapping option which permits the multi-master system bus to be surrendered to a lower priority arbiter as if it were an arbiter of higher priority (i.e., when a lower priority arbiter requests the use of the multi-master system bus, the bus is surrendered as soon as it is possible). When ANYRQST is strapped low, the bus is surrendered according to Table 2. If ANYRQST is strapped high and $\overline{CBRQ}$ is activated, the bus is surrendered at the end of the present bus cycle. Strapping $\overline{CBRQ}$ low and ANYRQST high forces the 8289 arbiter to surrender the multi-master system bus after each transfer cycle. Note that when surrender occurs BREQ is driven false (high).
$\overline{IOB}$	I	**IO Bus:** A strapping option which configures the 8289 Arbiter to operate in systems having both an IO Bus (Peripheral Bus) and a multi-master system bus. The arbiter requests and surrenders the use of the multi-master system bus as a function of the status line, $\overline{S2}$. The multi-master system bus is permitted to be surrendered while the processor is performing IO commands and is requested whenever the processor performs a memory command. Interrupt cycles are assumed as coming from the peripheral bus and are treated as an IO command.
$\overline{AEN}$	O	**Address Enable:** The output of the 8289 Arbiter to the processor's address latches, to the 8288 Bus Controller and 8284A Clock Generator. $\overline{AEN}$ serves to instruct the Bus Controller and address latches when to tri-state their output drivers.
SYSB/$\overline{RESB}$	I	**System Bus/Resident Bus:** An input signal when the arbiter is configured in the S.R. Mode (RESB is strapped high) which determines when the multi-master system bus is requested and multi-master system bus surrendering is permitted. The signal is intended to originate from a form of address-mapping circuitry, as a decoder or PROM attached to the resident address bus. Signal transitions and glitches are permitted on this pin from ϕ1 of T4 to ϕ 1 of T2 of the processor cycle. During the period from ϕ1 of T2 to ϕ 1 of T4, only clean transitions are permitted on this pin (no glitches). If a glitch occurs, the arbiter may capture or miss it, and the multi-master system bus may be requested or surrendered, depending upon the state of the glitch. The arbiter requests the multi-master system bus in the S.R. Mode when the state of the SYSB/$\overline{RESB}$ pin is high and permits the bus to be surrendered when this pin is low.
$\overline{CBRQ}$	I/O	**Common Bus Request:** An input signal which instructs the arbiter if there are any other arbiters of lower priority requesting the use of the multi-master system bus. The $\overline{CBRQ}$ pins (open-collector output) of all the 8289 Bus Arbiters which surrender to the multi-master system bus upon request are connected together. The Bus Arbiter running the current transfer cycle will not itself pull the $\overline{CBRQ}$ line low. Any other arbiter connected to the $\overline{CBRQ}$ line can request the multi-master system bus. The arbiter presently running the current transfer cycle drops its $\overline{BREQ}$ signal and surrenders the bus whenever the proper surrender conditions exist. Strapping $\overline{CBRQ}$ low and ANYRQST high allows the multi-master system bus to be surrendered after each transfer cycle. See the pin definition of ANYRQST.
$\overline{INIT}$	I	**Initialize:** An active low multi-master system bus input signal used to reset all the bus arbiters on the multi-master system bus. After initialization, no arbiters have the use of the multi-master system bus.

Table 1. Pin Descriptions (Continued)

Symbol	Type	Name and Function
$\overline{BCLK}$	I	**Bus Clock:** The multi-master system bus clock to which all multi-master system bus interface signals are synchronized.
$\overline{BREQ}$	O	**Bus Request:** An active low output signal in the parallel Priority Resolving Scheme which the arbiter activates to request the use of the multi-master system bus.
$\overline{BPRN}$	I	**Bus Priority In:** The active low signal returned to the arbiter to instruct it that it may acquire the multi-master system bus on the next falling edge of BCLK. $\overline{BPRN}$ indicates to the arbiter that it is the highest priority requesting arbiter presently on the bus. The loss of $\overline{BPRN}$ instructs the arbiter that it has lost priority to a higher priority arbiter.

Symbol	Type	Name and Function
$\overline{BPRO}$	O	**Bus Priority Out:** An active low output signal used in the serial priority resolving scheme where $\overline{BPRO}$ is daisy-chained to $\overline{BPRN}$ of the next lower priority arbiter.
$\overline{BUSY}$	I/O	**Busy:** An active low open collector multi-master system bus interface signal used to instruct all the arbiters on the bus when the multi-master system bus is available. When the multi-master system bus is available the highest requesting arbiter (determined by $\overline{BPRN}$) seizes the bus and pulls $\overline{BUSY}$ low to keep other arbiters off of the bus. When the arbiter is done with the bus, it releases the $\overline{BUSY}$ signal, permitting it to go high and thereby allowing another arbiter to acquire the multi-master system bus.

FUNCTIONAL DESCRIPTION

The 8289 Bus Arbiter operates in conjunction with the 8288 Bus Controller to interface iAPX 86, 88 processors to a multi-master system bus (both the iAPX 86 and iAPX 88 are configured in their max mode). The processor is unaware of the arbiter's existence and issues commands as though it has exclusive use of the system bus. If the processor does not have the use of the multi-master system bus, the arbiter prevents the Bus Controller (8288), the data transceivers and the address latches from accessing the system bus (e.g. all bus driver outputs are forced into the high impedance state). Since the command sequence was not issued by the 8288, the system bus will appear as "Not Ready" and the processor will enter wait states. The processor will remain in Wait until the Bus Arbiter acquires the use of the multi-master system bus whereupon the arbiter will allow the bus controller, the data transceivers, and the address latches to access the system. Typically, once the command has been issued and a data transfer has taken place, a transfer acknowledge (XACK) is returned to the processor to indicate "READY" from the accessed slave device. The processor then completes its transfer cycle. Thus the arbiter serves to multiplex a processor (or bus master) onto a multi-master system bus and avoid contention problems between bus masters.

Arbitration Between Bus Masters

In general, higher priority masters obtain the bus when a lower priority master completes its present transfer cycle. Lower priority bus masters obtain the bus when a higher priority master is not accessing the system bus. A strapping option (ANYRQST) is provided to allow the arbiter to surrender the bus to a lower priority master as though it were a master of higher priority. If there are no other bus masters requesting the bus, the arbiter maintains the bus so long as its processor has not entered the HALT State. The arbiter will not voluntarily surrender the system bus and has to be forced off by another master's bus request, the HALT State being the only exception. Additional strapping options permit other modes of operation wherein the multi-master system bus is surrendered or requested under different sets of conditions.

Priority Resolving Techniques

Since there can be many bus masters on a multi-master system bus, some means of resolving priority between bus masters simultaneously requesting the bus must be provided. The 8289 Bus Arbiter provides several resolving techniques. All the techniques are based on a priority concept that at a given time one bus master will have priority above all the rest. There are provisions for using parallel priority resolving techniques, serial priority resolving techniques, and rotating priority techniques.

PARALLEL PRIORITY RESOLVING

The parallel priority resolving technique uses a separate bus request line ($\overline{BREQ}$) for each arbiter on the multi-master system bus, see Figure 4. Each $\overline{BREQ}$ line enters into a priority encoder which generates the binary address of the highest priority $\overline{BREQ}$ line which is active. The binary address is decoded by a decoder to select the corresponding $\overline{BPRN}$ (Bus Priority In) line to be returned to the highest priority requesting arbiter. The arbiter receiving priority ($\overline{BPRN}$ true) then allows its associated bus master onto the multi-master system bus as soon as it becomes available (i.e., the bus is no longer busy). When one bus arbiter gains priority over another arbiter it cannot immediately seize the bus, it must wait until the present bus transaction is complete.

Upon completing its transaction the present bus occupant recognizes that it no longer has priority and surrenders the bus by releasing BUSY. BUSY is an active low "OR" tied signal line which goes to every bus arbiter on the system bus. When BUSY goes inactive (high), the arbiter which presently has bus priority (BPRN true) then seizes the bus and pulls BUSY low to keep other arbiters off of the bus. See waveform timing diagram, Figure 5. Note that all multi-master system bus transactions are synchronized to the bus clock (BCLK). This allows the parallel priority resolving circuitry or any other priority resolving scheme employed to settle.

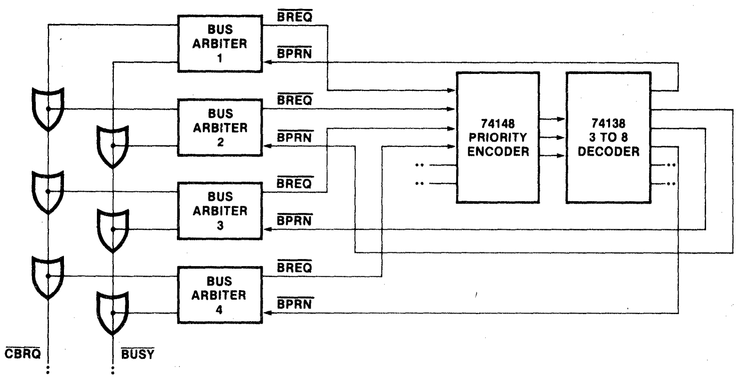


Figure 4. Parallel Priority Resolving Technique

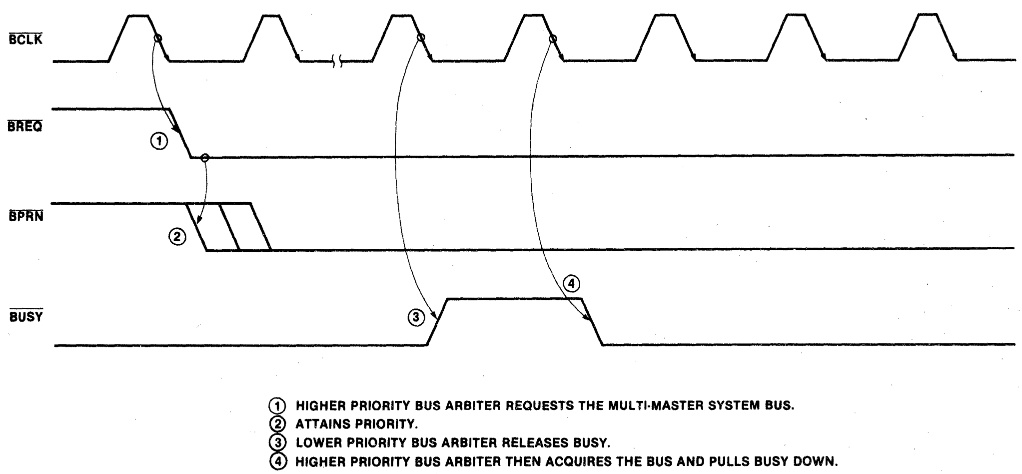
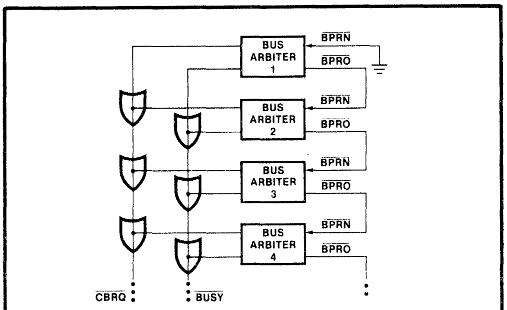


Figure 5. Higher Priority Arbiter obtaining the Bus from a Lower Priority Arbiter

SERIAL PRIORITY RESOLVING

The serial priority resolving technique eliminates the need for the priority encoder-decoder arrangement by daisy-chaining the bus arbiters together, connecting the higher priority bus arbiter's $\overline{BPRO}$ (Bus Priority Out) output to the $\overline{BPRN}$ of the next lower priority. See Figure 6.



THE NUMBER OF ARBITERS THAT MAY BE DAISY-CHAINED TOGETHER IN THE SERIAL PRIORITY RESOLVING SCHEME IS A FUNCTION OF BCLK AND THE PROPAGATION DELAY FROM ARBITER TO ARBITER. NORMALLY, AT 10 MHz ONLY 3 ARBITER MAY BE DAISY-CHAINED.

Figure 6. Serial Priority Resolving

ROTATING PRIORITY RESOLVING

The rotating priority resolving technique is similar to that of the parallel priority resolving technique except that priority is dynamically re-assigned. The priority encoder is replaced by a more complex circuit which rotates priority between requesting arbiters thus allowing each arbiter an equal chance to use the multi-master system bus, over time.

Which Priority Resolving Technique To Use

There are advantages and disadvantages for each of the techniques described above. The rotating priority resolving technique requires substantial external logic to implement while the serial technique uses no external logic but can accommodate only a limited number of bus arbiters before the daisy-chain propagation delay exceeds the multi-master's system bus clock ($\overline{BCLK}$). The parallel priority resolving technique is in general a good compromise between the other two techniques. It allows for many arbiters to be present on the bus while not requiring too much logic to implement.

8289 MODES OF OPERATION

There are two types of processors in the iAPX 86 family. An Input/Output processor (the 8089 IOP) and the iAPX 86/10, 88/10 CPUs. Consequently, there are two basic operating modes in the 8289 bus arbiter. One, the IOB (I/O Peripheral Bus) mode, permits the processor access to both an I/O Peripheral Bus and a multi-master system bus. The second, the RESB (Resident Bus mode), permits the processor to communicate over both a Resident Bus and a multi-master system bus. An I/O Peripheral Bus is a bus where all devices on that bus, including memory, are treated as I/O devices and are addressed by I/O commands. All memory commands are directed to another bus, the multi-master system bus. A Resident Bus can issue both memory and I/O commands, but it is a distinct and separate bus from the multi-master system bus. The distinction is that the Resident Bus has only one master, providing full availability and being dedicated to that one master.

The $\overline{IOB}$ strapping option configures the 8289 Bus Arbiter into the $\overline{IOB}$ mode and the strapping option RESB configures it into the RESB mode. It might be noted at this point that if both strapping options are strapped false, the arbiter interfaces the processor to a multi-master system bus only (see Figure 7). With both options strapped true, the arbiter interfaces the processor to a multi-master system bus, a Resident Bus, and an I/O Bus.

In the $\overline{IOB}$ mode, the processor communicates and controls a host of peripherals over the Peripheral Bus. When the I/O Processor needs to communicate with system memory, it does so over the system memory bus. Figure 8 shows a possible I/O Processor system configuration.

The iAPX 86 and iAPX 88 processors can communicate with a Resident Bus and a multi-master system bus. Two bus controllers and only one Bus Arbiter would be needed in such a configuration as shown in Figure 9. In such a system configuration the processor would have access to memory and peripherals of both busses. Memory mapping techniques are applied to select which bus is to be accessed. The SYSB/RESB input on the arbiter serves to instruct the arbiter as to whether or not the system bus is to be accessed. The signal connected to SYSB/RESB also enables or disables commands from one of the bus controllers.

A summary of the modes that the 8289 has, along with its response to its status lines inputs, is summarized in Table 2.

*In some system configurations it is possible for a non-I/O Processor to have access to more than one Multi-Master System Bus, see 8289 Application Note.

Table 2. Summary of 8289 Modes, Requesting and Relinquishing the Multi-Master System Bus

	Status Lines From 8086 or 8088 or 8089			IOB Mode Only	RESB (Mode) Only IOB = High RESB = High		IOB Mode RESB Mode IOB = Low RESB = High		Single Bus Mode IOB = High RESB = Low
	$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	$\overline{IOB}$ = Low	SYSB/RESB = High	SYSB/$\overline{RESB}$ = Low	SYSB/$\overline{RESB}$ = High	SYSB/$\overline{RESB}$ = Low	
I/O COMMANDS	0	0	0	x		x	x	x	
	0	0	1	x		x	x	x	
	0	1	0	x		x	x	x	
HALT	0	1	1	x	x	x	x	x	x
MEM COMMANDS	1	0	0			x		x	
	1	0	1			x		x	
	1	1	0			x		x	
IDLE	1	1	1	x	x	x	x	x	x

NOTES:

1. X = Multi-Master System Bus is allowed to be Surrendered.
2. ✓ = Multi-Master System Bus is Requested.

Mode	Pin Strapping	Multi-Master System Bus Requested**	Multi-Master System Bus Surrendered*
Single Bus Multi-Master Mode	$\overline{IOB}$ = High RESB = Low	Whenever the processor's status lines go active	HLT + TI • CBRQ + HPBRQ†
RESB Mode Only	$\overline{IOB}$ = High RESB = High	SYSB/$\overline{RESB}$ = High • ACTIVE STATUS	(SYSB/$\overline{RESB}$ = Low + TI) • CBRQ + HLT + HPBRQ
IOB Mode Only	$\overline{IOB}$ = Low RESB = Low	Memory Commands	(I/O Status + TI) • CBRQ + HLT + HPBRQ
IOB Mode·RESB Mode	$\overline{IOB}$ = Low RESB = High	(Memory Command) • (SYSB/$\overline{RESB}$ = High)	((I/O Status Commands) + SYSB/$\overline{RESB}$ = LOW)) • CBRQ + HPBRQ† + HLT

NOTES:

*$\overline{LOCK}$ prevents surrender of Bus to any other arbiter, $\overline{CRQLCK}$ prevents surrender of Bus to any lower priority arbiter.

**Except for HALT and Passive or IDLE Status.

†HPBRQ, Higher priority Bus request or $\overline{BPRN}$ = 1.

1. $\overline{IOB}$ Active Low.
2. RESB Active High.
3. + is read as "OR" and • as "AND."
4. TI = Processor Idle Status $\overline{S2}$, $\overline{S1}$, $\overline{S0}$ = 111
5. HLT = Processor Halt Status $\overline{S2}$, $\overline{S1}$, $\overline{S0}$ = 011

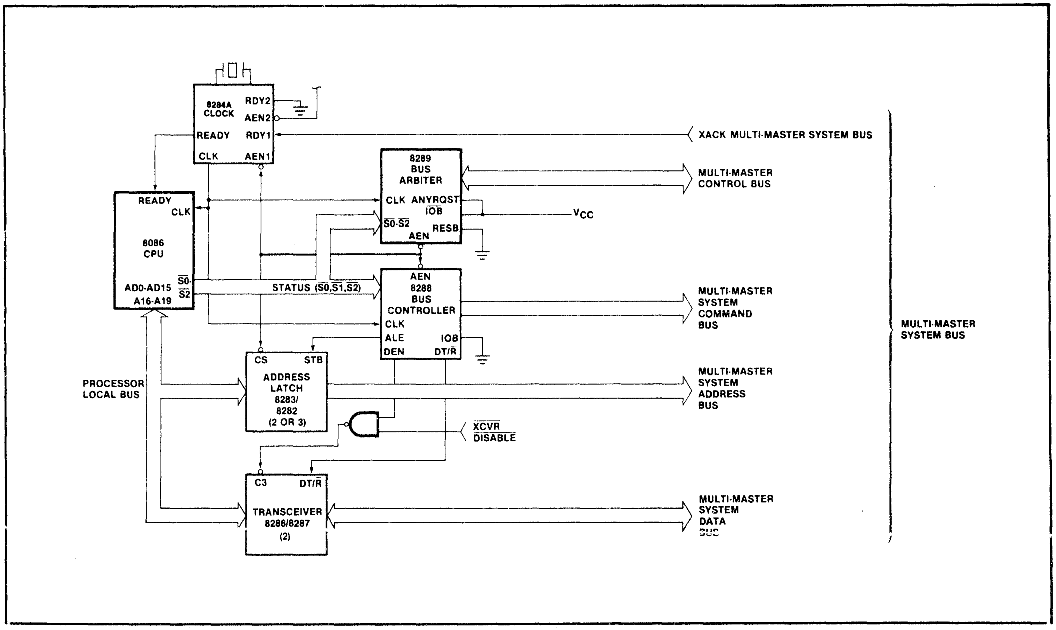


Figure 7. Typical Medium Complexity CPU System

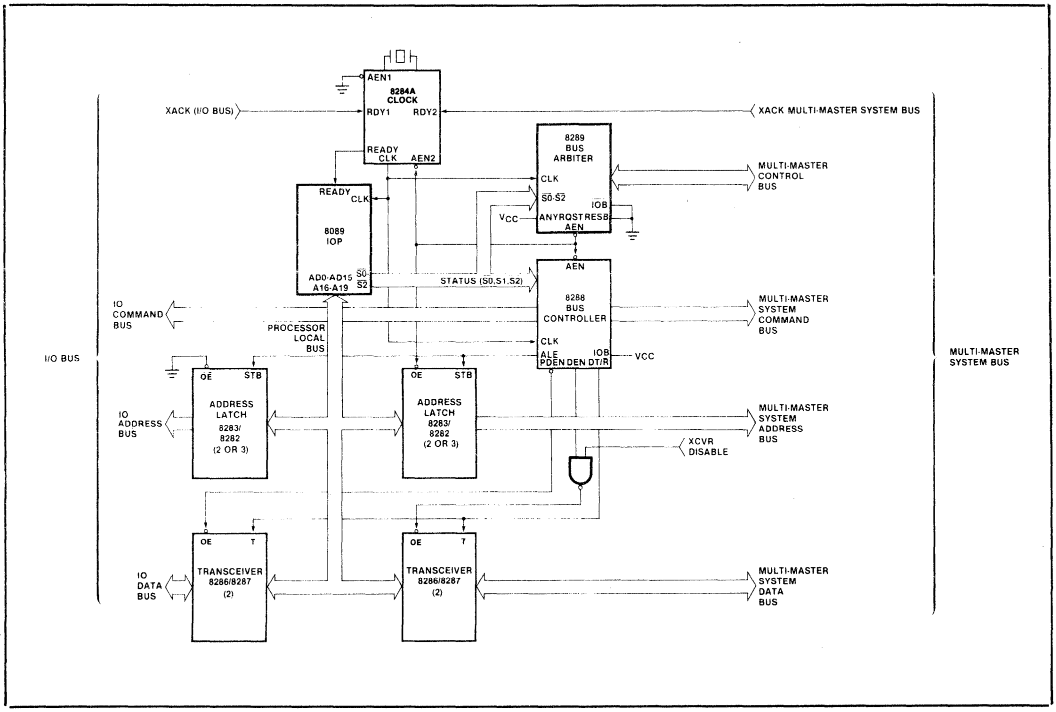


Figure 8. Typical Medium Complexity IOB System

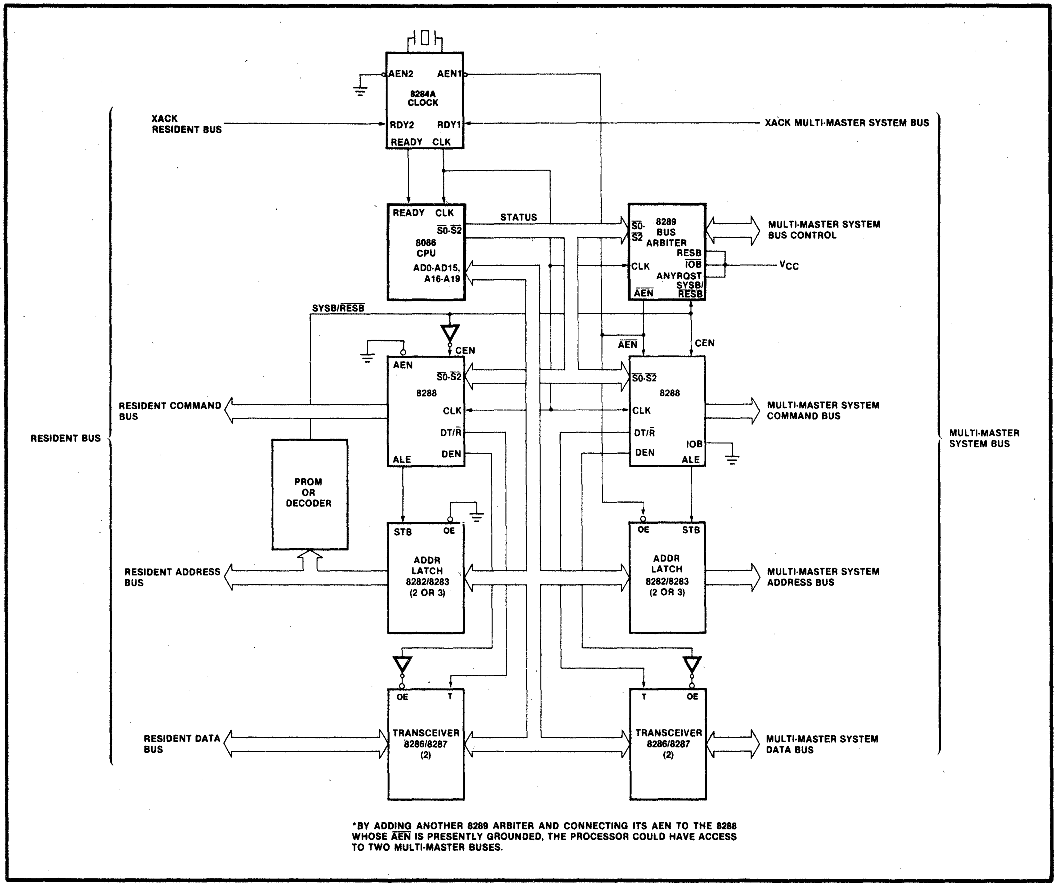


Figure 9. 8289 Bus Arbiter Shown in System-Resident Bus Configuration

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... 0°C to 70°C
Storage Temperature.............. −65°C to +150°C
All Output and Supply Voltages......... −0.5V to +7V
All Input Voltages.................. −1.0V to +5.5V
Power Dissipation........................ 1.5 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5V ±10%)

Symbol	Parameter	Min.	Max.	Units	Test Condition
V_C	Input Clamp Voltage		−1.0	V	V_{CC} = 4.50V, I_C = −5 mA
I_F	Input Forward Current		−0.5	mA	V_{CC} = 5.50V, V_F = 0.45V
I_R	Reverse Input Leakage Current		60	μA	V_{CC} = 5.50, V_R = 5.50
V_{OL}	Output Low Voltage $\overline{BUSY}$, $\overline{CBRQ}$ $\overline{AEN}$ $\overline{BPRO}$, $\overline{BREQ}$		 0.45 0.45 0.45	 V V V	 I_{OL} = 20 mA I_{OL} = 16 mA I_{OL} = 10 mA
V_{OH}	Output High Voltage $\overline{BUSY}$, $\overline{CBRQ}$		Open Collector		
	All Other Outputs	2.4		V	I_{OH} = 400 μA
I_{CC}	Power Supply Current		165	mA	
V_{IL}	Input Low Voltage		.8	V	
V_{IH}	Input High Voltage	2.0		V	
Cin Status	Input Capacitance		25	pF	
Cin (Others)	Input Capacitance		12	pF	

A.C. CHARACTERISTICS (V_{CC} = +5V ±10%, T_A = 0°C to 70°C)

TIMING REQUIREMENTS

Symbol	Parameter	8289 Min.	8289-1 Min.	Max.	Unit	Test Condition
TCLCL	CLK Cycle Period	125	100		ns	
TCLCH	CLK Low Time	65	53		ns	
TCHCL	CLK High Time	35	26		ns	
TSVCH	Status Active Setup	65	55	TCLCL-10	ns	
TSHCL	Status Inactive Setup	50	45	TCLCL-10	ns	
THVCH	Status Active Hold	10	10		ns	
THVCL	Status Inactive Hold	10	10		ns	
TBYSBL	BUSY↑↓Setup to BCLK↓	20	20		ns	
TCBSBL	CBRQ↑↓Setup to BCLK↓	20	20		ns	
TBLBL	BCLK Cycle Time	100	100		ns	
TBHCL	BLCK High Time	30	30	.65[TBLBL]	ns	
TCLLL1	LOCK Inactive Hold	10	10		ns	
TCLLL2	LOCK Active Setup	40	40		ns	
TPNBL	BPRN↑↓ to BCLK Setup Time	15	15		ns	
TCLSR1	SYSB/RESB Setup	0	0		ns	
TCLSR2	SYSB/RESB Hold	20	20		ns	
TIVIH	Initialization Pulse Width	3 TBLBL+ 3 TCLCL	3 TBLBL+ 3 TCLCL		ns	

A.C. CHARACTERISTICS (Continued)

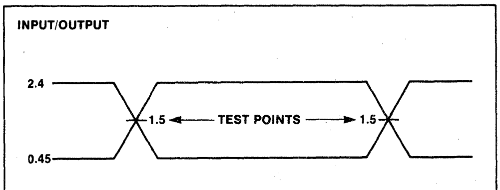
TIMING RESPONSES

Symbol	Parameter	Min.	Max.	Unit	Test Condition
TBLBRL	BCLK to BREQ Delay↓↑		35	ns	
TBLPOH	BCLK to BPRO↓↑ (See Note 1)		40	ns	
TPNPO	BPRN↓↑to BPRO↓↑Delay (See Note 1)		25	ns	
TBLBYL	BCLK to BUSY Low		60	ns	
TBLBYH	BCLK to BUSY Float (See Note 2)		35	ns	
TCLAEH	CLK to AEN High		65	ns	
TBLAEL	BCLK to AEN Low		40	ns	
TBLCBL	BCLK to CBRQ Low		60	ns	
TRLCRH	BCLK to CBRQ Float (See Note 2)		35	ns	
TOLOH	Output Rise Time		20	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		12	ns	From 2.0V to 0.8V

↓↑ Denotes that spec applies to both transitions of the signal.

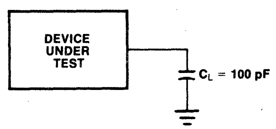
NOTES:
1. BCLK generates the first BPRO wherein subsequent BPRO changes lower in the chain are generated through BPRON.
2. Measured at .5V above GND.

A.C. TESTING INPUT, OUTPUT WAVEFORM



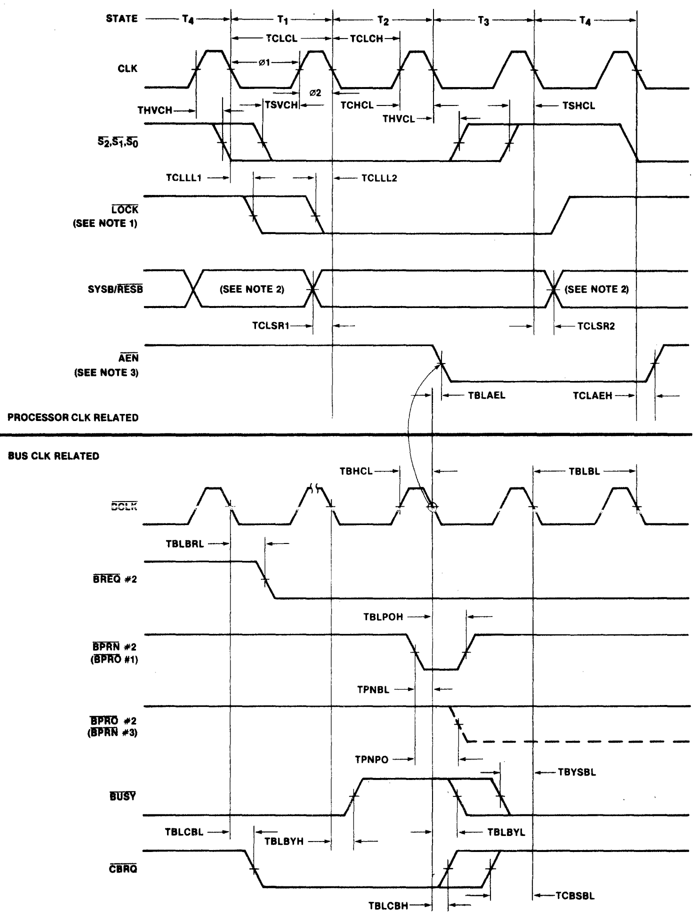
A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." THE CLOCK IS DRIVEN AT 4.3V and 0.25V. TIMING MEASUREMENTS ARE MADE AT 1.5V FOR BOTH A LOGIC "1" AND "0." INPUT RISE AND FALL TIMES (MEASURED BETWEEN 0.8V AND 2.0V) ARE DRIVEN AT 5 ± 2 NS.

A.C. TESTING LOAD CIRCUIT



C_L = 100 pF
C_L INCLUDES JIG CAPACITANCE

WAVEFORMS



NOTES:

1. LOCK ACTIVE CAN OCCUR DURING ANY STATE, AS LONG AS THE RELATIONSHIPS SHOWN ABOVE WITH RESPECT TO THE CLK ARE MAINTAINED. LOCK INACTIVE HAS NO CRITICAL TIME AND CAN BE ASYNCHRONOUS. CRQLCK HAS NO CRITICAL TIMING AND IS CONSIDERED AN ASYNCHRONOUS INPUT SIGNAL
2. GLITCHING OF SYSB/RESB PIN IS PERMITTED DURING THIS TIME. AFTER ϕ 2 OF T1, AND BEFORE ϕ1 OF T4, SYSB/RESB SHOULD BE STABLE.
3. AEN LEADING EDGE IS RELATED TO BCLK, TRAILING EDGE TO CLK. THE TRAILING EDGE OF AEN OCCURS AFTER BUS PRIORITY IS LOST.

ADDITIONAL NOTES:

The signals related to CLK are typical processor signals, and do not relate to the depicted sequence of events of the signals referenced to BCLK. The signals shown related to the BCLK represent a hypothetical sequence of events for illustration. Assume 3 bus arbiters of priorities 1, 2 and 3 configured in serial priority resolving scheme as shown in Figure 6. Assume arbiter 1 has the bus and is holding busy low. Arbiter #2 detects its processor wants the bus and pulls low BREQ#2. If BPRN#2 is high (as shown), arbiter #2 will pull low CBRQ line. CBRQ signals to the higher priority arbiter #1 that a lower priority arbiter wants the bus. [A higher priority arbiter would be granted BPRN when it makes the bus request rather than having to wait for another arbiter to release the bus through CBRQ].** Arbiter #1 will relinquish the multi-master system bus when it enters a state not requiring it (see Table 1), by lowering its BPRO#1 (tied to BPRN#2) and releasing BUSY. Arbiter #2 now sees that it has priority from BPRN#2 being low and releases CBRQ. As soon as BUSY signifies the bus is available (high), arbiter #2 pulls BUSY low on next falling edge of BCLK. Note that if arbiter #2 didn't want the bus at the time it received priority, it would pass priority to the next lower priority arbiter by lowering its BPRO #2 [TPNPO].

**Note that even a higher priority arbiter which is acquiring the bus through BPRN will momentarily drop CBRQ until it has acquired the bus.

intel®

APPLICATION NOTE

AP-67

September 1979

8086 System Design

George Alexy
Microcomputer Applications

ORDER NUMBER: 230792-001

8086 System Design

Contents

1. INTRODUCTION

The 8086 family, Intel's new series of microprocessors and system components, offers the designer an advanced system architecture which can be structured to satisfy a broad range of applications. The variety of speed, configuration and component selections available within the family enables optimization of a specific design to both cost and performance objectives. More important however, the 8086 family concept allows the designer to develop a family of systems providing multiple levels of enhancement within a single design and a growth path for future designs.

This application note is directed toward the implementation of the system hardware and will provide an introduction to a representative sample of the systems configurable with the 8086 CPU member of the family. Application techniques and timing analysis will be given to aid the designer in understanding the system requirements, advantages and limitations. Additional Intel publications the reader may wish to reference are the 8086 User's Manual (9800722A), 8086 Assembly Language Reference Guide (9800749A), AP-28A MULTIBUS™ Interfacing (98005876B), INTEL MULTIBUS® SPECIFICATION (9800683), AP-45 Using the 8202 Dynamic RAM Controller (9800809A), AP-51 Designing 8086, 8088, 8089 Multiprocessor Systems with the 8289 Bus Arbiter and AP-59 Using the 8259A Programmable Interrupt Controller. References to other Intel publications will be made throughout this note.

2. 8086 OVERVIEW AND BASIC SYSTEM CONCEPTS

2A. 8086 Bus Cycle Definition

The 8086 is a true 16-bit microprocessor with 16-bit internal and external data paths, one megabyte of memory address space (2**20) and a separate 64K byte (2**16) I/O address space. The CPU communicates with its external environment via a twenty-bit time multiplexed address, status and data bus and a command bus. To transfer data or fetch instructions, the CPU executes a bus cycle (Fig. 2A1). The minimum bus cycle consists of four CPU clock cycles called T states. During the first T state (T1), the CPU asserts an address on the twenty-bit

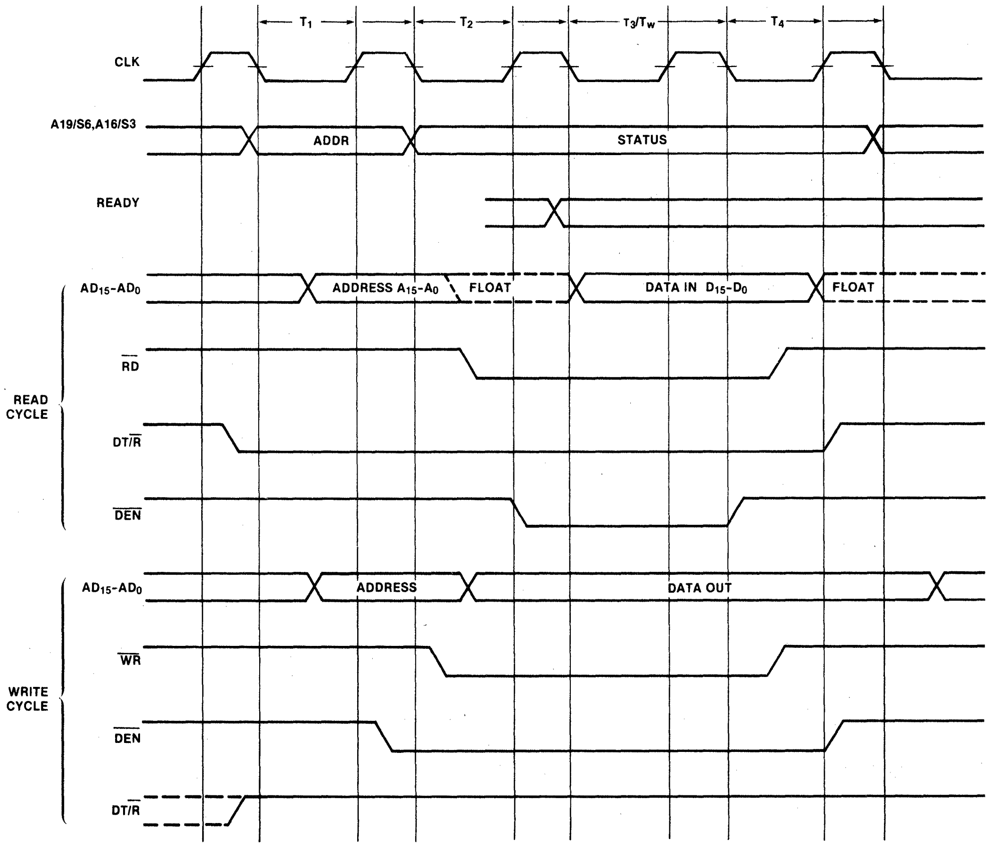


Figure 2A1. Basic 8086 Bus Cycle

multiplexed address/data/status bus. For the second T state (T2), the CPU removes the address from the bus and either three-states its outputs on the lower sixteen bus lines in preparation for a read cycle or asserts write data. Data bus transceivers are enabled in either T1 or T2 depending on the 8086 system configuration and the direction of the transfer (into or out of the CPU). Read, write or interrupt acknowledge commands are always enabled in T2. The maximum mode 8086 configuration (to be discussed later) also provides a write command enabled in T3 to guarantee data setup time prior to command activation.

During T2, the upper four multiplexed bus lines switch from address (A19-A16) to bus cycle status (S6,S5,S4,S3). The status information (Table 2A1) is available primarily for diagnostic monitoring. However, a decode of S3 and S4 could be used to select one of four banks of memory, one assigned to each segment register. This technique allows partitioning the memory by segment to expand the memory addressing beyond one megabyte. It also provides a degree of protection by preventing erroneous write operations to one segment from overlapping into another segment and destroying information in that segment.

The CPU continues to provide status information on the upper four bus lines during T3 and will either continue to assert write data or sample read data on the lower sixteen bus lines. If the selected memory or I/O device is not capable of transferring data at the maximum CPU transfer rate, the device must signal the CPU "not ready" and force the CPU to insert additional clock cycles (Wait states TW) after T3. The 'not ready' indication must be presented to the CPU by the start of T3. Bus activity during TW is the same as T3. When the selected device has had sufficient time to complete the transfer, it asserts "Ready" and allows the CPU to continue from the TW states. The CPU will latch the data on the bus during the last wait state or during T3 if no wait states are requested. The bus cycle is terminated in T4 (command lines are disabled and the selected external device deselects from the bus). The bus cycle appears to devices in the system as an asynchronous event consisting of an address to select the device followed by a read strobe or data and a write strobe. The selected device accepts bus data during a write cycle and drives the desired data onto the bus during a read cycle. On termination of the command, the device latches write data or disables its bus drivers. The only control the device has on the bus cycle is the insertion of wait cycles.

The 8086 CPU only executes a bus cycle when instructions or operands must be transferred to or from memory or I/O devices. When not executing a bus cycle, the bus interface executes idle cycles (TI). During the idle cycles, the CPU continues to drive status information from the previous bus cycle on the upper address lines. If the previous bus cycle was a write, the CPU continues to drive the write data onto the multiplexed bus until the start of the next bus cycle. If the CPU executes idle cycles following a read cycle, the CPU will not drive the lower 16 bus lines until the next bus cycle is required.

Since the CPU prefetches up to six bytes of the instruction stream for storage and execution from an internal instruction queue, the relationship of instruction fetch and associated operand transfers may be skewed in time and separated by additional instruction fetch bus cycles. In general, if an instruction is fetched into the 8086's internal instruction queue, several additional instructions may be fetched before the instruction is removed from the queue and executed. If the instruction being executed from the queue is a jump or other control transfer instruction, any instructions remaining in the queue are not executed and are discarded with no effect on the CPU's operation. The bus activity observed during execution of a specific instruction is dependent on the preceding instructions but is always deterministic within the specific sequence.

Table 2A1

S3	S4	
0	0	Alternate (relative to the ES segment)
1	0	Stack (relative to the SS segment)
0	1	Code/None (relative to the CS segment or a default of zero)
1	1	Data (relative to the DS segment)
S5 = IF (interrupt enable flag)		
S6 = 0 (indicates the 8086 is on the bus)		

2B. 8086 Address and Data Bus Concepts

Since the majority of system memories and peripherals require a stable address for the duration of the bus cycle, the address on the multiplexed address/data bus during T1 should be latched and the latched address used to select the desired peripheral or memory location. Since the 8086 has a 16-bit data bus, the multiplexed bus components of the 8085 family are not applicable to the 8086 (a device on address/data bus lines 8-15 will not be able to receive the byte selection address on lines 0-7). To demultiplex the bus (Fig. 2B1a), the 8086 system provides an Address Latch Enable signal (ALE) to capture the address in either the 8282 or 8283 8-bit bi-stable latches (Diag. 2B1). The latches are either inverting (8283) or non-inverting (8282) and have outputs driven by three-state buffers that supply 32 mA drive capability and can switch a 300 pF capacitive load in 22 ns (inverting) or 30 ns (non-inverting). They propagate the address through to the outputs while ALE is high and latch the address on the falling edge of ALE. This only delays address access and chip select decoding by the propagation delay of the latch. The outputs are enabled through the low active $\overline{OE}$ input. The demultiplexing of the multiplexed address/data bus (latchings of the address from the multiplexed bus), can be done locally at appropriate points in the system or at the CPU with a separate address bus distributing the address throughout the system (Fig. 2B1b). For optimum system performance and compatibility with multiprocessor and MULTIBUS™ configurations, the latter technique is strongly recommended over the first. The remainder of this note will assume the bus is demultiplexed at the CPU.

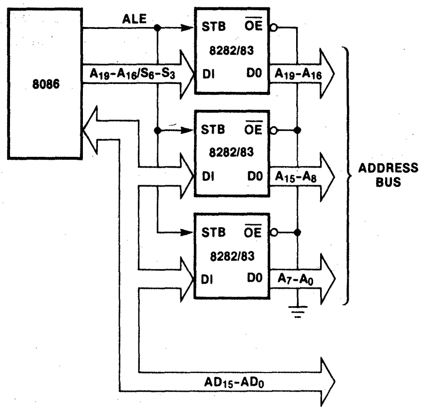


Figure 2B1a. Demultiplexing the 8086 Bus

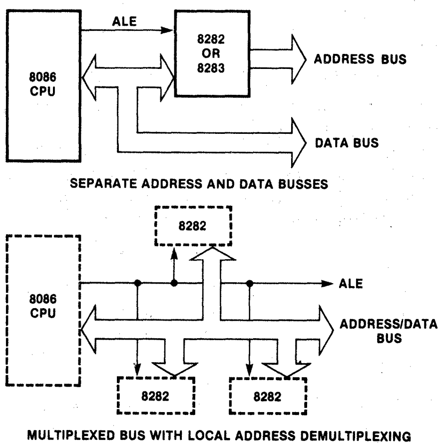


Figure 2B1b.

The programmer views the 8086 memory address space as a sequence of one million bytes in which any byte may contain an eight bit data element and any two consecutive bytes may contain a 16-bit data element. There is no constraint on byte or word addresses (boundaries). The address space is physically implemented on a sixteen bit data bus by dividing the address space into two banks of up to 512K bytes (Fig. 2B2). One bank is connected to the lower half of the sixteen-bit data bus (D7-0) and contains even addressed bytes (A0 = 0). The other bank is connected to the upper half of the data bus (D15-8) and contains odd addressed bytes (A0 = 1). A specific byte within each bank is selected by address lines A19-A1. To perform byte transfers to even addresses (Fig. 2B3a), the information is transferred over the lower half of the data bus (D7-0). A0 (active low) is used to enable the bank connected to the lower half of the data bus to participate in the transfer. Another signal provided by the 8086, Bus High Enable ($\overline{BHE}$), is used to disable the bank on the upper half of the data bus from participating in the transfer. This is necessary to prevent a write operation to the lower bank from destroying data in the upper bank. Since $\overline{BHE}$ is a multiplexed signal with timing identical to the A19-A16 address lines, it also should be latched with ALE to provide a stable signal during the bus cycle. During T2 through T4, the $\overline{BHE}$ output is multiplexed with status line S7 which is equal to $\overline{BHE}$. To perform byte transfers to odd addresses (Fig. 2B3b), the information is transferred over the upper half of the data bus (D15-D8) while $\overline{BHE}$ (active low) enables the upper bank and A0 disables the lower bank. Directing the data transfer to the appropriate half of the data bus and activation of $\overline{BHE}$ and A0 is performed by the 8086, transparent to the programmer. As an example, consider loading a byte of data into the CL register (lower half of the CX register) from an odd addressed memory location (referenced over the upper half of the 16-bit data bus). The data is transferred into the 8086 over the upper 8 bits of the data bus, automatically redirected to the lower half of the 8086 internal 16-bit data path and stored into the CL register. This capability also allows byte I/O transfers with the AL register to be directed to I/O devices connected to either the upper or lower half of the 16-bit data bus.

To access even addressed sixteen bit words (two consecutive bytes with the least significant byte at an even

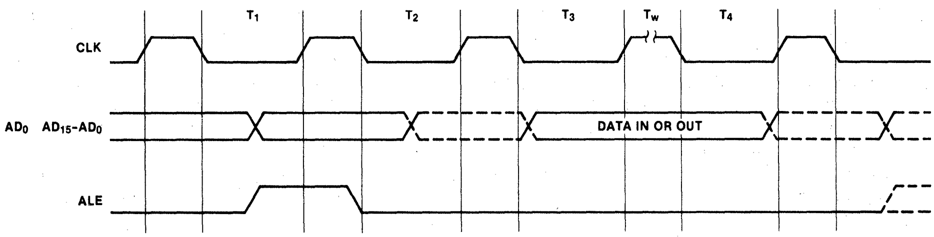


Diagram 2B1. ALE Timing

byte address), A19-A1 select the appropriate byte within each bank and A0 and $\overline{BHE}$ (active low) enable both banks simultaneously (Fig. 2B3c). To access an odd addressed 16-bit word (Fig. 2B3d), the least significant byte (addressed by A19-A1) is first transferred over the upper half of the bus (odd addressed byte, upper bank, $\overline{BHE}$ low active and A0 = 1). The most significant byte is accessed by incrementing the address (A19-A0) which allows A19-A1 to address the next physical word location (remember, A0 was equal to one which indicated a word referenced from an odd byte boundary). A second bus cycle is then executed to perform the transfer of the most significant byte with the lower bank (A0 is now active low and $\overline{BHE}$ is high). The sequence is automatically executed by the 8086 whenever a word transfer is executed to an odd address. Directing the upper and lower bytes of the 8086's internal sixteen-bit registers to the appropriate halves of the data bus is also performed automatically by the 8086 and is transparent to the programmer.

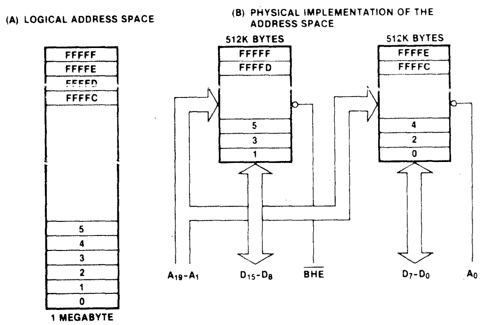


Figure 2B2. 8086 Memory

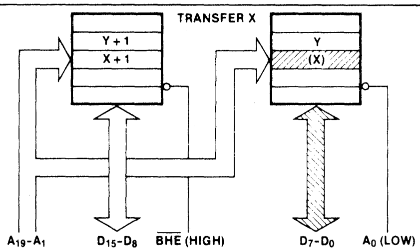


Figure 2B3a. Even Addressed Byte Transfer

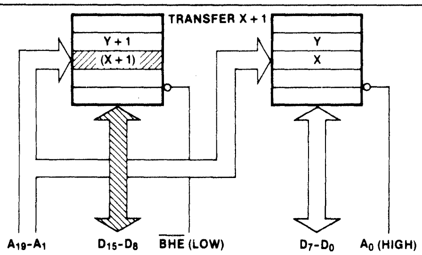


Figure 2B3b. Odd Addressed Byte Transfer

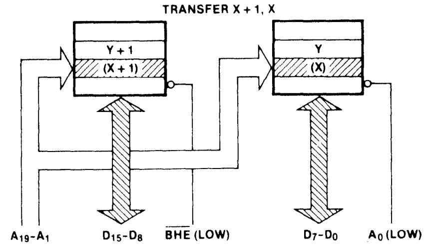


Figure 2B3c. Even Addressed Word Transfer

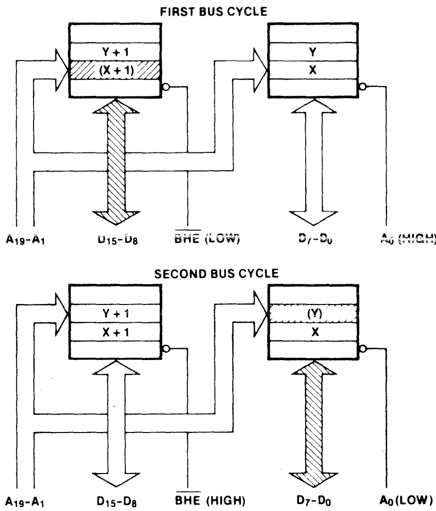


Figure 2B3d. Odd Addressed Word Transfer

During a byte read, the CPU floats the entire sixteen-bit data bus even though data is only expected on the upper or lower half of the data bus. As will be demonstrated later, this action simplifies the chip select decoding requirements for read only devices (ROM, EPROM). During a byte write operation, the 8086 will drive the entire sixteen-bit data bus. The information on the half of the data bus not transferring data is indeterminate. These concepts also apply to the I/O address space. Specific examples of I/O and memory interfacing are considered in the corresponding sections.

2C. System Data Bus Concepts

When referring to the system data bus, two implementation alternatives must be considered; (a) the multiplexed address/data bus (Fig. 2C1a) and a data bus buffered from the multiplexed bus by transceivers (Fig. 2C1b).

If memory or I/O devices are connected directly to the multiplexed bus, the designer must guarantee the devices do not corrupt the address on the bus during T1.

MULTIPLEXED DATA BUS

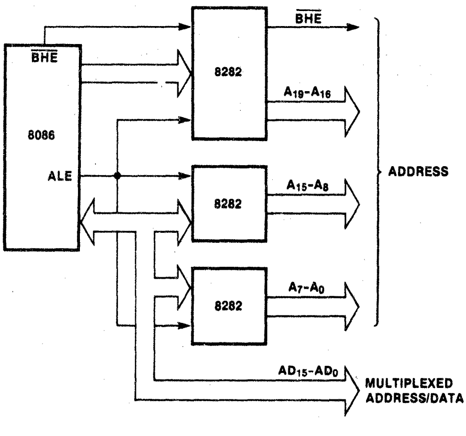


Figure 2C1a. Multiplexed Data Bus

BUFFERED DATA BUS

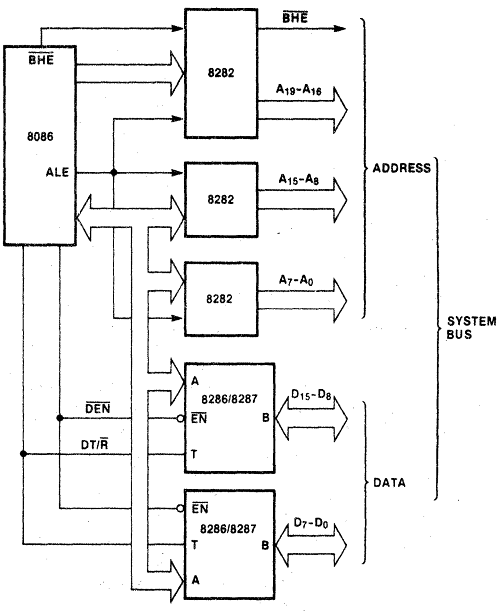


Figure 2C1b. Buffered Data Bus

To avoid this, device output drivers should not be enabled by the device chip select, but should have an output enable controlled by the system read signal (Fig. 2C2). The 8086 timing guarantees that read is not valid until after the address is latched by ALE (Diag. 2C1). All Intel peripherals, EPROM products and RAM's for microprocessors provide output enable or read inputs to allow connection to the multiplexed bus.

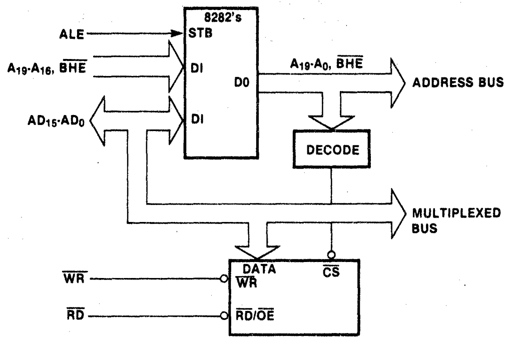


Figure 2C2. Devices with Output Enables on the Multiplexed Bus

Several techniques are available for interfacing devices without output enables to the multiplexed bus but each introduces other restrictions or limitations. Consider Figure 2C3 which has chip select gated with read and write. Two problems exist with this technique. First, the chip select access time is reduced to the read access time, and may require a faster device if maximum system performance (no wait states) is to be achieved (Diag. 2C2). Second, the designer must verify that chip select to write setup and hold times for the device are not violated (Diag. 2C3). Alternate techniques can be extracted from the bus interfacing techniques given later in this section but are subject to the associated restrictions. In general, the best solution is obtained with devices having output enables.

A subsequent limitation on the multiplexed bus is the 8086's drive capability of 2.0 mA and capacitive loading of 100 pF to guarantee the specified A.C. characteristics. Assuming capacitive loads of 20 pF per I/O device, 12 pF per address latch and 5-12 pF per memory device, a system mix of three peripherals and two to four memory devices (per bus line) are close to the loading limit.

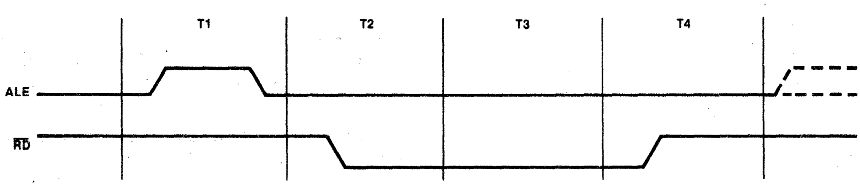


Diagram 2C1. Relationship of ALE to READ

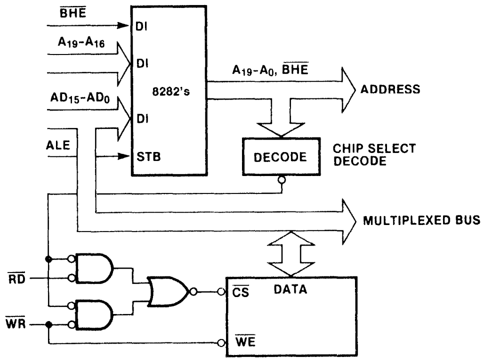


Figure 2C3. Devices without Output Enables on the Multiplexed Bus

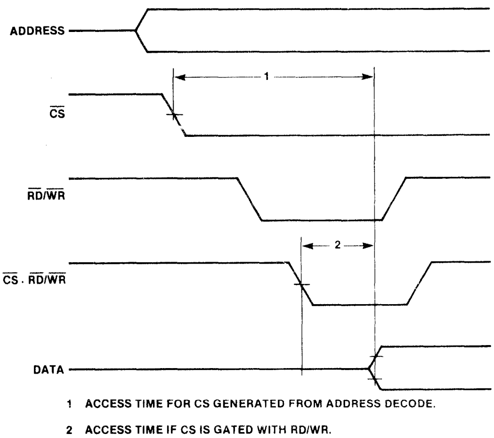


Diagram 2C2. Access Time: CS Gated with $\overline{RD}/\overline{WR}$

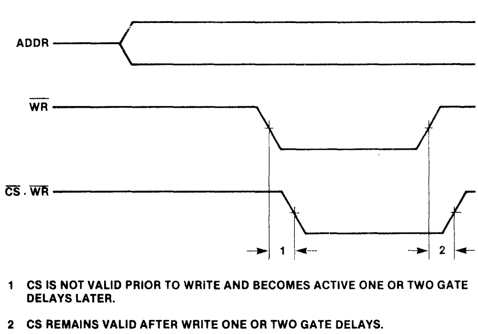
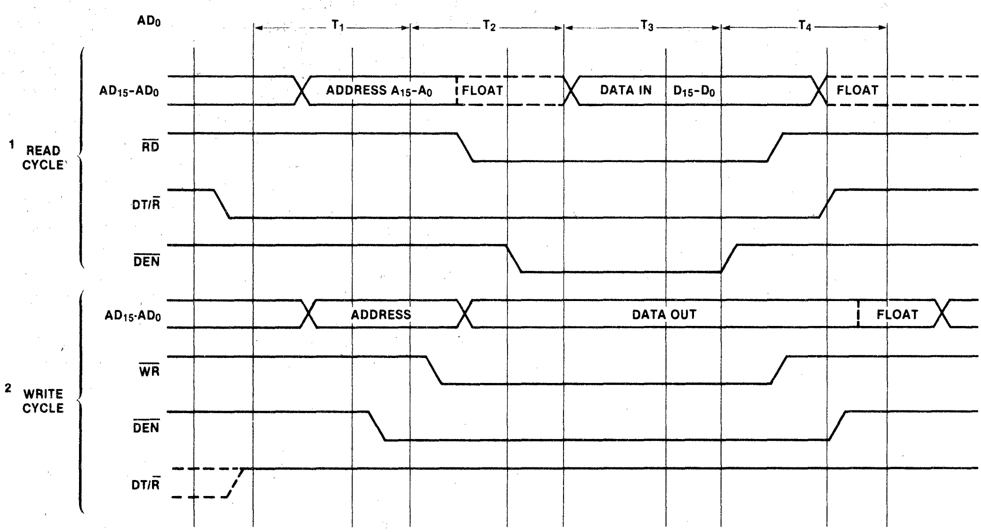


Diagram 2C3. CS to $\overline{WR}$ Set-Up and Hold

To satisfy the capacitive loading and drive requirements of larger systems, the data bus must be buffered. The 8286 non-inverting and 8287 inverting octal transceivers are offered as part of the 8086 family to satisfy this requirement. They have three-state output buffers that drive 32 mA on the bus interface and 10 mA on the CPU interface and can switch capacitive loads of 300 pF at the bus interface and 100 pF on the CPU interface in 22 ns (8287) or 30 ns (8286). To enable and control the direction of the transceivers, the 8086 system provides Data ENable (DEN) and Data Transmit/Receive (DT/$\overline{R}$) signals (Fig. 2C1b). These signals provide the appropriate timing to guarantee isolation of the multiplexed bus from the system during T1 and elimination of bus contention with the CPU during read and write (Diag. 2C4). Although the memory and peripheral devices are isolated from the CPU (Fig. 2C4), bus contention may still exist in the system if the devices do not have an output enable control other than chip select. As an example, bus contention will exist during transition from one chip select to another (the newly selected device begins driving the bus before the previous device has disabled its drivers). Another, more severe case exists during a write cycle. From chip select to write active, a device whose outputs are controlled only by chip select, will drive the bus simultaneously with write data being driven through the transceivers by the CPU (Diag. 2C5). The same technique given for circumventing these problems on the multiplexed bus can be applied here with the same limitations.

One last extension to the bus implementation is a second level of buffering to reduce the total load seen by devices on the system bus (Fig. 2C5). This is typically done for multiboard systems and isolation of memory arrays. The concerns with this configuration are the additional delay for access and more important, control of the second transceiver in relationship to the system bus and the device being interfaced to the system bus. Several techniques for controlling the transceiver are given in Figure 2C6. This first technique (Fig. 2C6a) simply distributes DEN and DT/$\overline{R}$ throughout the system. DT/$\overline{R}$ is inverted to provide proper direction control for the second level transceivers. The second example (Fig. 2C6b) provides control for devices with output enables. $\overline{RD}$ is used to normally direct data from the system bus to the peripheral. The buffer is selected whenever a device on the local bus is chip selected. Bus contention is possible on the device's local bus during a read as the read simultaneously enables the device output and changes the transceiver direction. The contention may also occur as the read is terminated.

For devices without output enables, the same technique can be applied (Fig. 2C6c) if the chip select to the device is conditioned by read or write. Controlling the chip select with read/write prevents the device from driving against the transceiver prior to the command being received. The limitations with this technique are access limited to read/write time and limited CS to write setup and hold times.



1 DEN IS ENABLED AFTER THE 8086 HAS FLOATED THE MULTIPLEXED BUS

2 DEN ENABLES THE TRANSCEIVERS EARLY IN THE CYCLE, BUT DT/R GUARANTEES THE TRANSCEIVERS ARE IN TRANSMIT RATHER THAN RECEIVE MODE AND WILL NOT DRIVE AGAINST THE CPU.

Diagram 2C4. Bus Transceiver Control

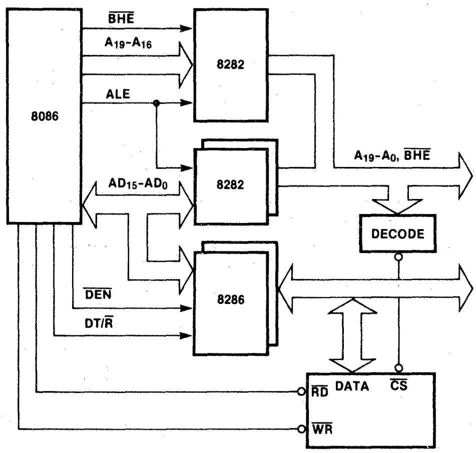


Figure 2C4. Devices with Output Enables on the System Bus

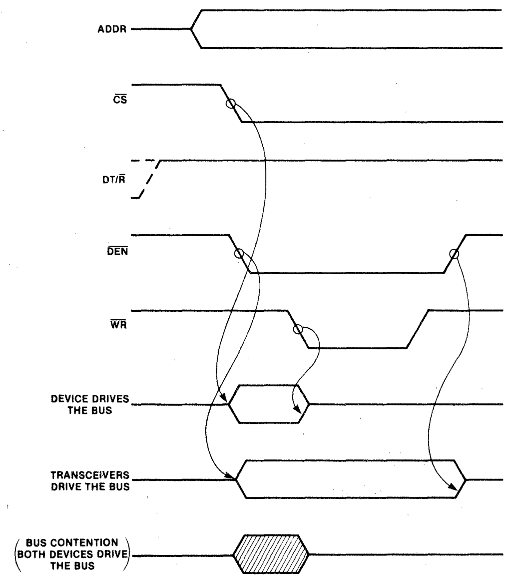


Diagram 2C5.

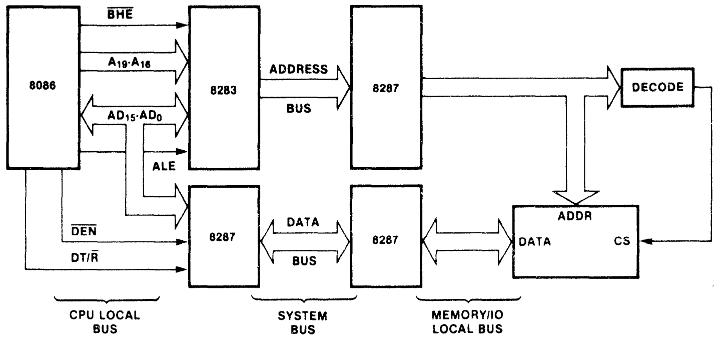


Figure 2C5. Fully Buffered System

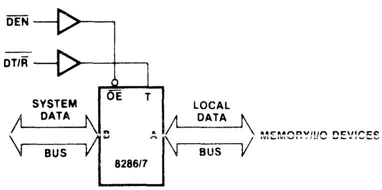


Figure 2C6a. Controlling System Transceivers with DEN and DT/R

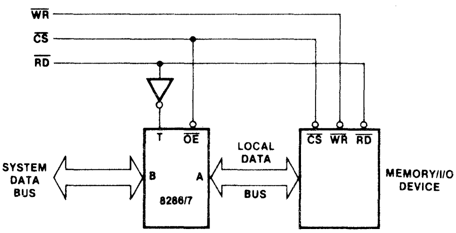


Figure 2C6b. Buffering Devices with OE/RD

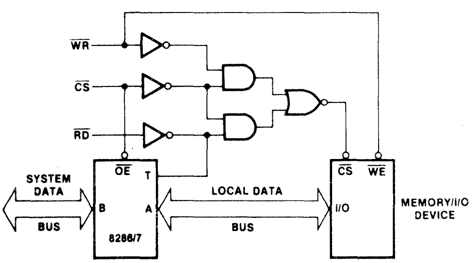


Figure 2C6c. Buffering Devices without OE/RD and with Common or Separate Input/Output

An alternate technique applicable to devices with and without output enables is shown in Figure 2C6d. RD again controls the direction of the transceiver but it is not enabled until a command and chip select are active. The possibility for bus contention still exists but is reduced to variations in output enable vs. direction change time for the transceiver. Full access time from chip select is now available, but data will not be valid prior to write and will only be held valid after write by the delay to disable the transceiver.

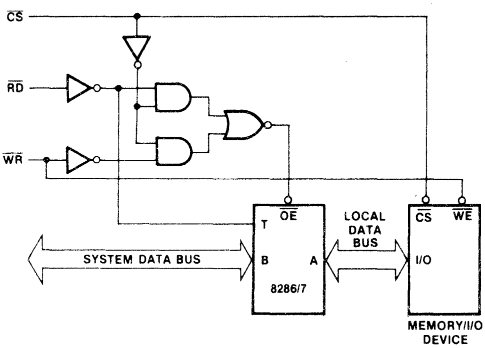


Figure 2C6d. Buffering Devices without OE/RD and with Common or Separate Input/Output

One last technique is given for devices with separate inputs and outputs (Fig. 2C6e). Separate bus receivers and drivers are provided rather than a single transceiver. The receiver is always enabled while the bus driver is controlled by RD and chip select. The only possibility for bus contention in this system occurs as multiple devices on each line of the local read bus are enabled and disabled during chip selection changes.

Throughout this note, the multiplexed bus will be considered the local CPU bus and the demultiplexed address and buffered data bus will be the system bus. For additional information on bus contention and the system problems associated with it, refer to Appendix 1.

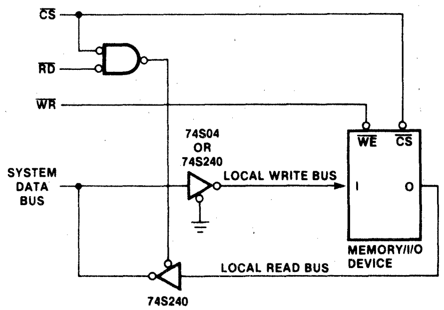


Figure 2C6e. Buffering Devices without OE/RD and with Separate Input/Output

2D. Multiprocessor Environment

The 8086 architecture supports multiprocessor systems based on the concept of a shared system bus (Fig. 2D1). All CPU's in the system communicate with each other and share resources via the system bus. The bus may be either the Intel Multibus™ system bus or an extension of the system bus defined in the previous section. The major addition required to the demultiplexed system bus is arbitration logic to control access to the system bus. As each CPU asynchronously requests access to the shared bus, the arbitration logic resolves priorities and grants bus access to the highest priority CPU. Having gained access to the bus, the CPU completes its transfer and will either relinquish the bus or wait to be forced to relinquish the bus. For a discussion on Multibus™ arbitration techniques, refer to AP-28A, Intel Multibus™ Interfacing.

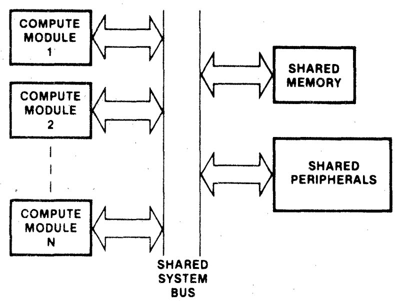


Figure 2D1. 8086 Family Multiprocessor System

To support a multimaster interface to the Multibus system bus for the 8086 family, the 8289 bus arbiter is included as part of the family. The 8289 is compatible with the 8086's local bus and in conjunction with the 8288 bus controller, implements the Multibus protocol for bus arbitration. The 8289 provides a variety of arbitration and prioritization techniques to allow optimization of bus availability, throughput and utilization of shared resources. Additional features (implemented through strapping options) extend the configuration options beyond a pure CPU interface to the multimaster system bus for access to shared resources to include concurrent support of a local CPU bus for private resources. For specific configurations and additional information on the 8289, refer to application note AP-51.

3. 8086 SYSTEM DETAILS

3A. Operating Modes

Possibly the most unique feature of the 8086 is the ability to select the base machine configuration most suited to the application. The MN/$\overline{\text{MX}}$ input to the 8086 is a strapping option which allows the designer to select between two functional definitions of a subset of the 8086 outputs.

MINIMUM MODE

The minimum mode 8086 (Fig. 3A1) is optimized for small to medium (one or two boards), single CPU systems. Its system architecture is directed at satisfying the requirements of the lower to middle segment of high performance 16-bit applications. The CPU maintains the full megabyte memory space, 64K byte I/O space and 16-bit data path. The CPU directly provides all bus control (DT/$\overline{\text{R}}$, $\overline{\text{DEN}}$, ALE, M/$\overline{\text{IO}}$), commands ($\overline{\text{RD}}$,$\overline{\text{WR}}$,$\overline{\text{INTA}}$) and a simple CPU preemption mechanism (HOLD, HLDA) compatible with existing DMA controllers.

MAXIMUM MODE

The maximum mode (Fig. 3A2) extends the system architecture to support multiprocessor configurations, and local instruction set extension processors (co-processors). Through addition of the 8288 bipolar bus controller, the 8086 outputs assigned to bus control and commands in the minimum mode are redefined to allow these extensions and enhance general system performance. Specifically, (1) two prioritized levels of processor preemption ($\overline{\text{RQ}}/\overline{\text{GT0}}$, $\overline{\text{RQ}}/\overline{\text{GT1}}$) allow multiple processors to reside on the 8086's local bus and share its interface to the system bus, (2) Queue status (QS0,QS1) is available to allow external devices like ICE™-86 or special instruction set extension co-processors to track the CPU instruction execution, (3) access control to shared resources in multiprocessor systems is supported by a hardware bus lock mechanism and (4) system command and configuration options are expanded via ancillary devices like the 8288 bus controller and 8289 bus arbiter.

The queue status indicates what information is being removed from the internal queue and when the queue is being reset due to a transfer of control (Table 3A1). By monitoring the $\overline{\text{S0}}$,$\overline{\text{S1}}$,$\overline{\text{S2}}$ status lines for instructions entering the 8086 (1,0,0 indicates code access while A0 and $\overline{\text{BHE}}$ indicate word or byte) and QS0, QS1 for instructions leaving the 8086's internal queue, it is possible to track the instruction execution. Since instructions are executed from the 8086's internal queue, the queue status is presented each CPU clock cycle and is not related to the bus cycle activity. This mechanism (1) allows a co-processor to detect execution of an

ESCAPE instruction which directs the co-processor to perform a specific task and (2) allows ICE-86 to trap execution of a specific memory location. An example of a circuit used by ICE is given in Figure 3A3. The first up down counter tracks the depth of the queue while the second captures the queue depth on a match. The second counter decrements on further fetches from the queue until the queue is flushed or the count goes to zero indicating execution of the match address. The first counter decrements on fetch from the queue (QS0 = 1) and increments on code fetches into the queue. Note that a normal code fetch will transfer two bytes into the queue so two clock increments are given to the counter (T201 and T301) unless a single byte is loaded over the upper half of the bus (A0-P is high). Since the execution unit (EU) is not synchronized to the bus interface unit (BIU), a fetch from the queue can occur simultaneously with a transfer into the queue. The exclusive-or gate driving the ENP input of the first counter allows these simultaneous operations to cancel each other and not modify the queue depth.

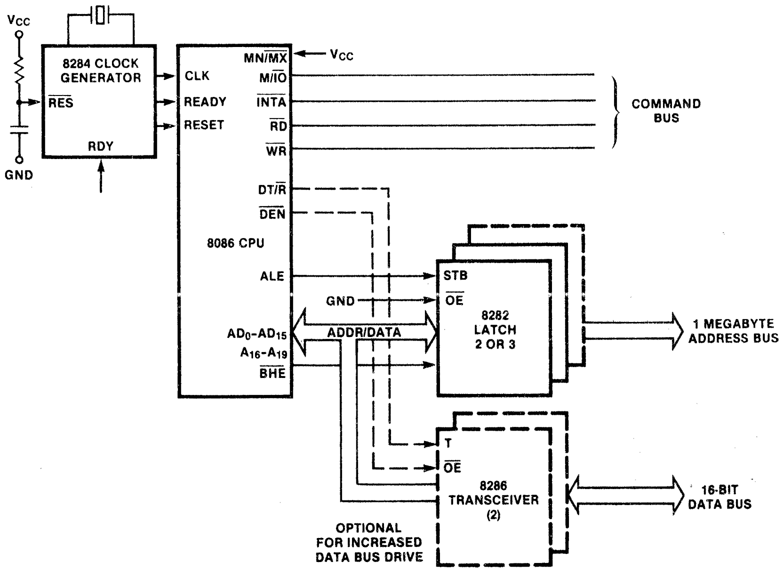


Figure 3A1. Minimum Mode 8086

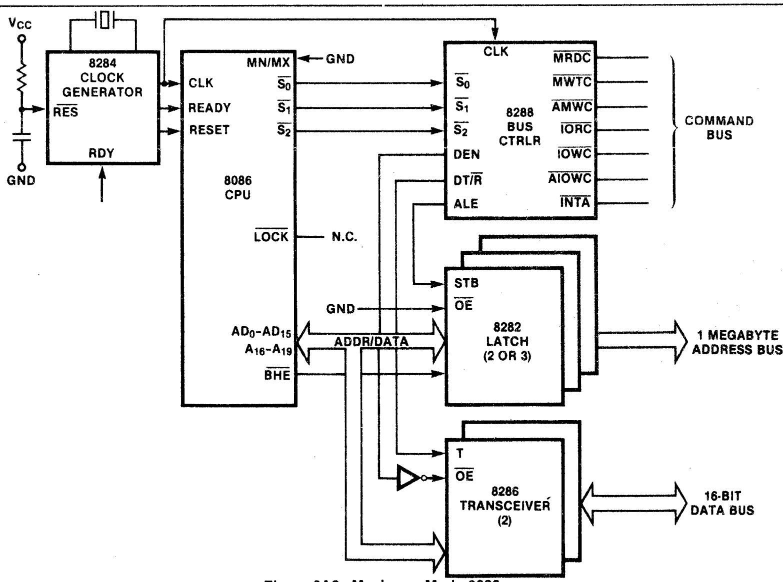


Figure 3A2. Maximum Mode 8086

TABLE 3A1. QUEUE STATUS

QS_1	QS_0	
0 (LOW)	0	No Operation
0	1	First Byte of Op Code from Queue
1 (HIGH)	0	Empty the Queue
1	1	Subsequent Byte from Queue

The queue status is valid during the CLK cycle after which the queue operation is performed.

To address the problem of controlling access to shared resources, the maximum mode 8086 provides a hardware $\overline{LOCK}$ output. The $\overline{LOCK}$ output is activated through the instruction stream by execution of the LOCK prefix instruction. The $\overline{LOCK}$ output goes active in the first CPU clock cycle following execution of the prefix and remains active until the clock following the completion of the instruction following the LOCK prefix. To provide bus access control in multiprocessor systems, the $\overline{LOCK}$ signal should be incorporated into the system bus arbitration logic resident to the CPU.

During normal multiprocessor system operation, priority of the shared system bus is determined by the arbitration circuitry on a cycle by cycle basis. As each CPU requires a transfer over the system bus, it requests access to the bus via its resident bus arbitration logic. When the CPU gains priority (determined by the system bus arbitration scheme and any associated logic), it takes control of the bus, performs its bus cycle and either maintains bus control, voluntarily releases the bus or is forced off the bus by the loss of priority. The lock mechanism prevents the CPU from losing bus control (either voluntarily or by force) and guarantees a CPU the ability to execute multiple bus cycles (during execution of the locked instruction) without intervention and possible corruption of the data by another CPU. A classic use of the mechanism is the 'TEST and SET semaphore' during which a CPU must read from a shared memory location and return data to the location without allowing another CPU to reference the same location between the TEST operation (read) and the SET operation (write). In the 8086 this is accomplished with a locked exchange instruction.

```
LOCK XCHG reg, MEMORY ; reg is any register
                      ;MEMORY is the address of the
                      ;semaphore
```

The activity of the $\overline{LOCK}$ output is shown in Diagram 3A1. Another interesting use of the $\overline{LOCK}$ for multiprocessor systems is a locked block move which allows high speed message transfer from one CPU's message buffer to another.

During the locked instruction, a request for processor preemption ($\overline{RQ}/\overline{GT}$) is recorded but not acknowledged until completion of the locked instruction. The $\overline{LOCK}$ has no direct affect on interrupts. As an example, a locked HALT instruction will cause HOLD (or $\overline{RQ}/\overline{GT}$) requests to be ignored but will allow the CPU to exit the HALT state on an interrupt. In general, prefix bytes are considered extensions of the instructions they precede. Therefore, interrupts that occur during execution of a prefix are not acknowledged (assuming interrupts are enabled) until completion of the instruction following the prefixes (except for instructions which allow servicing interrupts during their execution, i.e., HALT, WAIT and repeated string primitives). Note that multiple prefix bytes may precede an instruction. As another example, consider a 'string primitive' preceded by the repetition

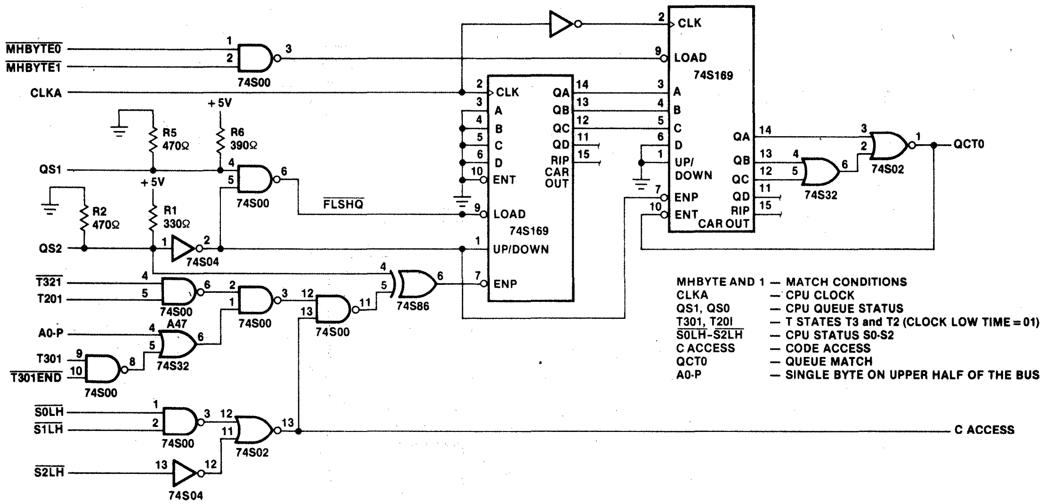


Figure 3A3. Example Circuit to Track the 8086 Queue prefix (REP) which is interruptible after each execution of the string primitive. This holds even if the REP prefix is combined with the LOCK prefix and prevents interrupts from being locked out during a block move or other repeated string operation. As long as the operation is not interrupted, $\overline{LOCK}$ remains active. Further information on the operation of an interrupted string operation with multiple prefixes is presented in the section dealing with the 8086 interrupt structure.

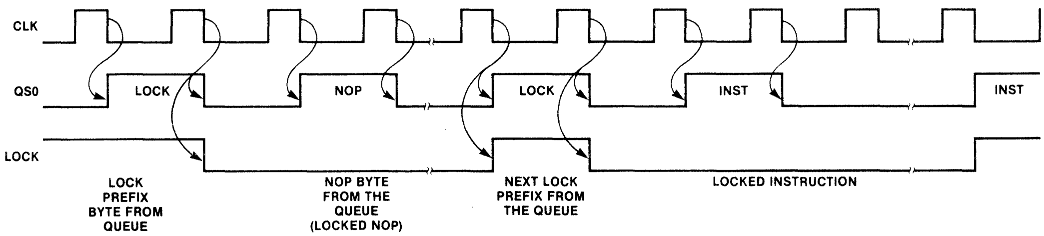
Three additional status lines ($\overline{S0}$, $\overline{S1}$, $\overline{S2}$) are defined to provide communications with the 8288 and 8289. The status lines tell the 8288 when to initiate a bus cycle, what type of command to issue and when to terminate the bus cycle. The 8288 samples the status lines at the beginning of each CPU clock (CLK). To initiate a bus cycle, the CPU drives the status lines from the passive state ($\overline{S0}$, $\overline{S1}$, $\overline{S2} = 1$) to one of seven possible command codes (Table 3A2). This occurs on the rising edge of the clock during T4 of the previous bus cycle or a TI (idle cycle, no current bus activity). The 8288 detects the status change by sampling the status lines on the high to low transition of each clock cycle. The 8288 starts a bus cycle by generating ALE and appropriate buffer direction control in the clock cycle immediately following detection of the status change (T1). The bus transceivers and the selected command are enabled in the next clock cycle (T2) (or T3 for normal write commands). When the status returns to the passive state, the 8288 will terminate the command as shown in Diagram 3A2. Since the CPU will not return the status to the passive state until the 'ready' indication is received, the 8288 will maintain active command and bus control for any number of wait cycles. The status lines may also be used by other processors on the 8086's local bus to monitor bus activity and control the 8288 if they gain control of the local bus.

TABLE 3A2. STATUS LINE DECODES

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	Halt
1 (HIGH)	0	0	Code Access
1	0	1	Read Memory
1	1	0	Write Memory
1	1	1	Passive

The 8288 provides the bus control (DEN, DT/$\overline{R}$, ALE) and commands ($\overline{INTA}$, $\overline{MRDC}$, $\overline{IORC}$, $\overline{MWTC}$, $\overline{AMWC}$, $\overline{IOWC}$, $\overline{AIOWC}$) removed from the CPU. The command structure has separate read and write commands for memory and I/O to provide compatibility with the Multibus command structure.

The advanced write commands are enabled one clock period earlier than the normal write to accommodate the wider write pulse widths often required by peripherals and static RAMs. The normal write provides data setup prior to write to accommodate dynamic RAM memories and I/O devices which strobe data on the leading edge of write. The advanced write commands do not guarantee that data is valid prior to the leading edge of the command. The DEN signal in the maximum mode is inverted from the minimum mode to extend transceiver control by allowing logical conjunction of DEN with other signals. While not appearing to be a significant benefit in the basic maximum mode configuration, introduction of interrupt control and various system configurations will demonstrate the usefulness of qualifying DEN. Diagram 3A3 compares the timing of the minimum and maximum mode bus transfer commands. Although the



1 QUEUE STATUS INDICATES FIRST BYTE OF OPCODE FROM THE QUEUE.

2 THE $\overline{LOCK}$ OUTPUT WILL GO INACTIVE BETWEEN SEPARATE LOCKED INSTRUCTIONS.

3 TWO CLOCKS ARE REQUIRED FOR DECODE OF THE LOCK PREFIX AND ACTIVATION OF THE $\overline{LOCK}$ SIGNAL.

4 SINCE QUEUE STATUS REFLECTS THE QUEUE OPERATION IN THE PREVIOUS CLOCK CYCLE, THE $\overline{LOCK}$ OUTPUT ACTUALLY GOES ACTIVE COINCIDENT WITH THE START OF THE NEXT INSTRUCTION AND REMAINS ACTIVE FOR ONE CLOCK CYCLE FOLLOWING THE INSTRUCTION.

5 IF THE INSTRUCTION FOLLOWING THE LOCK PREFIX IS NOT IN THE QUEUE, THE $\overline{LOCK}$ OUTPUT STILL GOES ACTIVE AS SHOWN WHILE THE INSTRUCTION IS BEING FETCHED.

6 THE BIU WILL STILL PERFORM INSTRUCTION FETCH CYCLES DURING EXECUTION OF A LOCKED INSTRUCTION. THE $\overline{LOCK}$ MERELY LOCKS THE BUS TO THIS CPU FOR WHATEVER BUS CYCLES THE CPU PERFORMS DURING THE LOCKED INSTRUCTION.

Diagram 3A1. 8086 Lock Activity

maximum mode configuration is designed for multi-processor environments, large single CPU designs (either Multibus systems or greater than two PC boards) should also use the maximum mode. Since the 8288 is a bipolar dedicated controller device, its output drive for the commands (32 mA) and tolerances on AC characteristics (timing parameters and worse case delays) provide better large system performance than the minimum mode 8086.

In addition to assuming the functions removed from the CPU, the 8288 provides additional strapping options and controls to support multiprocessor configurations and peripheral devices on the CPU local bus. These capabilities allow assigning resources (memory or I/O) as shared (available on the Multibus system bus) or private (accessible only by this CPU) to reduce contention for access to the Multibus system bus and improve multi-CPU system performance. Specific configuration possibilities are discussed in AP-51.

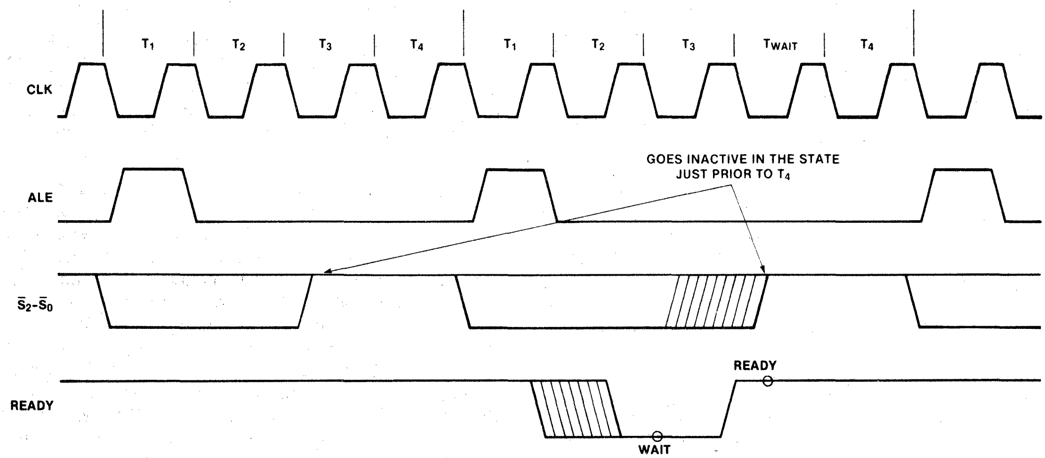


Diagram 3A2. Status Line Activation and Termination

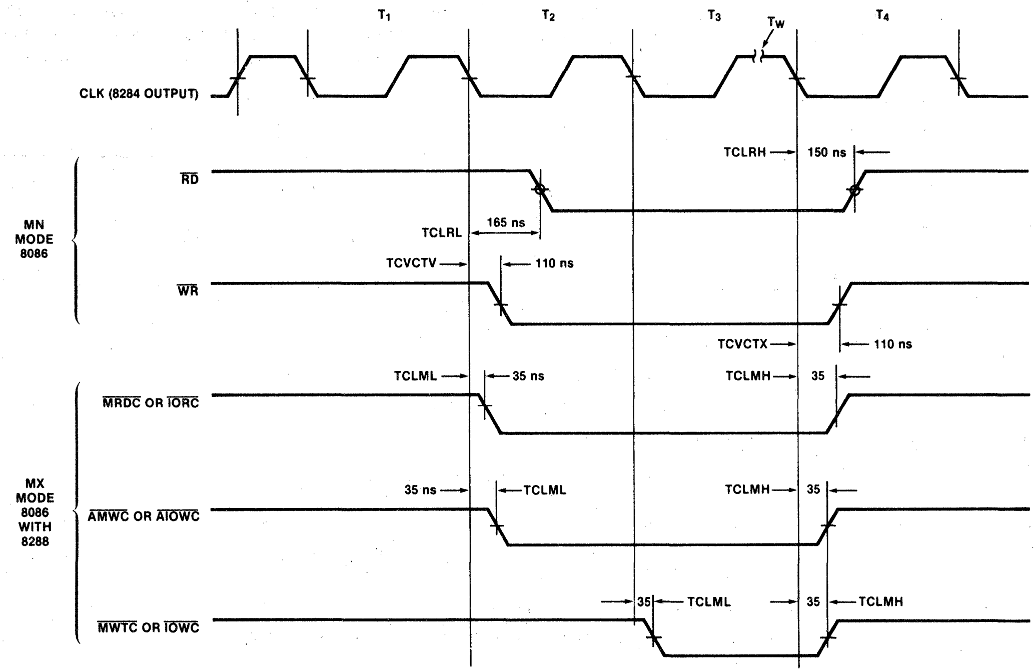


Diagram 3A3. 8086 Minimum and Maximum Mode Command Timing

3B. Clock Generation

The 8086 requires a clock signal with fast rise and fall times (10 ns max) between low and high voltages of −0.5 to +0.6 low and 3.9 to VCC+1.0 high. The maximum clock frequency of the 8086 is 5 MHz and 8 MHz for the 8086-2. Since the design of the 8086 incorporates dynamic cells, a minimum frequency of 2 MHz is required to retain the state of the machine. Due to the minimum frequency requirement, single stepping or cycling of the CPU may not be accomplished by disabling the clock. The timing and voltage requirements of the CPU clock are shown in Figure 3B1. In general, for frequencies below the maximum, the CPU clock need not satisfy the frequency dependent pulse width limitations stated in the 8086 data sheet. The values specified only reflect the minimum values which must be satisfied and are stated in terms of the maximum clock frequency. As the clock frequency approaches the maximum frequency of the CPU, the clock must conform to a 33% duty cycle to satisfy the CPU minimum clock low and high time specifications.

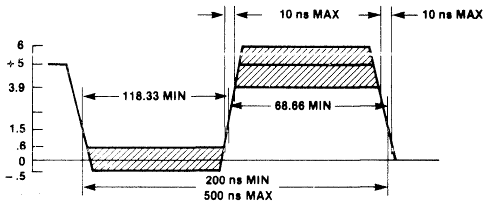


Figure 3B1. 8086 Clock

An optimum 33% duty cycle clock with the required voltage levels and transition times can be obtained with the 8284 clock generator (Fig. 3B2). Either an external frequency source or a series resonant crystal may drive the 8284. The selected source must oscillate at 3X the desired CPU frequency. To select the crystal inputs of the 8284 as the frequency source for clock generation, the F/$\overline{C}$ input to the 8284 must be strapped to ground. The strapping option allows selecting either the crystal or the external frequency input as the source for clock generation. Although the 8284 provides an input for a tank circuit to accommodate overtone mode crystals, fundamental mode crystals are recommended for more accurate and stable frequency generation. When selecting a crystal for use with the 8284, the series resistance should be as low as possible. Since other circuit components will tend to shift the operating frequency from resonance, the operating impedance will typically be higher than the specified series resistance. If the attenuation of the oscillator's feedback circuit reduces the loop gain to less than one, the oscillator will fail. Since the oscillator delays in the 8284 appear as inductive elements to the crystal, causing it to run at a frequency below that of the pure series resonance, a capacitor should be placed in series with the crystal and the X2 input of the 8284. This capacitor serves to cancel this inductive element. The value of the capacitor (CL) must not cause the impedance of the feedback circuit to reduce the loop gain below one. The impedance of the capacitor is a function of the operating frequency and can be determined from the following equation:

$$XCL = 1/2\pi * F * CL$$

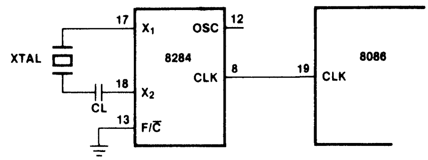


Figure 3B2. 8284 Clock Generator

It is recommended that the crystal series resistance plus XCL be kept less than 1K ohms. This capacitor also serves to debias the crystal and prevent a DC voltage bias from straining and perhaps damaging the crystalline structure. As the crystal frequency increases, the amount of capacitance should be decreased. For example, a 12 MHz crystal may require CL ~ 24 pF while 22 MHz may require CL ~ 8 pF. If very close correlation with the pure series resonance is not necessary, a nominal CL value of 12-15 pF may be used with a 15 MHz crystal (5 MHz 8086 operation). Board layout and component variances will affect the actual amount of inductance and therefore the series capacitance required to cancel it out (this is especially true for wire-wrapped layouts).

Two of the many vendors which supply crystals for Intel microprocessors are listed in Table 3B1 along with a list of crystal part numbers for various frequencies which may be of interest. For additional information on specifying crystals for Intel components refer to application note AP-35.

TABLE 3B1. CRYSTAL VENDORS

f	Parallel/ Series	Crystek[1] Corp.	CTS Knight,[2] Inc.
15.0 MHz	S	CY15A	MP150
18.432	S	CY19B*	MP184*
24.0 MHz	S	CY24A	MP240

*Intel also supplies a crystal numbered 8801 for this application.

Notes: 1. Address: 1000 Crystal Drive, Fort Meyers, Florida 33901
2. Address: 400 Reimann Ave., Sandwich, Illinois

If a high accuracy frequency source, externally variable frequency source or a common source for driving multiple 8284's is desired, the External Frequency Input (EFI) of the 8284 can be selected by strapping the F/$\overline{C}$ input to 5 volts through ~1K ohms (Fig. 3B3). The external frequency source should be TTL compatible, have a 50% duty cycle and oscillate at three times the desired CPU operating frequency. The maximum EFI frequency the 8284 can accept is slightly above 24 MHz with minimum clock low and high times of 13 ns. Although no minimum EFI frequency is specified, it should not violate the CPU minimum clock rate. If a common frequency source is used to drive multiple 8284's distributed throughout the system, each 8284 should be driven by its own line from the source. To minimize noise in the system, each line should be a twisted pair driven by a buffer like the 74LS04 with the ground of the twisted pair connecting the grounds of the source and receiver. To minimize clock skew, the lines to all 8284's should be of equal length. A simple technique for generating a master frequency source for additional 8284's is shown in Figure 3B4. One 8284 with a crystal is used to generate the desired frequency. The oscillator output of the 8284 (OSC) equals the crystal frequency and is used to drive the external frequency to all other 8284's in the system.

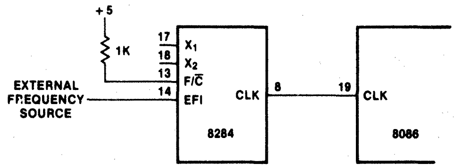


Figure 3B3. 8284 with External Frequency Source

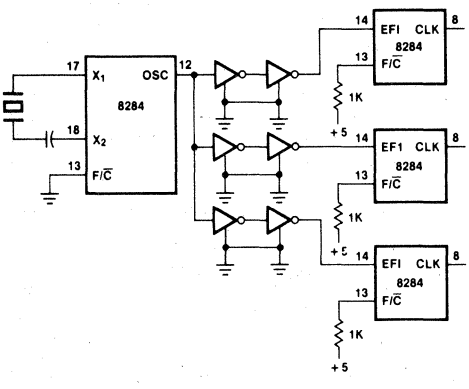


Figure 3B4. External Frequency for Multiple 8284s

The oscillator output is inverted from the oscillator signal used to drive the CPU clock generator circuit. Therefore, the oscillator output of one 8284 should not drive the EFI input of a second 8284 if both are driving clock inputs of separate CPU's that are to be synchronized. The variation on EFI to CLK delay over a range of 8284's may approach 35 to 45 ns. If, however, all 8284's are of the same package type, have the same relative supply voltage and operate in the same temperature environment, the variation will be reduced to between 15 and 25 ns.

There are three frequency outputs from the 8284, the oscillator (OSC) mentioned above, the system clock (CLK) which drives the CPU, and a peripheral clock (PCLK) that runs at one half the CPU clock frequency. The oscillator output is only driven by the crystal and is not affected by the F/C̄ strapping option. If a crystal is not connected to the 8284 when the external frequency input is used, the oscillator output is indeterminate. The CPU clock is derived from the selected frequency source by an internal divide by three counter. The counter generates the 33% duty cycle clock which is optimum for the CPU at maximum frequency. The peripheral clock has a 50% duty cycle and is derived from the CPU clock. Diagram 3B0 shows the relationship of CLK to OSC and PCLK to CLK. The maximum skew is 20 ns between OSC and CLK, and 22 ns between CLK and PCLK.

Since the state of the 8284 divide by three counter is indeterminate at system initialization (power on), an external sync to the counter (CSYNC) is provided to allow synchronization of the CPU clock to an external event. When CSYNC is brought high, the CLK and PCLK outputs are forced high. When CSYNC returns low, the next positive clock from the frequency source starts clock generation. CSYNC must be active for a minimum of two periods of the frequency source. If CSYNC is asynchronous to the frequency source, the circuit in Figure 3B5 should be used for synchronization. The two latches minimize the probability of a meta-stable state in the latch driving CSYNC. The latches are clocked with the inverse of the frequency source to guarantee the 8284 setup and hold time of CSYNC to the frequency source (Diag. 3B1). If a single 8284 is to be synchronized to an external event and an external frequency source is not used, the oscillator output of the 8284 may be used to

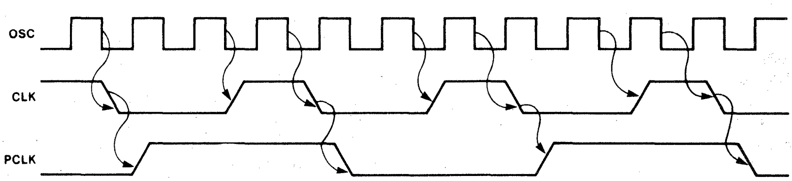


Diagram 3B0. OSC → CLK and CLK → PCLK Relationships synchronize CSYNC (Fig. 3B6). Since the oscillator output is inverted from the internal oscillator signal, the inverter in the previous example is not required. If multiple 8284's are to be synchronized, an external frequency source must drive all 8284's and a single CSYNC synchronization circuit must drive the CSYNC input of all 8284's (Fig. 3B7). Since activation of CSYNC may cause violation of CPU minimum clock low time, it should only be enabled during reset or CPU clock high. CSYNC must also be disabled a minimum of four CPU clocks before the end of reset to guarantee proper CPU reset.

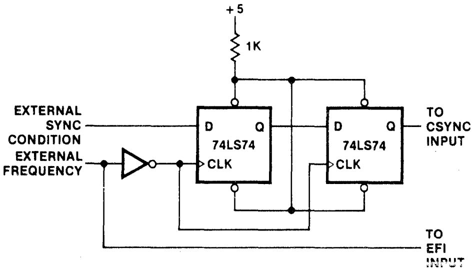
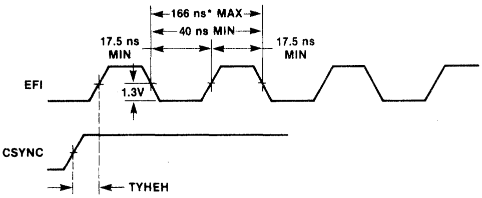


Figure 3B5. Synchronizing CSYNC with EFI



*MAX IS SPEC'ED TO GUARANTEE MAX 8086 CLOCK FREQUENCY

Diagram 3B1. CSYNC Setup and Hold to EFI

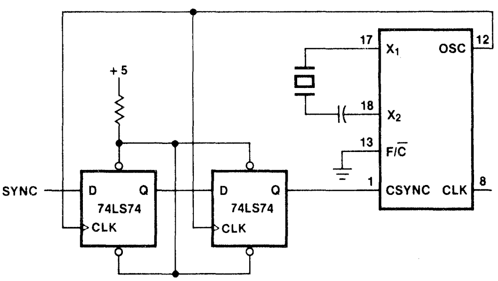


Figure 3B6. EFI from 8284 Oscillator

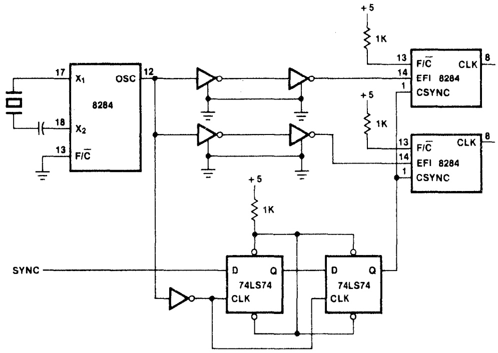


Figure 3B7. Synchronizing Multiple 8284s

Due to the fast transitions and high drive (5 mA) of the 8284 CLK output, it may be necessary to put a 10 to 100 ohm resistor in series with the clock line to eliminate ringing (resistor value depending on the amount of drive required). If multiple sources of CLK are needed with minimum skew, CLK can be buffered by a high drive device (74S241) with outputs tied to 5 volts through 100 ohms to guarantee VOH = 3.9 min (8086 minimum clock input high voltage) (Fig. 3B8). A single 8284 should not be used to generate the CLK for multiple CPU's that do not share a common local (multiplexed) bus since the 8284 synchronizes ready to the CPU and can only accommodate ready for a single CPU. If multiple CPU's share a local bus, they should be driven with the same clock to optimize transfer of bus control. Under these circumstances, only one CPU will be using the bus for a particular bus cycle which allows sharing a common READY signal (Fig. 3B9).

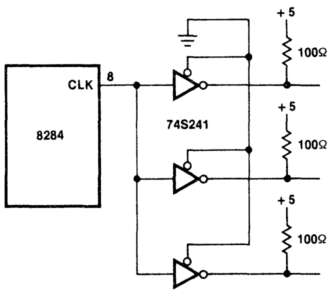


Figure 3B8. Buffering the 8284 CLK Output

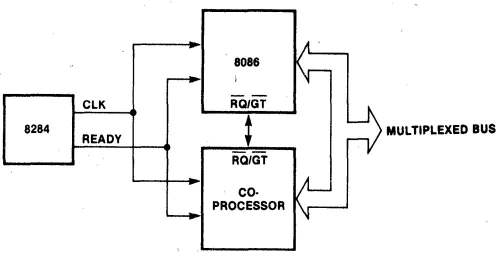


Figure 3B9. 8086 and Co-Processor on the Local Bus Share a Common 8284

3C. Reset

The 8086 requires a high active reset with minimum pulse width of four CPU clocks except after power on which requires a 50 μs reset pulse. Since the CPU internally synchronizes reset with the clock, the reset is internally active for up to one clock period after the external reset. Non-Maskable Interrupts (NMI) or hold requests on RQ/GT which occur during the internal reset, are not acknowledged. A minimum mode hold request or maximum mode RQ pulses active immediately after the internal reset will be honored before the first instruction fetch.

From reset, the 8086 will condition the bus as shown in Table 3C1. The multiplexed bus will three-state upon detection of reset by the CPU. Other signals which three-state will be driven to the inactive state for one clock low interval prior to entering three-state (Fig. 3C1). In the minimum mode, ALE and HLDA are driven inactive and are not three-stated. In the maximum mode, RQ/GT lines are held inactive and the queue status indicates no activity. The queue status will not indicate a reset of the queue so any user defined external circuits monitoring the queue should also be reset by the system reset. 22K ohm pull-up resistors should be connected to the CPU command and bus control lines to guarantee the inactive state of these lines in systems where leakage currents or bus capacitance may cause the voltage levels to settle below the minimum high voltage of devices in the system. In maximum mode systems, the 8288 contains internal pull-ups on the S0-S2 inputs to maintain the inactive state for these lines when the CPU floats the bus. The high state of the status lines during reset causes the 8288 to treat the reset sequence as a passive state. The condition of the 8288 outputs for the passive state are shown in Table 3C2. If the reset occurs during a bus cycle, the return of the status lines to the passive state will terminate the bus cycle and return the command lines to the inactive state. Note that the 8288 does not three-state the command outputs based on the passive state of the status lines. If the designer needs to three-state the CPU off the bus during reset in a single CPU system, the reset signal should also be connected to the 8288's AEN input and the output enable of the address latches (Fig. 3C2). This forces the command and address bus interface to three-state while the inactive state of DEN from the 8288 three-states the transceivers on the data bus.

Table 3C1. 8086 Bus During Reset

Signals	Condition
$AD_{15\text{-}0}$	Three-State
$A_{19\text{-}16}/S_{6\text{-}3}$	Three-State
BHE/S_7	Three-State
S2/(M/IO)	Driven to "1" then three-state
S1/(DT/R)	Driven to "1" then three-state
S0/DEN	Driven to "1" then three-state
LOCK/WR	Driven to "1" then three-state
RD	Driven to "1" then three-state
INTA	Driven to "1" then three-state
ALE	0
HLDA	0
RQ/GT0	1
RQ/GT1	1
QS0	0
QS1	0

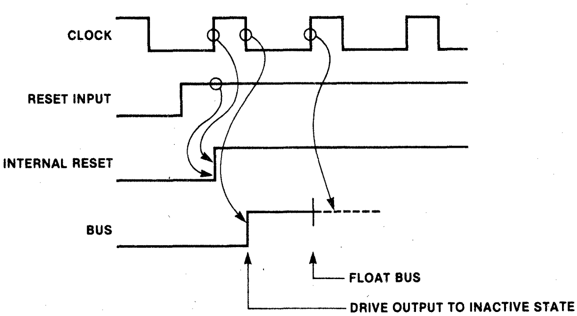


Figure 3C1. 8086 Bus Conditioning on Reset

TABLE 3C2. 8288 OUTPUTS DURING PASSIVE MODE

ALE	0
DEN	0
DT/$\overline{R}$	1
MCE/$\overline{PDEN}$	0/1
COMMANDS	1

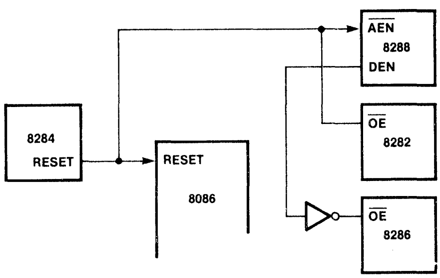


Figure 3C2. Reset Disable for Max Mode 8086 Bus Interface

For multiple processor systems using arbitration of a multimaster bus, the system reset should be connected to the $\overline{INIT}$ input of the 8289 bus arbiter in addition to the 8284 reset input (Fig. 3C3). The low active $\overline{INIT}$ input forces all 8289 outputs to their inactive state. The inactive state of the 8289 $\overline{AEN}$ output will force the 8288 to three-state the command outputs and the address latches to three-state the address bus interface. DEN inactive from the 8288 will three-state the data bus interface. For the multimaster CPU configuration, the reset should be common to all CPU's (8289's and 8284's) and satisfy the maximum of either the CPU reset requirements or 3 TBLBL (3 8289 bus clock times) + 3 TCLCL (3 8086 clock cycle times) to satisfy 8289 reset requirements.

If the 8288 command outputs are three-stated during reset, the command lines should be pulled up to V_{CC} through 2.2K ohm resistors.

The reset signal to the 8086 can be generated by the 8284. The 8284 has a schmitt trigger input ($\overline{RES}$) for generating reset from a low active external reset. The hysteresis specified in the 8284 data sheet implies that at least .25 volts will separate the 0 and 1 switching point of the 8284 reset input. Inputs without hysteresis will switch from low to high and high to low at approximately the same voltage threshold. The inputs are guaranteed to switch at specified low and high voltages (VIL and VIH) but the actual switching point is anywhere in-between. Since VIL min is specified at .8 volts, the hysteresis guarantees that the reset will be active until the input reaches at least 1.05 volts. A reset will not be recognized until the input drops at least .25 volts below the reset inputs VIH of 2.6 volts.

To guarantee reset from power up, the reset input must remain below 1.05 volts for 50 microseconds after V_{CC} has reached the minimum supply voltage of 4.5 volts. The hysteresis allows the reset input to be driven by a simple RC circuit as shown in Figure 3C4. The calculated RC value does not include time for the power supply to reach 4.5 volts or the charge accumulated during this interval. Without the hysteresis, the reset output might oscillate as the input voltage passes through the switching voltage of the input. The calculated RC value provides the minimum required reset period of 50 microseconds for 8284's that switch at the 1.05 volt level and a reset period of approximately 162 microseconds for 8284's that switch at the 2.6 volt level. If tighter tolerance between the minimum and maximum reset times is necessary, the reset circuit shown in Figure 3C5 might be used rather than the simple RC circuit. This circuit provides a constant current source and a linear charge rate on the capacitor rather than the inverse exponential charge rate of the RC circuit. The maximum reset period for this implementation is 124 microseconds.

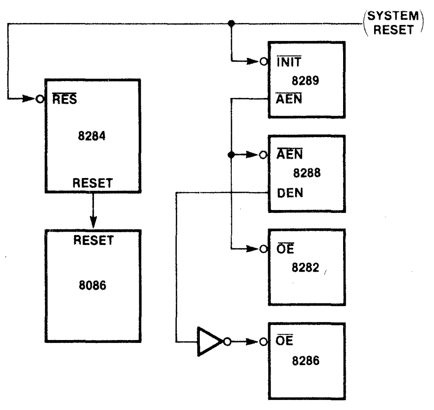


Figure 3C3. Reset Disable of for Max Mode 8086 Bus Interface in Multi CPU System

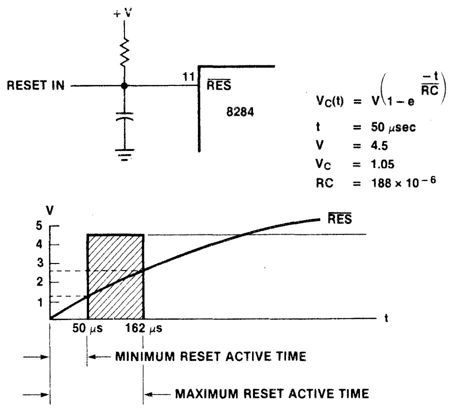


Figure 3C4. 8284 Reset Circuit

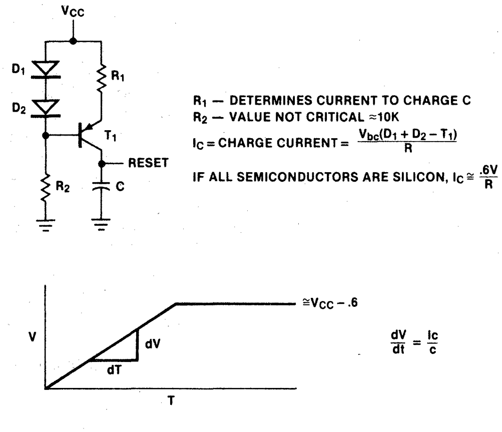


Figure 3C5. Constant Current Power-On Reset Circuit

The 8284 synchronizes the reset input with the CPU clock to generate the RESET signal to the CPU (Fig. 3C6). The output is also available as a general reset to the entire system. The reset has no effect on any clock circuits in the 8284.

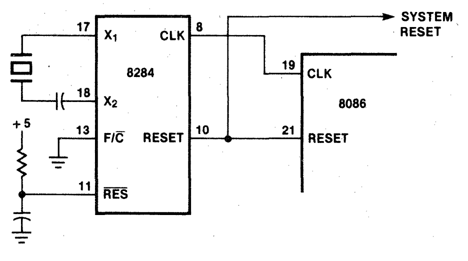


Figure 3C6. 8086 Reset and System Reset

3D. Ready Implementation and Timing

As discussed previously, the ready signal is used in the system to accommodate memory and I/O devices that cannot transfer information at the maximum CPU bus bandwidth. Ready is also used in multiprocessor systems to force the CPU to wait for access to the system bus or Multibus system bus. To insert a wait state in the bus cycle, the READY signal to the CPU must be inactive (low) by the end of T2. To avoid insertion of a wait state, READY must be active (high) within a specified setup time prior to the positive transition during T3. Depending on the size and characteristics of the system, ready implementation may take one of two approaches.

The classical ready implementation is to have the system 'normally not ready.' When the selected device receives the command ($\overline{RD}$/$\overline{WR}$/$\overline{INTA}$) and has had sufficient time to complete the command, it activates READY to the CPU, allowing the CPU to terminate the bus cycle. This implementation is characteristic of large multiprocessor, Multibus systems or systems where propagation delays, bus access delays and device characteristics inherently slow down the system. For maximum system performance, devices that can run with no wait states must return 'READY' within the previously described limit. Failure to respond in time will only result in the insertion of one or more wait cycles.

An alternate technique is to have the system 'normally ready.' All devices are assumed to operate at the maximum CPU bus bandwidth. Devices that do not meet the requirement must disable READY by the end of T2 to guarantee the insertion of wait cycles. This implementation is typically applied to small single CPU systems and reduces the logic required to control the ready signal. Since the failure of a device requiring wait states to disable READY by the end of T2 will result in premature termination of the bus cycle, the system timing must be carefully analyzed when using this approach.

The 8086 has two different timing requirements on READY depending on the system implementation. For a 'normally ready' system to insert a wait state, the READY must be disabled within 8 ns (TRYLCL) after the end of T2 (start of T3) (Diag. 3D1). To guarantee proper

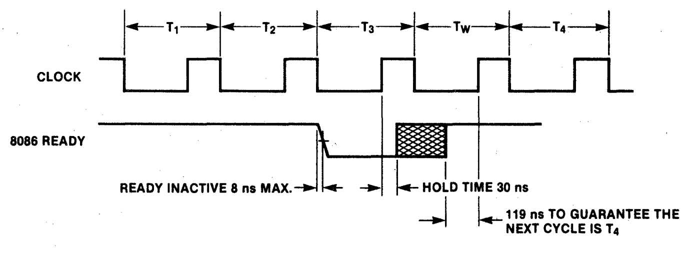


Diagram 3D1. Normally Ready System Inserting a Wait State operation of the 8086, the READY input must not change from ready to not ready during the clock low time of T3. For a 'normally not ready' system to avoid wait states, READY must be active within 119 ns (TRYHCH) of the positive clock transition during T3 (Diag. 3D2). For both cases, READY must satisfy a hold time of 30 ns (TCHRYX) from the T3 or TW positive clock transition.

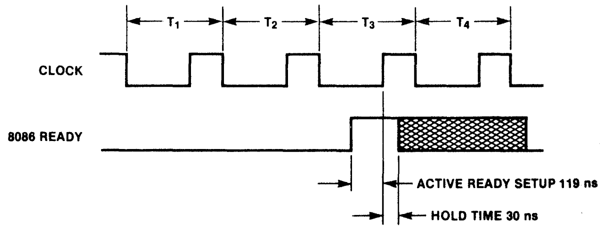


Diagram 3D2. Normally Not Ready System Avoiding a Wait State

To generate a stable READY signal which satisfies the previous setup and hold times, the 8284 provides two separate system ready inputs (RDY1, RDY2) and a single synchronized ready output (READY) for the CPU. The RDY inputs are qualified with separate access enables ($\overline{AEN1}$,$\overline{AEN2}$, low active) to allow selecting one of the two ready signals (Fig. 3D1). The gated signals are logically OR'ed and sampled at the beginning of each CLK cycle to generate READY to the CPU (Diag. 3D3). The sampled READY signal is valid within 8 ns (TRYLCL) after CLK to satisfy the CPU timing requirements on 'not ready' and ready. Since READY cannot change until the next CLK, the hold time requirements are also satisfied. The system ready inputs to the 8284 (RDY1,RDY2) must be valid 35 ns (TRIVCL) before T3 and $\overline{AEN}$ must be valid 60 ns before T3. For a system using only one RDY input, the associated $\overline{AEN}$ is tied to ground while the other $\overline{AEN}$ is connected to 5 volts through ~1K ohms (Fig. 3D2a). If the system generates a low active ready signal, it can be connected to the 8284 $\overline{AEN}$ input if the additional setup time required by the 8284 $\overline{AEN}$ input is satisfied. In this case, the associated RDY input would be tied high (Fig. 3D2b).

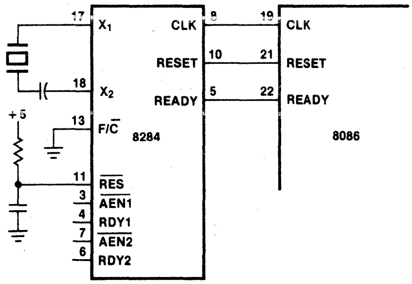
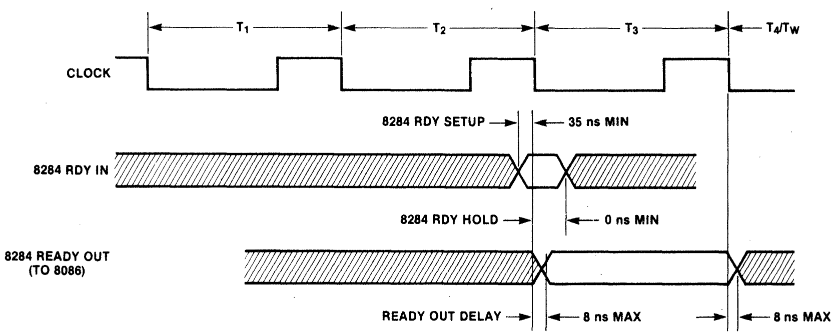


Figure 3D1. Ready Inputs to the 8284 and Output to the 8086



NOTE: THE 8284 DATA SHEET SPECIFIES READY OUT DELAY (TRYLCL) AS −8 ns 'BEFORE' THE END OF T_2 WHICH IMPLIES THE TIMING SHOWN.

Diagram 3D3. 8284 with 8086 Ready Timing

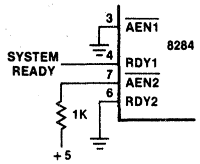


Figure 3D2a. Using RDY1/RDY2 to Generate Ready

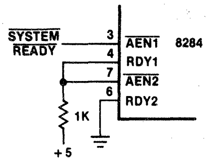


Figure 3D2b. Using AEN1/AEN2 to Generate Ready

The majority of memory and peripheral devices which fail to operate at the maximum CPU frequency typically do not require more than one wait state. The circuit given in Figure 3D3 is an example of a simple wait state generator. The system ready line is driven low whenever a device requiring one wait state is selected. The flip flop is cleared by ALE, enabling RDY to the 8284. If no wait states are required, the flip flop does not change. If the system ready is driven low, the flip flop toggles on the low to high clock transition of T2 to force one wait state. The next low to high clock transition toggles the flip flop again to indicate ready and allow completion of the bus cycle. Further changes in the state of the flip flop will not affect the bus cycle. The circuit allows approximately 100 ns for chip select decode and conditioning of the system ready (Diag. 3D4).

If the system is 'normally not ready,' the programmer should not assign executable code to the last six bytes of physical memory. Since the 8086 prefetches instructions, the CPU may attempt to access non-existent memory when executing code at the end of physical memory. If the access to non-existent memory fails to enable READY, the system will be caught in an indefinite wait.

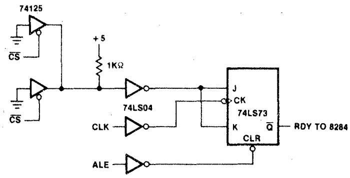


Figure 3D3. Single Wait State Generator

3E. Interrupt Structure

The 8086 interrupt structure is based on a table of interrupt vectors stored in memory locations 0H through 003FFH. Each vector consists of two bytes for the instruction pointer and two bytes for the code segment. These two values combine to form the address of the interrupt service routine. This allows the table to contain up to 256 interrupt vectors which specify the starting address of the service routines anywhere in the one megabyte address space of the 8086. If fewer than 256 different interrupts are defined in the system, the user need only allocate enough memory for the interrupt vector table to provide the vectors for the defined interrupts. During initial system debug, however, it may be desirable to assign all undefined interrupt types to a trap routine to detect erroneous interrupts.

Each vector is associated with an interrupt type number which points to the vector's location in the interrupt vector table. The interrupt type number multiplied by four gives the displacement of the first byte of the associated interrupt vector from the beginning of the table. As an example, interrupt type number 5 points to the sixth entry in the interrupt vector table. The contents of this entry in the table points to the interrupt service routine for type 5 (Fig. 3E1). This structure allows the user to specify the memory address of each service routine by placing the address (instruction pointer and code segment values) in the table location provided for that type interrupt.

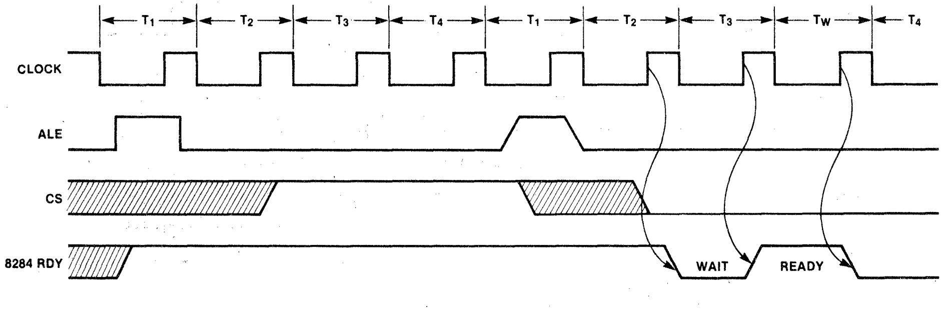


Diagram 3D4.

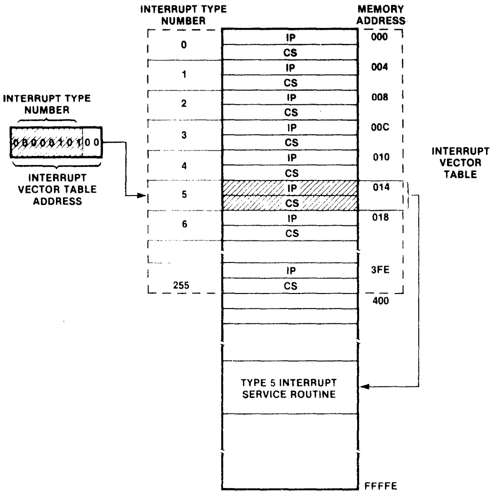


Figure 3E1. Direction to Interrupt Service Routine through the Interrupt Vector Table

All interrupts in the 8086 must be assigned an interrupt type which uniquely identifies each interrupt. There are three classes of interrupt types in the 8086; predefined interrupt types which are issued by specific functions within the 8086 and user defined hardware and software interrupts. Note that any interrupt type including the predefined interrupts can be issued by the user's hardware and/or software.

PREDEFINED INTERRUPTS

The predefined interrupt types in the 8086 are listed below with a brief description of how each is invoked. When invoked, the CPU will transfer control to the memory location specified by the vector associated with the specific type. The user must provide the interrupt service routine and initialize the interrupt vector table with the appropriate service routine address. The user may additionally invoke these interrupts through hardware or software. If the preassigned function is not used in the system, the user may assign some other function to the associated type. However, for compatibility with future Intel hardware and software products for the 8086 family, interrupt types 0-31 should not be assigned as user defined interrupts.

TYPE 0 — DIVIDE ERROR

This interrupt type is invoked whenever a division operation is attempted during which the quotient exceeds the maximum value (ex. division by zero). The interrupt is non-maskable and is entered as part of the execution of the divide instruction. If interrupts are not reenabled by the divide error interrupt service routine, the service routine execution time should be included in the worst case divide instruction execution time (primarily when considering the longest instruction execution time and its effect on latency to servicing hardware interrupts).

TYPE 1 — SINGLE STEP

This interrupt type occurs one instruction after the TF (Trap Flag) is set in the flag register. It is used to allow software single stepping through a sequence of code. Single stepping is initiated by copying the flags onto the stack, setting the TF bit on the stack and popping the flags. The interrupt routine should be the single step routine. The interrupt sequence saves the flags and program counter, then resets the TF flag to allow the single step routine to execute normally. To return to the routine under test, an interrupt return restores the IP, CS and flags with TF set. This allows the execution of the next instruction in the program under test before trapping back to the single step routine. Single Step is not masked by the IF (Interrupt Flag) bit in the flag register.

TYPE 2 — NMI (Non-Maskable Interrupt)

This is the highest priority hardware interrupt and is non-maskable. The input is edge triggered but is synchronized with the CPU clock and must be active for two clock cycles to guarantee recognition. The interrupt signal may be removed prior to entry to the service routine. Since the input must make a low to high transition to generate an interrupt, spurious transitions on the input should be suppressed. If the input is normally high, the NMI low time to guarantee triggering is two CPU clock times. This input is typically reserved for catastrophic failures like power failure or timeout of a system watchdog timer.

TYPE 3 — ONE BYTE INTERRUPT

This is invoked by a special form of the software interrupt instruction which requires a single byte of code space. Its primary use is as a breakpoint interrupt for software debug. With full representation within a single byte, the instruction can map into the smallest instruction for absolute resolution in setting breakpoints. The interrupt is not maskable.

TYPE 4 — INTERRUPT ON OVERFLOW

This interrupt occurs if the overflow flag (OF) is set in the flag register and the INTO instruction is executed. The instruction allows trapping to an overflow error service routine. The interrupt is non-maskable.

Interrupt types 0 and 2 can occur without specific action by the programmer (except for performing a divide for Type 0) while types 1, 3, and 4 require a conscious act by the programmer to generate these interrupt types. All but type 2 are invoked through software activity and are directly associated with a specific instruction.

USER DEFINED SOFTWARE INTERRUPTS

The user can generate an interrupt through the software with a two byte interrupt instruction INT nn. The first byte is the INT opcode while the second byte (nn) contains the type number of the interrupt to be performed. The INT instruction is not maskable by the interrupt enable flag. This instruction can be used to transfer control to routines that are dynamically relocatable and whose location in memory is not known by the calling program. This technique also saves the flags of the calling program on the stack prior to transferring control. The called procedure must return control with an interrupt return (IRET) instruction to remove the flags from the stack and fully restore the state of the calling program.

All interrupts invoked through software (all interrupts discussed thus far with the exception of NMI) are not maskable with the IF flag and initiate the transfer of control at the end of the instruction in which they occur. They do not initiate interrupt acknowledge bus cycles and will disable subsequent maskable interrupts by resetting the IF and TF flags. The interrupt vector for these interrupt types is either implied or specified in the instruction. Since the NMI is an asynchronous event to the CPU, the point of recognition and initiation of the transfer of control is similar to the maskable hardware interrupts.

USER DEFINED HARDWARE INTERRUPTS

The maskable interrupts initiated by the system hardware are activated through the INTR pin of the 8086 and are masked by the IF bit of the status register (interrupt flag). During the last clock cycle of each instruction, the state of the INTR pin is sampled. The 8086 deviates from this rule when the instruction is a MOV or POP to a segment register. For this case, the interrupts are not sampled until completion of the following instruction. This allows a 32-bit pointer to be loaded to the stack pointer registers SS and SP without the danger of an interrupt occurring between the two loads. Another exception is the WAIT instruction which waits for a low active input on the $\overline{TEST}$ pin. This instruction also continuously samples the interrupt request during its execution and allows servicing interrupts during the wait. When an interrupt is detected, the WAIT instruction is again fetched prior to servicing the interrupt to guarantee the interrupt routine will return to the WAIT instruction.

UNINTERRUPTABLE INSTRUCTION SEQUENCE

```
MOV SS, NEW$STACK$SEGMENT
MOV SP, NEW$STACK$POINTER
```

Also, since prefixes are considered part of the instruction they precede, the 8086 will not sample the interrupt line until completion of the instruction the prefix(es) precede(s). An exception to this (other than HALT or WAIT) is the string primatives preceded by the repeat (REP) prefix. The repeated string operations will sample the interrupt line at the completion of each repetition. This includes repeat string operations which include the lock prefix. If multiple prefixes precede a repeated string operation, and the instruction is interrupted, only the prefix immediately preceding the string primative is restored. To allow correct resumption of the operation, the following programming technique may be used:

```
LOCKED$BLOCK$MOVE:  LOCK REP MOVS DEST, CS:SOURCE
                    AND CX,    CX
                    JNZ LOCKED$BLOCK$MOVE
```

The code bytes generated by the 8086 assembler for the MOVS instruction are (in descending order): LOCK prefix, REP prefix, Segment Override prefix and MOVS. Upon return from the interrupt, the segment override prefix is restored to guarantee one additional transfer is performed between the correct memory locations. The instructions following the move operation test the repetition count value to determine if the move was completed and return if not.

If the INTR pin is high when sampled and the IF bit is set to enable interrupts, the 8086 executes an interrupt acknowledge sequence. To guarantee the interrupt will be acknowledged, the INTR input must be held active until the interrupt acknowledge is issued by the CPU. If the BIU is running a bus cycle when the interrupt condition is detected (as would occur if the BIU is fetching an instruction when the current instruction completes), the

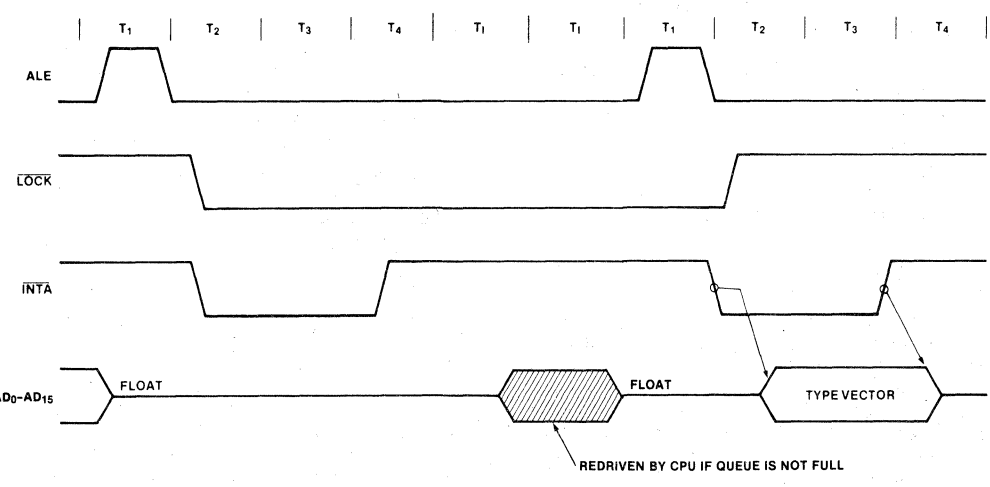


Figure 3E2. Interrupt Acknowledge Sequence interrupt must be valid at the 8086 2 clock cycles prior to T4 of the bus cycle if the next cycle is to be an interrupt acknowledge cycle. If the 2 clock setup is not satisfied, another pending bus cycle will be executed before the interrupt acknowledge is issued. If a hold request is also pending (this might occur if an interrupt and hold request are made during execution of a locked instruction), the interrupt is serviced after the hold request is serviced.

The interrupt acknowledge sequence is only generated in response to an interrupt on the 8086 INTR input. The associated bus activity is shown in Figure 3E2. The cycle consists of two $\overline{INTA}$ bus cycles separated by two idle clock cycles. During the bus cycles the $\overline{INTA}$ command is issued rather than read. No address is provided by the 8086 during either bus cycle ($\overline{BHE}$ and status are valid), however, ALE is still generated and will load the address latches with indeterminate information. This condition requires that devices in the system do not drive their outputs without being qualified by the Read Command. As will be shown later, the ALE is useful in maximum mode systems with multiple 8259A priority interrupt controllers. During the $\overline{INTA}$ bus cycles, DT/$\overline{R}$ and DEN are conditioned to allow the 8086 to receive a one byte interrupt type number from the interrupt system. The first $\overline{INTA}$ bus cycle signals an interrupt acknowledge cycle is in progress and allows the system to prepare to present the interrupt type number on the next $\overline{INTA}$ bus cycle. The CPU does not capture information on the bus during the first cycle. The type number must be transferred to the 8086 on the lower half of the 16-bit data bus during the second cycle. This implies that devices which present interrupt type numbers to the 8086 must be located on the lower half of the 16-bit data bus. The timing of the $\overline{INTA}$ bus cycles (with exception of address timing) is similar to read cycle timing. The 8086 interrupt acknowledge sequence deviates from the form used on 8080 and 8085 in that no instruction is issued as part of the sequence. The 8080 and 8085 required either a restart or call instruction be issued to affect the transfer of control.

In the minimum mode system, the M/$\overline{IO}$ signal will be low indicating I/O during the $\overline{INTA}$ bus cycles. The 8086 internal $\overline{LOCK}$ signal will be active from T2 of the first bus cycle until T2 of the second to prevent the BIU from honoring a hold request between the two $\overline{INTA}$ cycles.

In the maximum mode, the status lines $\overline{S0}$-$\overline{S2}$ will request the 8288 to activate the $\overline{INTA}$ output for each cycle. The $\overline{LOCK}$ output of the 8086 will be active from T2 of the first cycle until T2 of the second to prevent the 8086 from honoring a hold request on either $\overline{RQ}/\overline{GT}$ input and to prevent bus arbitration logic from relinquishing the bus between $\overline{INTA}$'s in multi-master systems. The consequences of READY are identical to those for READ and WRITE cycles.

Once the 8086 has the interrupt type number (from the bus for hardware interrupts, from the instruction stream for software interrupts or from the predefined condition), the type number is multiplied by four to form the displacement to the corresponding interrupt vector in the interrupt vector table. The four bytes of the interrupt vector are: least significant byte of the instruction pointer, most significant byte of the instruction pointer, least significant byte of the code segment register, most significant byte of the code segment register. During the transfer of control, the CPU pushes the flags and current code segment register and instruction pointer onto the stack. The new code segment and instruction pointer values are loaded and the single step and interrupt flags are reset. Resetting the interrupt flag disables response to further hardware interrupts in the service routine unless the flags are specifically re-enabled by the service routine. The CS and IP values are read from the interrupt vector table with data read cycles. No segment registers are used when referencing the vector table during the interrupt context switch. The vector displacement is added to zero to form the 20-bit address and S4, S3 = 10 indicating no segment register selection.

The actual bus activity associated with the hardware interrupt acknowledge sequence is as follows: Two interrupt acknowledge bus cycles, read new IP from the interrupt vector table, read new CS from the interrupt vector table, Push flags, Push old CS, Opcode fetch of the first instruction of the interrupt service routine, and Push old IP. After saving the old IP, the BIU will resume normal operation of prefetching instructions into the queue and servicing EU requests for operands. S5 (interrupt enable flag status) will go inactive in the second clock cycle following reading the new CS.

The number of clock cycles from the end of the instruction during which the interrupt occurred to the start of interrupt routine execution is 61 clock cycles. For software generated interrupts, the sequence of bus cycles is the same except no interrupt acknowledge bus cycles are executed. This reduces the delay to service routine execution to 51 clocks for INT nn and single step, 52 clocks for INT3 and 53 clocks for INTO. The same interrupt setup requirements with respect to the BIU that were stated for the hardware interrupts also apply to the software interrupts. If wait states are inserted by either the memories or the device supplying the interrupt type number, the given clock times will increase accordingly.

When considering the precedence of interrupts for multiple simultaneous interrupts, the following guidelines apply: 1. INTR is the only maskable interrupt and if detected simultaneously with other interrupts, resetting of IF by the other interrupts will mask INTR. This causes INTR to be the lowest priority interrupt serviced after all other interrupts unless the other interrupt service routines reenable interrupts. 2. Of the nonmaskable interrupts (NMI, Single Step and software generated), in general, Single Step has highest priority (will be serviced first) followed by NMI, followed by the software interrupts. This implies that a simultaneous NMI and Single Step trap will cause the NMI service routine to follow single step; a simultaneous software trap and Single Step trap will cause the software interrupt service routine to follow single step and a simultaneous NMI and software trap will cause the NMI service routine to be executed followed by the software interrupt service routine. An exception to this priority structure occurs if all three interrupts are pending. For this case, transfer of control to the software interrupt service routine followed by the NMI trap will cause both the NMI and software interrupt service routines to be executed without single stepping. Single stepping resumes upon execution of the instruction following the instruction causing the software interrupt (the next instruction in the routine being single stepped).

If the user does not wish to single step before INTR service routines, the single step routine need only disable interrupts during execution of the program being single stepped and reenable interrupts on entry to the single step routine. Disabling the interrupts during the program under test prevents entry into the interrupt service routine while single step (TF = 1) is active. To prevent single stepping before NMI service routines, the single step routine must check the return address on the stack for the NMI service routine address and return control to that routine without single step enabled. As examples, consider Figures 3E3a and 3E3b. In 3E3a Single Step and NMI occur simultaneously while in 3E3b, NMI, INTR and a divide error all occur during a divide instruction being single stepped.

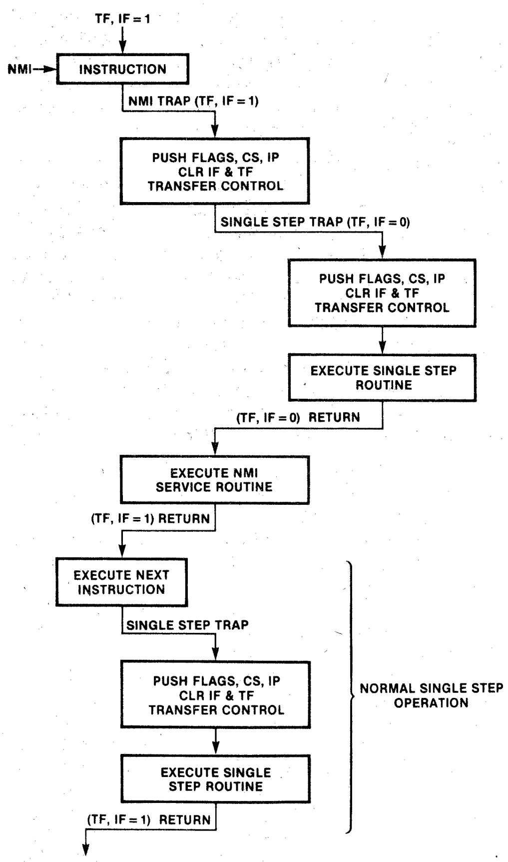


Figure 3E3a. NMI During Single Stepping and Normal Single Step Operation

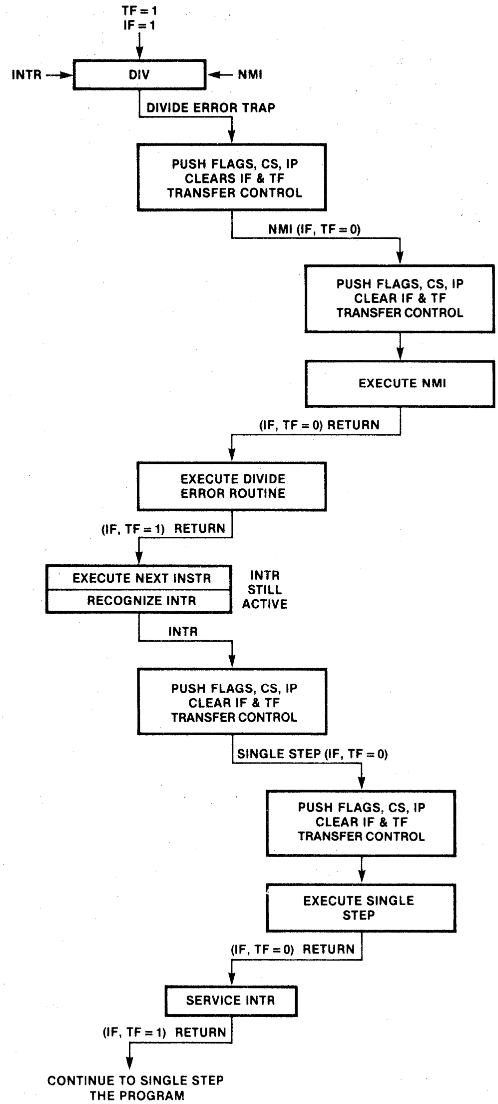


Figure 3E3b. NMI, INTR, Single Step and Divide Error Simultaneous Interrupts

SYSTEM CONFIGURATIONS

To accommodate the $\overline{\text{INTA}}$ protocol of the maskable hardware interrupts, the 8259A is provided as part of the 8086 family. This component is programmable to operate in both 8080/8085 systems and 8086 systems. The devices are cascadable in master/slave arrangements to allow up to 64 interrupts in the system. Figures 3E4 and 3E5 are examples of 8259A's in minimum and maximum mode 8086 systems. The minimum mode configuration (a) shows an 8259A connected to the CPU's multiplexed bus. Configuration (b) illustrates an 8259A connected to a demultiplexed bus system. These interconnects are also applicable to maximum mode systems. The configuration given for a maximum mode system shows a master 8259A on the CPU's multiplexed bus with additional slave 8259A's out on the buffered system bus. This configuration demonstrates several unique features of the maximum mode system interface. If the master 8259A receives interrupts from a mix of slave 8259A's and regular interrupting devices, the slaves must provide the type number for devices connected to them while the master provides the type number for devices directly attached to its interrupt inputs. The master 8259A is programmable to determine if an interrupt is from a direct input or a slave 8259A and will use this information to enable or disable the data bus transceivers (via the 'nand' function of DEN and $\overline{EN}$). If the master must provide the type number, it will disable the data bus transceivers. If the slave provides the type number, the master will enable the data bus transceivers. The $\overline{EN}$ output is normally high to allow the 8086/8288 to control the bus transceivers. To select the proper slave when servicing a slave interrupt, the master must provide a cascade address to the slave. If the 8288 is not strapped in the I/O bus mode (the 8288 IOB input connected to ground), the MCE/$\overline{PDEN}$ output becomes a MCE or Master Cascade Enable output. This signal is only active during $\overline{INTA}$ cycles as shown in Figure 3E6 and enables the master 8259A's cascade address onto the 8086's local bus during ALE. This allows the address latches to capture the cascade address with ALE and allows use of the system address bus for selecting the proper slave 8259A. The MCE is gated with $\overline{LOCK}$ to minimize local bus contention between the 8086 three-stating its bus outputs and the cascade address being enabled onto the bus. The first $\overline{INTA}$ bus cycle allows the master to resolve internal priorities and output a cascade address to be transmitted to the slaves on the subsequent $\overline{INTA}$ bus cycle. For additional information on the 8259A, reference application note AP-59.

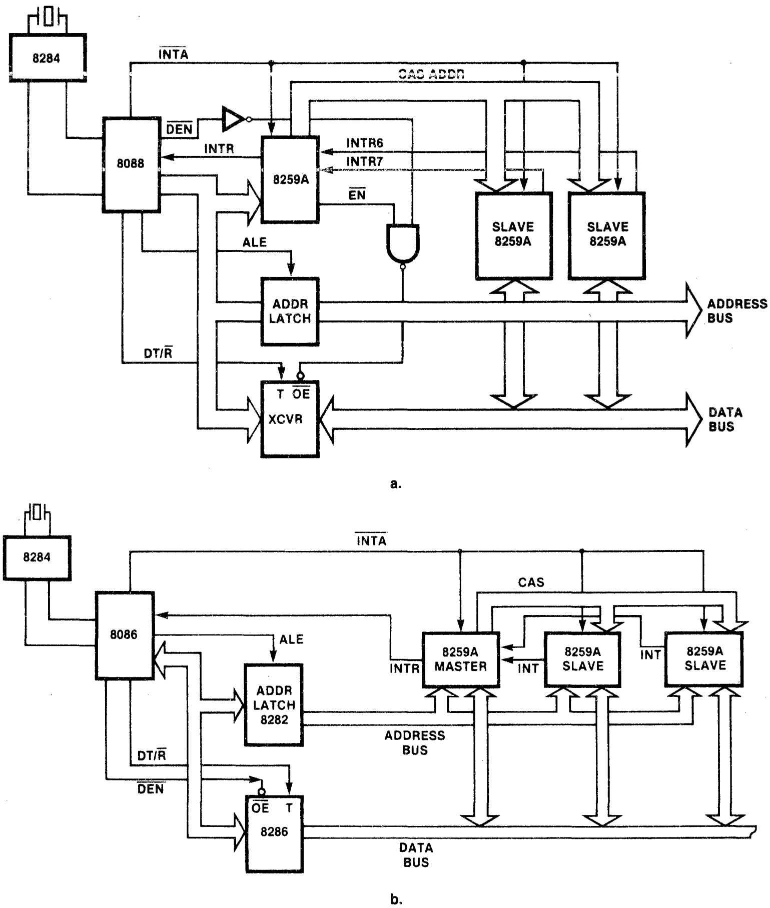


Figure 3E4. Min Mode 8086 with Master 8259A on the Local Bus and Slave 8259As on the System Bus

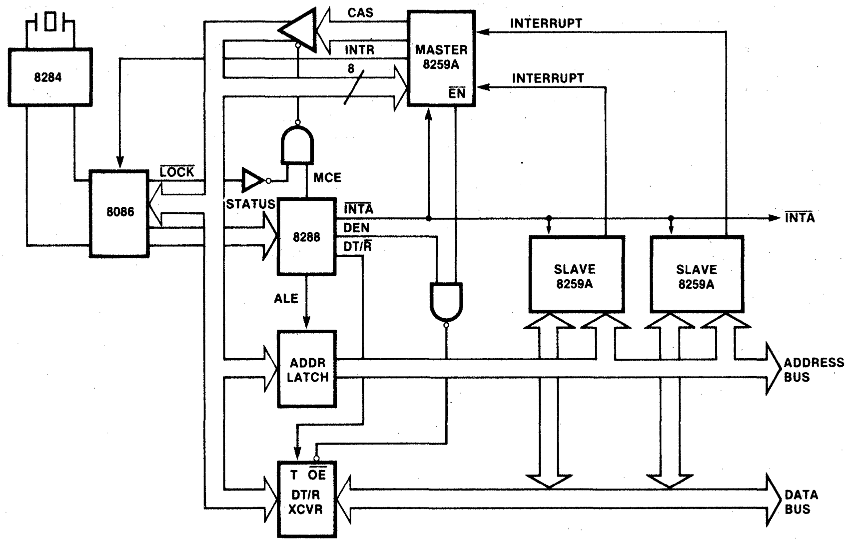


Figure 3E5. Max Mode 8086 with Master 8259A on the Local Bus and Slave 8259As on the System Bus

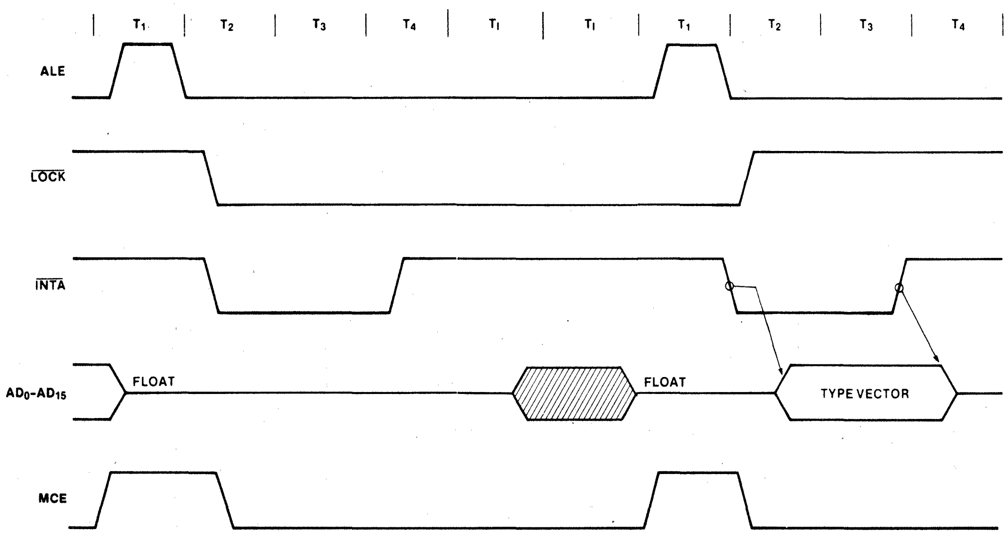


Figure 3E6. MCE Timing to Gate 8259A CAS Address onto the 8086 Local Bus

3F. Interpreting the 8086 Bus Timing Diagrams

At first glance, the 8086 bus timing diagrams (Diag. 3F1 min mode and Diag. 3F2 max mode) appear rather complex. However, with a few words of explanation on how to interpret them, they become a powerful tool in determining system requirements. The timing diagrams for both the minimum and maximum modes may be divided into six sections: (1) address and ALE timing; (2) read cycle timing; (3) write cycle timing; (4) interrupt acknowledge timing; (5) ready timing; and (6) HOLD/HLDA or $\overline{RQ}/\overline{GT}$ timing. Since the A.C. characteristics of the signals are specified relative to the CPU clock, the relationship between the majority of signals can be deduced by simply determining the clock cycles between the clock edges the signals are relative to and adding or subtracting the appropriate minimum or maximum parameter values. One aspect of system timing not compensated for in this approach is the worst case relationship between minimum and maximum parameter values (also known as tracking relationships). As an example, consider a signal which has specified minimum and maximum turn on and turn off delays. Depending on device characteristics, it may not be possible for the component to simultaneously demonstrate a maximum turn-on and minimum turn-off delay even though worst case analysis might imply the possibility. This argument is characteristic of MOS devices and is therefore applicable to the 8086 A.C. characteristics. The message is: worst case analysis mixing minimum and maximum delay parameters will typically exceed the worst case obtainable and therefore should not be subjected to further subjective degradation to obtain worst-worst case values. This section will provide guidelines for specific areas of 8086 timing sensitive to tracking relationships.

A. MINIMUM MODE BUS TIMING

1. ADDRESS and ALE

The address/ALE timing relationship is important to determine the ability to capture a valid address from the multiplexed bus. Since the 8282 and 8283 latches capture the address on the trailing edge of ALE, the critical timing involves the state of the address lines when ALE terminates. If the address valid delay is assumed to be maximum TCLAV and ALE terminates at its earliest point, TCHLLmin (assuming zero minimum delay), the address would be valid only TCLCHmin-TCLAVmax = 8 ns prior to ALE termination. This result is unrealistic in the assumption of maximum TCLAV and minimum TCHLL. To provide an accurate measure of the true worst case, a separate parameter specifies the minimum time for address valid prior to the end of ALE (TAVAL). TAVAL = TCLCH-60 ns overrides the clock related timings and guarantees 58 ns of address setup to ALE termination for a 5 MHz 8086. The address is guaranteed to remain valid beyond the end of ALE by the TLLAX parameter. This specification overrides the relationship between TCHLL and TCLAX which might seem to imply the address may not be valid by the end of the latest possible ALE. TLLAX holds for the entire address bus. The TCLAXmin spec on the address indicates the earliest the bus will go invalid if not restrained by a slow ALE. TLLAX and TCLAX apply to the entire multiplexed bus for both read and write cycles. AD15-0 is three-stated for read cycles and immediately switched to write data during write cycles. AD19-16 immediately switch from address to status for both read and write cycles. The minimum ALE pulse width is guaranteed by TLHLLmin which takes precedence over the value obtained by relating TCLLHmax and TCHLLmin.

To determine the worst case delay to valid address on a demultiplexed address bus, two paths must be considered: (1) delay of valid address and (2) delay to ALE. Since the 8282 and 8283 are flow through latches, a valid address is not transmitted to the address bus until ALE is active. A comparison of address valid delay TCLAVmax with ALE active delay TCLLHmax indicates TCLAVmax is the worst case. Subtracting the latch propagation delay gives the worst case address bus valid delay from the start of the bus cycle.

2. Read Cycle Timing

Read timing consists of conditioning the bus, activating the read command and establishing the data transceiver enable and direction controls. DT/$\overline{R}$ is established early in the bus cycle and requires no further consideration. During read, the $\overline{DEN}$ signal must allow the transceivers to propagate data to the CPU with the appropriate data setup time and continue to do so until the required data hold time. The $\overline{DEN}$ turn on delay allows TCLCL + TCHCLmin − TCVCTVmax − TDVCL = 127 ns transceiver enable time prior to valid data required by the CPU. Since the CPU data hold time TCLDXmin and minimum $\overline{DEN}$ turnoff delay TCVCTXmin are both 10 ns relative to the same clock edge, the hold time is guaranteed. Additionally, $\overline{DEN}$ must disable the transceivers prior to the CPU redriving the bus with the address for the next bus cycle. The maximum $\overline{DEN}$ turn off delay (TCVCTXmax) compared with the minimum delay for addresses out of the 8086 (TCLCL + TCLAVmin) indicates the transceivers are disabled at least 105 ns before the CPU drives the address onto the multiplexed bus.

If memory or I/O devices are connected directly to the multiplexed address and data bus, the TAZRL parameter guarantees the CPU will float the bus before activating read and allowing the selected device to drive the bus. At the end of the bus cycle, the TRHAV parameter specifies the bus float delay the device being deselected must satisfy to avoid contention with the CPU driving the address for the next bus cycle. The next bus cycle may start as soon as the cycle following T4 or any number of clock cycles later.

The minimum delay from read active to valid data at the CPU is 2TCLCL − TCLRLmax − TDVCL = 205 ns. The minimum pulse width is 2TCLCL − 75 ns = 325 ns. This specification (TRLRH) overrides the result which could be derived from clock relative delays (2TCLCL − TCLRLmax + TCLRHmin).

3. Write Cycle Timing

The write cycle involves providing write data to the system, generating the write command and controlling data bus transceivers. The transceiver direction control signal DT/$\overline{R}$ is conditioned to transmit at the end of each read cycle and does not change during a write cycle.

This allows the transceiver enable signal DEN to be active early in the cycle (while addresses are valid) without corrupting the address on the multiplexed bus. The write data and write command are both enabled from the leading edge of T2. Comparing minimum WR active delay TCVCTVmin with the maximum write data delay TCLDV indicates that write data may be not valid until 100 ns after write is active. The devices in the system should capture data on the trailing edge of the write command rather than the leading edge to guarantee valid data. The data from the 8086 is valid a minimum of 2TCLCL − TCLDVmax + TCVCTXmin = 300 ns before the trailing edge of write. The minimum write pulse width is TWLWH = 2TCLCL − 60 ns = 340 ns. The CPU maintains valid write data TWHDX ns after write. The TWHDZ specification overrides the result derived by relating TCLCHmin and TCHDZmin which implies write data may only be valid 18 ns after WR. The 8086 floats the bus after write only if being forced off the bus by a HOLD or RQ input. Otherwise, the CPU simply switches the output drivers from data to address at the beginning of the next bus cycle. As with the read cycle, the next bus cycle may start in the clock cycle following T4 or any clock cycle later.

DEN is disabled a minimum of TCLCHmin + TCVCTXmin − TCVCTXmax = 18 ns after write to guarantee data hold time to the selected device. Since we are again evaluating a minimum TCVCTX with a maximum TCVCTX, the real minimum delay from the end of write to transceiver disable is approximately 60 ns.

4. Interrupt Acknowledge Timing

The interrupt acknowledge sequence consists of two interrupt acknowledge bus cycles as previously described. The detailed timing of each cycle is identical to the read cycle timing with two exceptions: command timing and address/data bus timing.

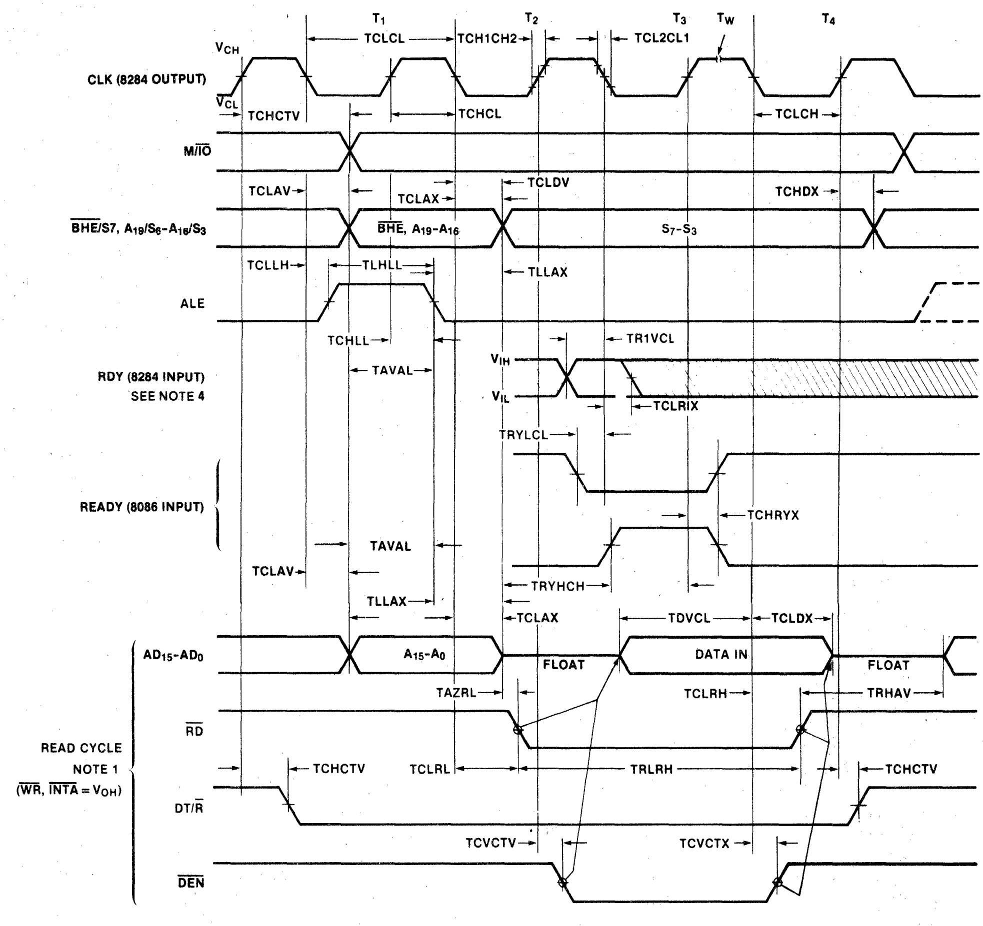
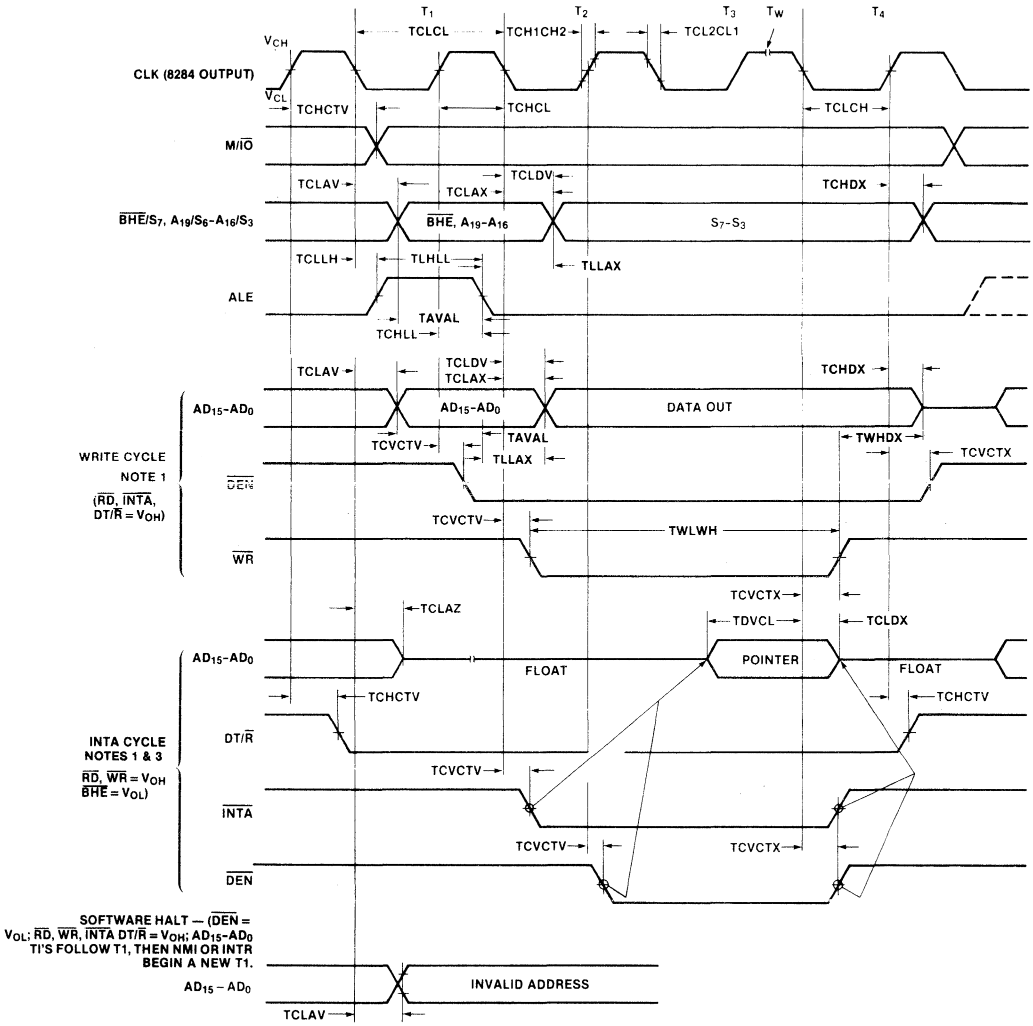


Figure 3F1. 8086 Bus Timing — Minimum Mode System



NOTES:
1. ALL SIGNALS SWITCH BETWEEN V_{OH} AND V_{OI} UNLESS OTHERWISE SPECIFIED.
2. RDY IS SAMPLED NEAR THE END OF T_2, T_3, T_W TO DETERMINE IF T_W MACHINES STATES ARE TO BE INSERTED.
3. BOTH INTA CYCLES RUN BACK-TO-BACK. THE 8088 LOCAL ADDR/DATA BUS IS FLOATING DURING THE SECOND INTA CYCLE. CONTROL SIGNALS SHOWN FOR SECOND INTA CYCLE.
4. SIGNALS AT 8284 ARE SHOWN FOR REFERENCE ONLY.
5. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS OTHERWISE NOTED.

Figure 3F1. 8086 Bus Timing — Minimum Mode System (Con't)

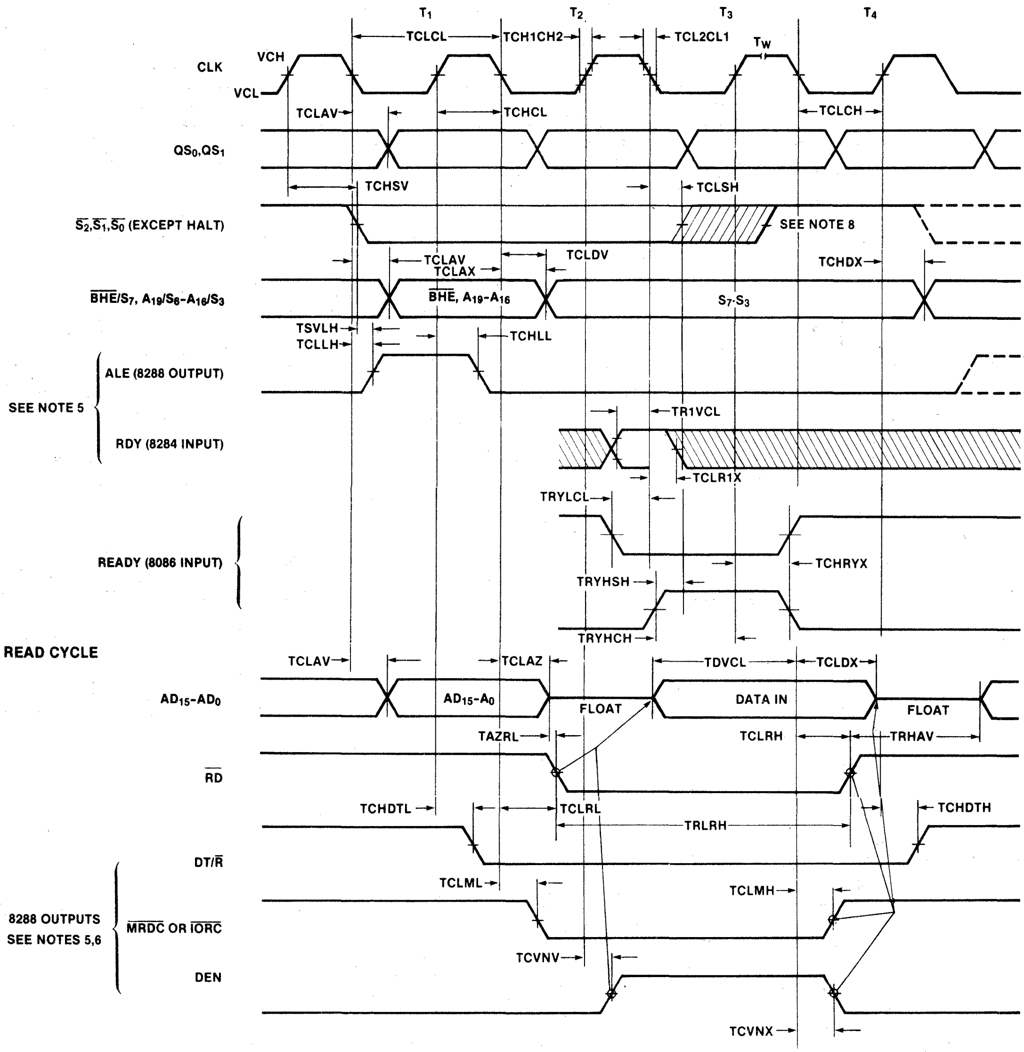
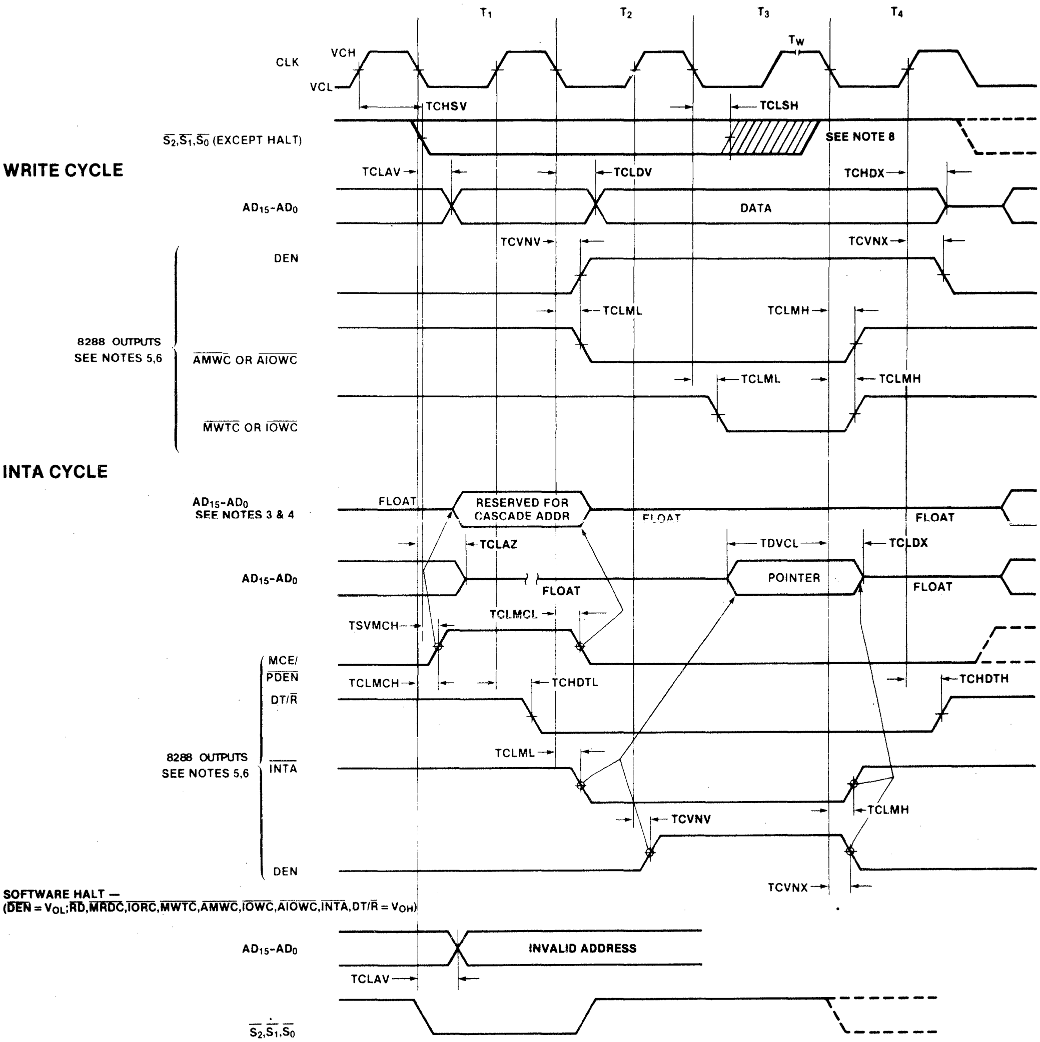


Figure 3F2a. 8086 Bus Timing — Maximum Mode System (Using 8288)



NOTES:
1. ALL SIGNALS SWITCH BETWEEN V_{OH} AND V_{OL} UNLESS OTHERWISE SPECIFIED.
2. RDY IS SAMPLED NEAR THE END OF T_2, T_3, T_W TO DETERMINE IF T_W MACHINES STATES ARE TO BE INSERTED.
3. CASCADE ADDRESS IS VALID BETWEEN FIRST AND SECOND INTA CYCLES.
4. BOTH INTA CYCLES RUN BACK-TO-BACK. THE 8086 LOCAL ADDR/DATA BUS IS FLOATING DURING THE SECOND INTA CYCLE. CONTROL FOR POINTER ADDRESS IS SHOWN FOR SECOND INTA CYCLE.
5. SIGNALS AT 8284 OR 8288 ARE SHOWN FOR REFERENCE ONLY.
6. THE ISSUANCE OF THE 8288 COMMAND AND CONTROL SIGNALS (MRDC, MWTC, AMWC, IORC, IOWC, AIOWC, INTA AND DEN) LAGS THE ACTIVE HIGH 8288 CEN.
7. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS OTHERWISE NOTED.
8. STATUS INACTIVE IN STATE JUST PRIOR TO T_4.

Figure 3F2b. 8086 Bus Timing — Maximum Mode System (Using 8288) (Con't)

The multiplexed address/data bus floats from the beginning (T1) of the $\overline{INTA}$ cycle (within TCLAZ ns). The upper four multiplexed address/status lines do not three-state. The address value on A19-A16 is indeterminate but the status information will be valid (S3 = 0, S4 = 0, S5 = IF, S6 = 0, S7 = $\overline{BHE}$ = 0). The multiplexed address/data lines will remain in three-state until the cycle after T4 of the $\overline{INTA}$ cycle. This sequence occurs for each of the $\overline{INTA}$ bus cycles. The interrupt type number read by the 8086 on the second $\overline{INTA}$ bus cycle must satisfy the same setup and hold times required for data during a read cycle.

The $\overline{DEN}$ and DT/$\overline{R}$ signals are enabled for each $\overline{INTA}$ cycle and do not remain active between the two cycles. Their timing for each cycle is identical to the read cycle.

The $\overline{INTA}$ command has the same timing as the write command. It is active within 110 ns of the start of T2 providing 260 ns of access time from command to data valid at the 8086. The command is active a minimum of TCVCTXmin = 10 ns into T4 to satisfy the data hold time of the 8086. This provides minimum $\overline{INTA}$ pulse width of 300 ns, however taking signal delay tracking into consideration gives a minimum pulse width of 340 ns. Since the maximum inactive delay of $\overline{INTA}$ is TCVCTXmax = 110 ns and the CPU will not drive the bus until 15 ns (TCLAVmin) into the next clock cycle, 105 ns are available for interrupt devices on the local bus to float their outputs. If the data bus is buffered, $\overline{DEN}$ provides the same amount of time for local bus transceivers to three-state their outputs.

5. Ready Timing

The detailed timing requirements of the 8086 ready signal and the system ready signal into the 8284 are described in Section 3D. The system ready signal is typically generated from either the address decode of the selected device or the address decode and the command ($\overline{RD}$, $\overline{WR}$, $\overline{INTA}$). For a system which is normally not ready, the time to generate ready from a valid address and not insert a wait state, is 2TCLCL − TCLAVmax − TR1VCLmax = 255 ns. This time is available for buffer delays and address decoding to determine if the selected device does not require a wait state and drive the RDY line high. If wait cycles are required, the user hardware must provide the appropriate ready delay. Since the address will not change until the next ALE, the RDY will remain valid throughout the cycle. If the system is normally ready, selected devices requiring wait states also have 255 ns to disable the RDY line. The user circuitry must delay re-enabling RDY by the appropriate number of wait states.

If the $\overline{RD}$ command is used to enable the RDY signal, TCLCL − TCLRLmax − TRIVCLmax = 15 ns are available for external logic. If the $\overline{WR}$ command is used, TCLCL − TCVCTVmax − TRIVCLmax = 55 ns are available. Comparison of RDY control by address or command indicates that address decoding provides the best timing. If the system is normally not ready, address decode alone could be used to provide RDY for devices not requiring wait states while devices requiring wait states may use a combination of address decode and command to activate a wait state generator. If the system is normally ready, devices not requiring wait states do nothing to RDY while devices needing wait states should disable RDY via the address decode and use a combination of address decode and command to activate a delay to re-enable RDY.

If the system requires no wait states for memory and a fixed number of wait states for $\overline{RD}$ and $\overline{WR}$ to all I/O devices, the M/$\overline{IO}$ signal can be used as an early indication of the need for wait cycles. This allows a common circuit to control ready timing for the entire system without feedback of address decodes.

6. Other Considerations

Detailed HOLD/HLDA timing is covered in the next section and is not examined here. One last signal consideration needs to be mentioned for the minimum mode system. The $\overline{TEST}$ input is sampled by the 8086 only during execution of the WAIT instruction. The $\overline{TEST}$ signal should be active for a minimum of 6 clock cycles during the WAIT instruction to guarantee detection.

B. MAXIMUM MODE BUS TIMING

The maximum mode 8086 bus operations are logically equivalent to the minimum mode operation. Detailed timing analysis now involves signals generated by the CPU and the 8288 bus controller. The 8288 also provides additional control and command signals which expand the flexibility of the system.

1. ADDRESS and ALE

In the maximum mode, the address information continues to come from the CPU while the ALE strobe is generated by the 8288. To determine the worst case relationships between ALE and the address, we first must determine 8288 ALE activation relative to the $\overline{S0}$-$\overline{S2}$ status from the CPU. The maximum mode timing diagram specifies two possible delay paths to generate ALE. The first is TCHSV + TSVLH measured from the rising edge of the clock cycle preceding T1. The second path is TCLLH measured from the start of T1. Since the 8288 initiates a bus cycle from the status lines leaving the passive state ($\overline{S0}$-$\overline{S2}$ = 1), if the 8086 is late in issuing the status (TCHSVmax) while the clock high time is a minimum (TCHCLmin), the status will not have changed by the start of T1 and ALE is issued TSVLH ns after the status changes. If the status changes prior to the beginning of T1, the 8288 will not issue the ALE until TCLLH ns after the start of T1. The resulting worst case delay to enable ALE (relative to the start of T1) is TCHSVmax + TSVLHmax − TCHCLmin = 58 ns. Note, when calculating signal relationships, be sure to use the proper maximum mode values rather than equivalent minimum mode values.

The trailing edge of ALE is triggered in the 8288 by the positive clock edge in T1 regardless of the delay to enable ALE. The resulting minimum ALE pulse width is TCLCHmax − 58 ns = 75 ns assuming TCHLL = 0. TCLCHmax must be used since TCHCLmin was assumed to derive the 58 ns ALE enable delay. The address is guaranteed to be valid TCLCHmin + TCHLLmin − TCLAVmax = 8 ns prior to the trailing edge of ALE to capture the address in the 8282 or 8283 latches. Again we have assumed a very conservative TCHLL = 0. Note, since the address and ALE are driven by separate devices, no tracking of A.C. characteristics can be assumed.

The address hold time to the latches is guaranteed by the address remaining valid until the end of T1 while ALE is disabled a maximum of 15 ns from the positive clock transition in T1 (TCHCLmin − TCHLLmax = 52 ns address hold time). The multiplexed bus transitions from address to status and write data or three-state (for read) are identical to the minimum mode timing. Also, since the address valid delay (TCLAV) remains the critical path in establishing a valid address, the address access times to valid data and ready are the same as the minimum mode system.

2. Read Cycle Timing

The maximum mode system offers read signals generated by both the 8086 and the 8288. The 8086 $\overline{RD}$ output signal timing is identical to the minimum mode system. Since the A.C. characteristics of the read commands generated by the 8288 are significantly better than the 8086 output, access to devices on the demultiplexed buffered system bus should use the 8288 commands. The 8086 $\overline{RD}$ signal is available for devices which reside directly on the multiplexed bus. The following evaluations for read, write and interrupt acknowledge only consider the 8288 command timing.

The 8288 provides separate memory and I/O read signals which conform to the same A.C. characteristics. The commands are issued TCLML ns after the start of T2 and terminate TCLMH ns after the start of T4. The minimum command length is 2TCLCL − TCLMLmax + TCLMLmin = 375 ns. The access time to valid data at the CPU is 2TCLCL − TCLMLmax − TDVCLmax = 335 ns. Since the 8288 was designed for systems with buffered data busses, the commands are enabled before the CPU has three-stated the multiplexed bus and should not be used with devices which reside directly on the multiplexed bus (to do so could result in bus contention during 8086 bus float and device turn-on).

The direction control for data bus transceivers is established in T1 while the transceivers are not enabled by DEN until the positive clock transition of T2. This provides TCLCH + TCVNVmin = 123 ns for 8086 bus float delay and TCHCLmin + TCLCL − TCVNVmax − TDVCLmax = 187 ns of transceiver active to data valid at the CPU. Since both DEN and command are valid a minimum of 10 ns into T4, the CPU data hold time TCLDX is guaranteed. A maximum DEN disable of 45 ns (TCVNX max) guarantees the transceivers are disabled by the start of the next 8086 bus cycle (215 ns minimum from the same clock edge). On the positive clock transition of T4, DT/$\overline{R}$ is returned to transmit in preparation for a possible write operation on the next bus cycle. Since the system memory and I/O devices reside on a buffered system bus, they must three-state their outputs before the device for the next bus cycle is selected (approximately 2TCLCL) or the transceivers drive write data onto the bus (approximately 2TCLCL).

3. Write Cycle Timing

In the maximum mode, the 8288 provides normal and advanced write commands for memory and I/O. The advanced write commands are active a full clock cycle ahead of the normal write commands and have timing identical to the read commands. The advanced write pulse width is 2TCLCL − TCLMLmax + TCLMHmin = 375 ns while the normal write pulse width is TCLCL − TCLMLmax + TCLMHmin = 175 ns. Write data setup time to the selected device is a function of either the data valid delay from the 8086 (TCLDV) or the transceiver enable delay TCVNV. The worst case delay to valid write data is TCLDV = 110 ns minus transceiver propagation delays. This implies the data may not be valid until 100 ns after the advanced write command but will be valid approximately TCLCL − TCLDVmax + TCLMLmin = 100 ns prior to the leading edge of the normal write command. Data will be valid 2TCLCL − TCLDVmax + TCLMHmin = 300 ns before the trailing edge of either write command. The data and command overlap for the advanced command is 300 ns while the overlap with the normal write command is 175 ns. The transceivers are disabled a minimum of TCLCHmin − TCLMHmax + TCVNXmin = 85 ns after the write command while the CPU provides valid data a minimum of TCLCHmin − TCLMHmax + TCHDZmin = 85 ns. This guarantees write data hold of 85 ns after the write command. The transceivers are disabled TCLCL − TCVNXmax + TCHDTLmin = 155 ns (assuming TCHDTL = 0) prior to transceiver direction change for a subsequent read cycle.

4. Interrupt Acknowledge Timing

The maximum mode $\overline{INTA}$ sequence is logically identical to the minimum mode sequence. The transceiver control (DEN and DT/$\overline{R}$) and $\overline{INTA}$ command timing of each interrupt acknowledge cycle is identical to the read cycle. As in the minimum mode system, the multiplexed address/data bus will float from the leading edge of T1 for each $\overline{INTA}$ bus cycle and not be driven by the CPU until after T4 of each $\overline{INTA}$ cycle. The setup and hold times on the vector number for the second cycle are the same as data setup and hold for the read. If the device providing the interrupt vector number is connected to the local bus, TCLCL − TCLAZmax + TCLMLmin = 130 ns are available from 8086 bus float to $\overline{INTA}$ command active. The selected device on the local bus must disable the system data bus transceivers since DEN is still generated by the 8288.

If the 8288 is not in the IOB (I/O Bus) mode, the 8288 MCE/$\overline{PDEN}$ output becomes the MCE output. This output is active during each $\overline{INTA}$ cycle and overlaps the ALE signal during T1. The MCE is available for gating cascade addresses from a master 8259A onto three of the upper AD15-AD8 lines and allowing ALE to latch the cascade address into the address latches. The address lines may then be used to provide CAS address selection to slave 8259A's located on the system bus (reference Figure 3E5). MCE is active within 15 ns of status or the start of T1 for each $\overline{INTA}$ cycle. MCE should not enable the CAS lines onto the multiplexed bus during the first cycle since the CPU does not guarantee to float the bus until 80 ns into the first $\overline{INTA}$ cycle. The first MCE can be inhibited by gating MCE with $\overline{LOCK}$. The 8086 $\overline{LOCK}$ output is activated during T2 of the first cycle and disabled during T2 of the second cycle. The overlap of $\overline{LOCK}$ with MCE allows the first MCE to be masked and the second MCE to gate the cascade address onto the local bus. Since the 8259A will not provide a cascade address until the second cycle, no information is lost. As with ALE, MCE is guaranteed valid within 58 ns of the start of T1 to allow 75 ns CAS address setup to the trailing edge of ALE. MCE remains active TCHCLmin − TCHLLmax + TCLMCLmin = 52 ns after ALE to provide data hold time to the latches.

If the 8288 is strapped in the IOB mode, the MCE output becomes $\overline{PDEN}$ and all I/O references are assumed to be devices on the local bus rather than the demultiplexed system bus. Since $\overline{INTA}$ cycles are considered I/O cycles, all interrupts are assumed to come from the local system and cascade addresses are not gated onto the system address bus. Additionally, the DEN signal is not enabled since no I/O transfers occur on the system bus. If the local I/O bus is also buffered by transceivers, the $\overline{PDEN}$ signal is used to enable those transceivers. $\overline{PDEN}$ A.C. characteristics are identical to DEN with $\overline{PDEN}$ enabled for I/O references and DEN enabled for instruction or memory data references.

5. Ready Timing

Ready timing based on address valid timing is the same for maximum and minimum mode systems. The delay from 8288 command valid to RDY valid at the 8284 is TCLCL − TCLMLmax − TRIVCLmin = 130 ns. This time is available for external circuits to determine the need to insert wait states and disable RDY or enable RDY to avoid wait states. $\overline{INTA}$, all read commands and advanced write commands provide this timing. The normal write command is not valid until after the RDY signal must be valid. Since both normal and advanced write commands are generated by the 8288 for all write cycles, the advanced write may be used to generate a RDY indication even though the selected device uses the normal write command.

Since separate commands are provided for memory and I/O, no M/$\overline{IO}$ signal is specifically available as in the minimum mode to allow an early 'wait state required' indication for I/O devices. The $\overline{S2}$ status line, however is logically equivalent to the M/$\overline{IO}$ signal and can be used for this purpose.

6. Other Considerations

The $\overline{RQ}/\overline{GT}$ timing is covered in the next section and will not be duplicated here. The only additional signals to be considered in the maximum mode are the queue status lines QS0, QS1. These signals are changed on the leading edge of each clock cycle (high to low transition) including idle and wait cycles (the queue status is independent of the bus activity). External logic may sample the lines on the low to high transition of each clock cycle. When sampled, the signals indicate the queue activity in the previous clock cycle and therefore lag the CPU's activity by one cycle. The $\overline{TEST}$ input requirements are identical to those stated for the minimum mode.

To inform the 8288 of HALT status when a HALT instruction is executed, the 8086 will initiate a status transition from passive to HALT status. The status change will cause the 8288 to emit an ALE pulse with an indeterminate address. Since no bus cycle is initiated (no command is issued), the results of this address will not affect CPU operation (i.e., no response such as READY is expected from the system). This allows external hardware to latch and decode all transitions in system status.

3G. Bus Control Transfer (HOLD/HLDA and $\overline{RQ}/\overline{GT}$)

The 8086 supports protocols for transferring control of the local bus between itself and other devices capable of acting as bus masters. The minimum mode configuration offers a signal level handshake similar to the 8080 and 8085 systems. The maximum mode provides an enhanced pulse sequence protocol designed to optimize utilization of CPU pins while extending the system configurations to two prioritized levels of alternate bus masters. These protocols are simply techniques for arbitration of control of the CPU's local bus and should not be confused with the need for arbitration of a system bus.

1. MINIMUM MODE

The minimum mode 8086 system uses a hold request input (HOLD) to the CPU and a hold acknowledge (HLDA) output from the CPU. To gain control of the bus, a device must assert HOLD to the CPU and wait for the HLDA before driving the bus. When the 8086 can relinquish the bus, it floats the $\overline{RD}$, $\overline{WR}$, $\overline{INTA}$ and M/$\overline{IO}$ command lines, the $\overline{DEN}$ and DT/$\overline{R}$ bus control lines and the multiplexed address/data/status lines. The ALE signal is not three-stated. The CPU acknowledges the request with HLDA to allow the requestor to take control of the bus. The requestor must maintain the HOLD request active until it no longer requires the bus. The HOLD request to the 8086 directly affects the bus interface unit and only indirectly affects the execution unit. The CPU will continue to execute from its internal queue until either more instructions are needed or an operand transfer is required. This allows a high degree of overlap between CPU and auxiliary bus master operation. When the requestor drops the HOLD signal, the 8086 will respond by dropping HLDA. The CPU will not re-drive the bus, command and control signals from three-state until it needs to perform a bus transfer. Since the 8086 may still be executing from its internal queue when HOLD drops, there may exist a period of time during which no device is driving the bus. To prevent the command lines from drifting below the minimum VIH level during the transition of bus control, 22K ohm pull up resistors should be connected to the bus command lines. The timing diagram in Figure 3G1 shows the handshake sequence and 8086 timing to sample HOLD, float the bus, and enable/disable HLDA relative to the CPU clock.

To guarantee valid system operation, the designer must assure that the requesting device does not assert control of the bus prior to the 8086 relinquishing control and that the device relinquishes control of the bus prior to the 8086 driving the bus. The HOLD request into the 8086 must be stable THVCH ns prior to the CPU's low to high clock transition. Since this input is not synchronized by the CPU, signals driving the HOLD input should be synchronized with the CPU clock to guarantee the setup time is not violated. Either clock edge may be used. The maximum delay between HLDA and the 8086 floating the bus is TCLAZmax − TCLHAVmin = 70 ns. If the system cannot tolerate the 70 ns overlap, HLDA active from the 8086 should be delayed to the device. The minimum delay for the CPU to drive the control bus from HOLD inactive is THVCHmin + 3TCLCL = 635 ns and THVCHmin + 3TCLCL + TCHCL = 701 ns to drive the multiplexed bus. If the device does not satisfy these requirements, HOLD inactive to the 8086 should be delayed. The delay from HLDA inactive to driving the busses is TCLCL + TCLCHmin − TCLHAVmax = 158 ns for the control bus and 2TCLCL − TCLHAVmax = 240 ns for the data bus.

1.1 Latency of HLDA to HOLD

The decision to respond to a HOLD request is made in the bus interface unit. The major factors that influence the decision are the current bus activity, the state of the $\overline{LOCK}$ signal internal to the CPU (activated by the software LOCK prefix) and interrupts.

If the $\overline{LOCK}$ is not active, an interrupt acknowledge cycle is not in progress and the BIU (Bus Interface Unit) is executing a T4 or TI when the HOLD request is received, the minimum latency to HLDA is:

35 ns	THVCH min (Hold setup)
65 ns	TCHCL min
200 ns	TCLCL (bus float delay)
10 ns	TCLHAV min (HLDA delay)
310 ns	@ 5 MHz

The maximum delay under these conditions is:

34 ns	(just missed setup time)
200 ns	delay to next sample
82 ns	TCHCL max
200 ns	TCLCL (bus float delay)
160 ns	TCLHAV max (HLDA delay)
677 ns	@ 5 MHz

If the BIU just initiated a bus cycle when the HOLD Request was received, the worst case response time is:

34 ns	THVCH (just missed)
82 ns	TCHCL max
7*200	bus cycle execution
N*200	N wait states/bus cycle
160 ns	TCLHAV max (HLDA delay)
1.676 μs	@ 5 MHz, no wait states

Note, the 200 ns delay for just missing is included in the delay for bus cycle execution. If the operand transfer is a word transfer to an odd byte boundary, two bus cycles are executed to perform the transfer. The BIU will not acknowledge a HOLD request between the two bus cycles. This type of transfer would extend the above maximum latency by four additional clocks plus N additional wait states. With no wait states in the bus cycle, the maximum would be 2.476 microseconds.

Although the minimum mode 8086 does not have a hardware $\overline{LOCK}$ output, the software LOCK prefix may still be included in the instruction stream. The CPU internally reacts to the LOCK prefix as would the maximum mode 8086. Therefore, the LOCK does not allow a HOLD request to be honored until completion of the instruction following the prefix. This allows an instruction which performs more than one memory reference (ex. ADD [BX], CX; which adds CX to [BX]) to execute without another bus master gaining control of the bus between memory references. Since the LOCK signal is active for one clock longer than the instruction execution, the maximum latency to HLDA is:

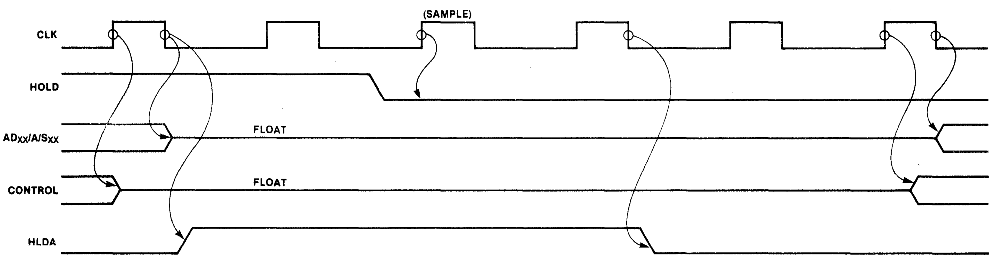


Figure 3G1. HOLD/HLDA Sequence

34 ns	THVCH (just miss)
200 ns	delay to next sample
82 ns	TCHCL max
(M + 1)*200 ns	LOCK instruction execution
200 ns	set up HLDA (internal)
160 ns	TCLHAV max (HLDA delay)
(M*200 ns) + 876 ns	@ 5 MHz

If the HOLD request is made at the beginning of an interrupt acknowledge sequence, the maximum latency to HLDA is:

34 ns	THVCH (just missed)
82 ns	TCHCL max
2600 ns	13 clock cycles for INTA
160 ns	TCLHAV max
2.876 μs	@ 5 MHz

1.2 Minimum Mode DMA Configuration

A typical use of the HOLD/HLDA signals in the minimum mode 8086 system is bus control exchange with DMA devices like the Intel 8257-5 or 8237 DMA controllers. Figure 3G2 gives a general interconnect for this type of configuration using the 8237-2. The DMA controller resides on the upper half of the 8086's local bus and shares the A8-A15 demultiplexing address latch of the 8086. All registers in the 8237-2 must be assigned odd addresses to allow initialization and interrogation by the CPU over the upper half of the data bus. The 8086 $\overline{RD}$/$\overline{WR}$ commands must be demultiplexed to provide separate I/O and memory commands which are compatible with the 8237-2 commands. The AEN control from the 8237-2 must disable the 8086 commands from the command bus, disable the address latches from the lower (A0-A7) and upper (A19-A16) address bus and select the 8237-2 address strobe (ADSTB) to the A8-A15 address latch. If the data bus is buffered, a pull-up resistor on the DEN line will keep the buffers disabled. The DMA controller will only transfer bytes between

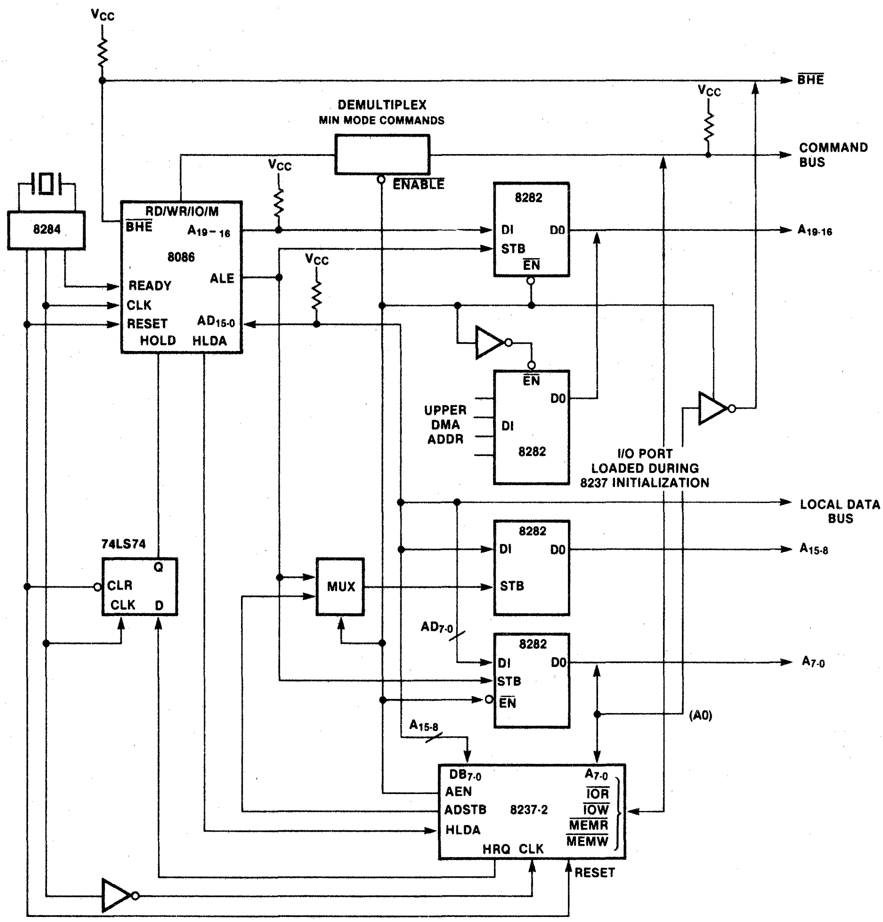


Figure 3G2. DMA Using the 8237-2

memory and I/O and requires the I/O devices to reside on an 8-bit bus derived from the 16-bit to 8-bit bus multiplex circuit given in Section 4. Address lines A7-A0 are driven directly by the 8237 and $\overline{BHE}$ is generated by inverting A0. If A19-A16 are used, they must be provided by an additional port with either a fixed value or initialized by software and enabled onto the address bus by AEN.

Figure 3G3 gives an interconnection for placing the 8257 on the system bus. By using a separate latch to hold the upper address from the 8257-5 and connecting the outputs to the address bus as shown, 16-bit DMA transfers are provided. In this configuration, AEN simultaneously enables A0 and $\overline{BHE}$ to allow word transfers. AEN still disables the CPU interface to the command and address busses.

2. MAXIMUM MODE ($\overline{RQ}/\overline{GT}$)

The maximum mode 8086 configuration supports a significantly different protocol for transferring bus control. When viewed with respect to the HOLD/HLDA sequence of the minimum mode, the protocol appears difficult to implement externally. However, it is necessary to understand the intent of the protocol and its purpose within the system architecture.

2.1 Shared System Bus ($\overline{RQ}/\overline{GT}$ Alternative)

The maximum mode $\overline{RQ}/\overline{GT}$ sequence is intended to transfer control of the CPU local bus between the CPU and alternate bus masters which reside totally on the local bus and share the complete CPU interface to the system bus. The complete interface includes the address latches, data transceivers, 8288 bus controller and 8289 multi master bus arbiter. If the alternate bus masters in the system do not reside directly on the 8086 local bus, system bus arbitration is required rather than local CPU bus arbitration. To satisfy the need for multi-master system bus arbitration at each CPU's system interface, the 8289 bus arbiter should be used rather than the CPU $\overline{RQ}/\overline{GT}$ logic.

To allow a device with a simple HOLD/HLDA protocol to gain control of a single CPU system bus, the circuit in Figure 3G4 could be used. The design is effectively a simple bus arbiter which isolates the CPU from the system bus when an alternate bus master issues a HOLD request. The output of the circuit, $\overline{AEN}$ (Address ENable), disables the 8288 and 8284 when the 8086 indicates idle status ($\overline{S0},\overline{S1},\overline{S2} = 1$), $\overline{LOCK}$ is not active and a HOLD request is active. With $\overline{AEN}$ inactive, the 8288 three-states the command outputs and disables DEN

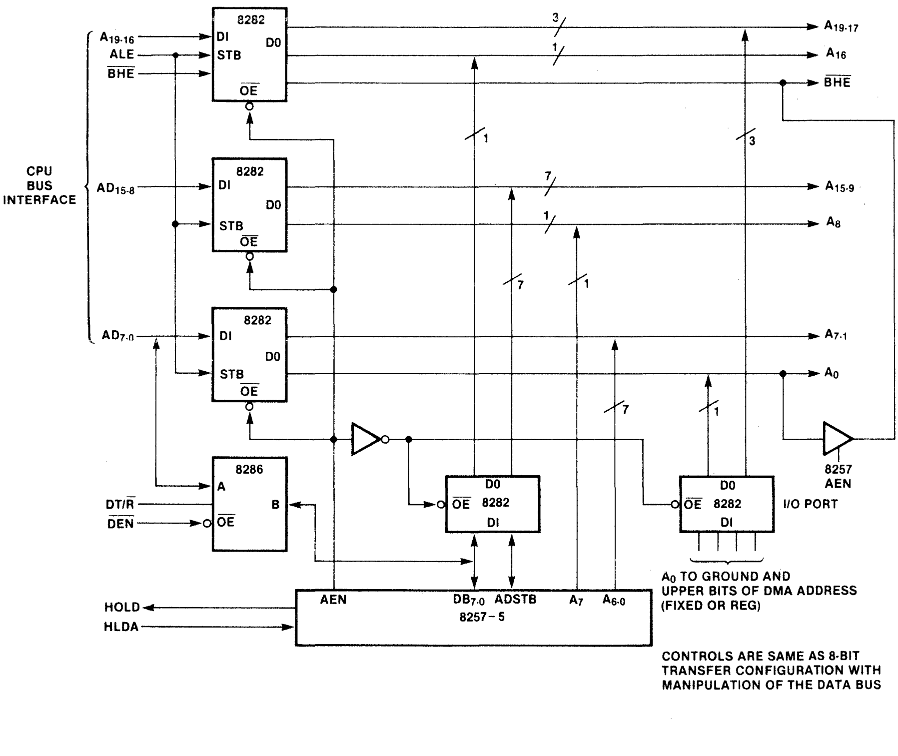


Figure 3G3. 8086 Min System, 8257 on System Bus 16-Bit Transfers which three-states the data bus transceivers. AEN must also three-state the address latch (8282 or 8283) outputs. These actions remove the 8086 from the system bus and allow the requesting device to drive the system bus. The AEN signal to the 8284 disables the ready input and forces a bus cycle initiated by the 8086 to wait until the 8086 regains control of the system bus. The CPU may actively drive its local bus during this interval.

The requesting device will not gain control of the bus during an 8086 initiated bus cycle, a locked instruction or an interrupt acknowledge cycle. The LOCK signal from the 8086 is active between INTA cycles to guarantee the CPU maintains control of the bus. Unlike the minimum mode 8086 HOLD response, this arbitration circuit allows the requestor to gain control of the bus between consecutive bus cycles which transfer a word operand on an odd address boundary and are not locked. Depending on the characteristics of the requesting device, any of the 74LS74 outputs can be used to generate a HLDA to the device.

Upon completion of its bus operations, the alternate bus master must relinquish control of the system bus and drop the HOLD request. After AEN goes inactive, the address latches and data transceivers are enabled but, if a CPU initiated bus cycle is pending, the 8288 will not drive the command bus until a minimum of 105 ns or maximum of 275 ns later. If the system is normally not ready, the 8284 AEN input may immediately be enabled with ready returning to the CPU when the selected device completes the transfer. If the system is normally ready, the 8284 AEN input must be delayed long enough to provide access time equivalent to a normal bus cycle. The 74LS74 latches in the design provide a minimum of TCLCHmin for the alternate device to float the system bus after releasing HOLD. They also provide 2TCLCL ns address access and 2TCLCL − TAEVCHmax ns (8288 command enable delay) command access prior to enabling 8284 ready detection. If HLDA is generated as shown in Figure 3G4, TCLCL ns are available for the 8086 to release the bus prior to issuing HLDA while HLDA is dropped almost immediately upon loss of HOLD.

A circuit configuration for an 8257-5 using this technique to interface with a maximum mode 8086 can be derived from Figure 3G3. The 8257-5 has its own address latch for buffering the address lines A15-A8 and uses its AEN output to enable the latch onto the address bus. The maximum latency from HOLD to HLDA for this circuit is dependent on the state of the system when the HOLD is issued. For an idle system the maximum delay is the propagation delay through the nand gate and R/S flip-flop (TD1) plus 2TCLCL plus TCLCHmax plus propagation delay of the 74LS74 and 74LS02 (TD2). For a locked instruction it becomes: TD1 + TD2 + (M + 2) *TCLCL + TCLCHmax where M is the number of clocks required for execution of the locked instruction. For the interrupt acknowledge cycle the latency is TD1 + TD2 + 9 *TCLCL + TCLCHmax.

2.2 Shared Local Bus (RQ/GT Usage)

The RQ/GT protocol was developed to allow up to two instruction set extension processors (co-processors) or other special function processors (like the 8089 I/O processor in local mode) to reside directly on the 8086 local bus. Each RQ/GT pin of the 8086 supports the full protocol for exchange of bus control (Fig. 3G5). The sequence consists of a request from the alternate bus master to gain control of the system bus, a grant from the CPU to indicate the bus has been relinquished and a release pulse from the alternate master when done. The two RQ/GT pins (RQ/GT0 and RQ/GT1) are prioritized with RQ/GT0 having the highest priority. The prioritization only occurs if requests have been received on both pins before a response has been given to either. For example, if a request is received on RQ/GT1 followed by a request on RQ/GT0 prior to a grant on RQ/GT1, RQ/GT0 will gain priority over RQ/GT1. However, if RQ/GT1 had already received a grant, a request on RQ/GT0 must wait until a release pulse is received on RQ/GT1.

The request/grant sequence interaction with the bus interface unit is similar to HOLD/HLDA. The CPU continues to execute until a bus transfer for additional instructions or data is required. If the release pulse is

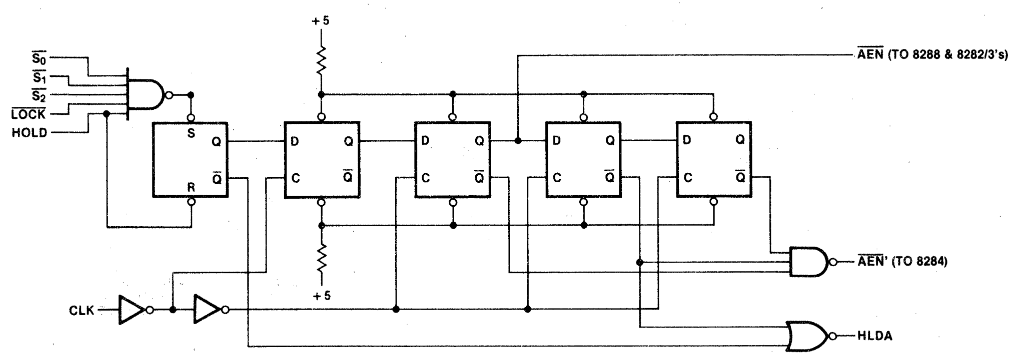


Figure 3G4. Circuit to Translate HOLD into AEN Disable for Max Mode 8086 received before the CPU needs the bus, it will not drive the bus until a transfer is required.

Upon receipt of a request pulse, the 8086 floats the multiplexed address, data and status bus, the $\overline{S0}$, $\overline{S1}$, and $\overline{S2}$ status lines, the $\overline{LOCK}$ pin and $\overline{RD}$. This action does not disable the 8288 command outputs from driving the command bus and does not disable the address latches from driving the address bus. The 8288 contains internal pull-up resistors on the $\overline{S0}$, $\overline{S1}$, and $\overline{S2}$ status lines to maintain the passive state while the 8086 outputs are three-state. The passive state prevents the 8288 from initiating any commands or activating DEN to enable the transceivers buffering the data bus. If the device issuing the $\overline{RQ}$ does not use the 8288, it must disable the 8288 command outputs by disabling the 8288 $\overline{AEN}$ input. Also, address latches not used by the requesting device must be disabled.

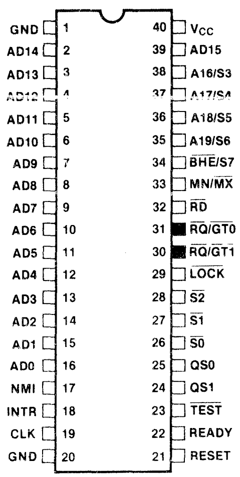


Figure 3G5. 8086 RQ/GT Connections

2.3 $\overline{RQ}/\overline{GT}$ Operation

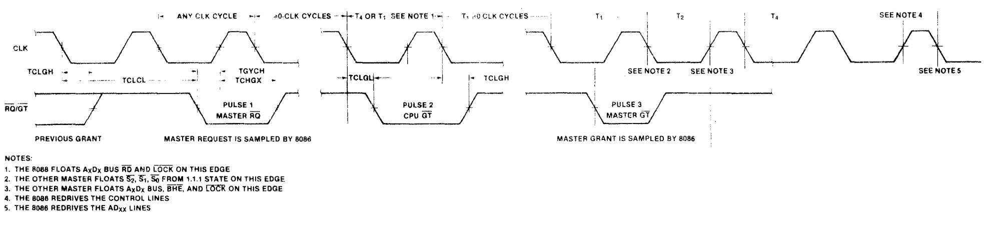
Detailed timing of the $\overline{RQ}/\overline{GT}$ sequence is given in Figure 3G6. To request a transfer of bus control via the $\overline{RQ}/\overline{GT}$ lines, the device must drive the line low for no more than one CPU clock interval to generate a request pulse. The pulse must be synchronized with the CPU clock to guarantee the appropriate setup and hold times to the clock edge which samples the $\overline{RQ}/\overline{GT}$ lines in the CPU. After issuing a request pulse, the device must begin sampling for a grant pulse with the next low to high clock edge. Since the 8086 can respond with a grant pulse in the clock cycle immediately following the request, the $\overline{RQ}/\overline{GT}$ line may not return to the positive level between the request and grant pulses. Therefore edge triggered logic is not valid for capturing a grant pulse. It also implies the circuitry which generates the request pulse must guarantee the request is removed in time to detect a grant from the CPU. After receiving the grant pulse, the requesting device may drive the local bus. Since the 8086 does not float the address and data bus, $\overline{LOCK}$ or $\overline{RD}$ until the high to low clock transition following the low to high clock transition the requestor uses to sample for the grant, the requestor should wait the float delay of the 8086 (TCLAZ) before driving the local bus. This precaution prevents bus contention during the access of bus control by the requestor.

To return control of the bus to the 8086, the alternate bus master relinquishes bus control and issues a release pulse on the same $\overline{RQ}/\overline{GT}$ line. The 8086 may drive the $\overline{S0}$-$\overline{S2}$ status lines, $\overline{RD}$ and $\overline{LOCK}$, three clock cycles after detecting the release pulse and the address/data bus TCHCLmin ns (clock high time) after the status lines. The alternate bus master should be three-stated off the local bus and have other 8086 interface circuits (8288 and address latches) re-enabled within the 8086 delay to regain control of the bus.

2.4 $\overline{RQ}/\overline{GT}$ Latency

The $\overline{RQ}$ to $\overline{GT}$ latency for a single $\overline{RQ}/\overline{GT}$ line is similar to the HOLD to HLDA latency. The cases given for the minimum mode 8086 also apply to the maximum mode. For each case the delay from $\overline{RQ}$ detection by the CPU to $\overline{GT}$ detection by the requestor is:

$$\text{(HOLD to HLDA delay)} - (\text{THVCH} + \text{TCHCL} + \text{TCLHAV})$$



NOTES:
1. THE 8088 FLOATS A_XD_X BUS $\overline{RD}$ AND $\overline{LOCK}$ ON THIS EDGE
2. THE OTHER MASTER FLOATS $\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$ FROM 1.1.1 STATE ON THIS EDGE
3. THE OTHER MASTER FLOATS A_XD_X BUS, $\overline{BHE}$, AND $\overline{LOCK}$ ON THIS EDGE
4. THE 8088 REDRIVES THE CONTROL LINES
5. THE 8086 REDRIVES THE AD_{XX} LINES

Figure 3G6. Request/Grant Sequence

This gives a clock cycle maximum delay for an idle bus interface. All other cases are the minimum mode result minus 476 ns. If the 8086 has previously issued a grant on one of the $\overline{RQ}/\overline{GT}$ lines, a request on the other $\overline{RQ}/\overline{GT}$ line will not receive a grant until the first device releases the interface with a release pulse on its $\overline{RQ}/\overline{GT}$ line. The delay from release on one $\overline{RQ}/\overline{GT}$ line to a grant on the other is typically one clock period as shown in Figure 3G7. Occasionally the delay from a release on $\overline{RQ}/\overline{GT1}$ to a grant on $\overline{RQ}/\overline{GT0}$ will take two clock cycles and is a function of a pending request for transfer of control from the execution unit. The latency from request to grant when the interface is under control of a bus master on the other $\overline{RQ}/\overline{GT}$ line is a function of the other bus master. The protocol embodies no mechanism for the CPU to force an alternate bus master off the bus. A watchdog timer should be used to prevent an errant alternate bus master from 'hanging' the system.

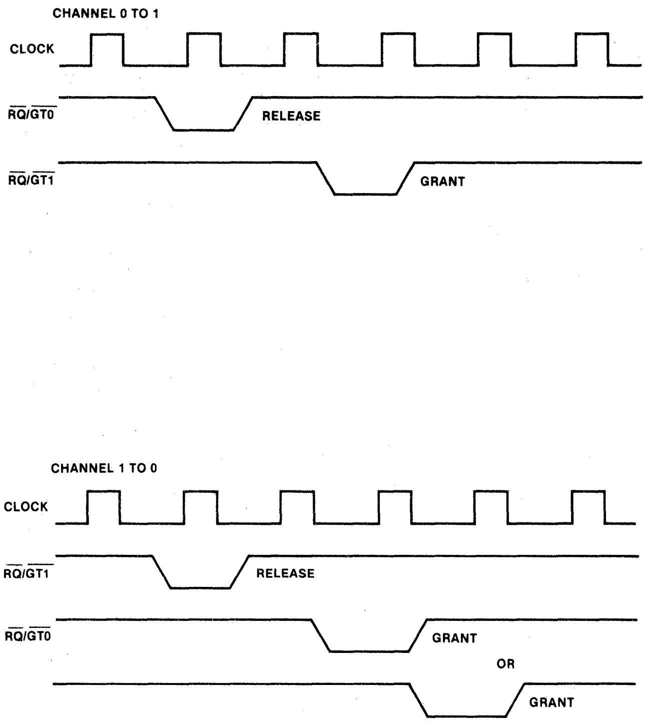


Figure 3G7. Channel Transfer Delay

2.5 RQ/GT to HOLD/HLDA Conversion

A circuit for translating a HOLD/HLDA hand-shake sequence into a $\overline{RQ}/\overline{GT}$ pulse sequence is given in Figure 3G8. After receiving the grant pulse, the HLDA is enabled TCHCLmin ns before the CPU has three-stated the bus. If the requesting circuit drives the bus within 20 ns of HLDA, it may be desirable to delay the acknowledge one clock period. The HLDA is dropped no later than one clock period after HOLD is disabled. The HLDA also drops at the beginning of the release pulse to provide 2TCLCL + TCLCH for the requestor to relinquish control of the status lines and 3TCLCL to float the remaining signals.

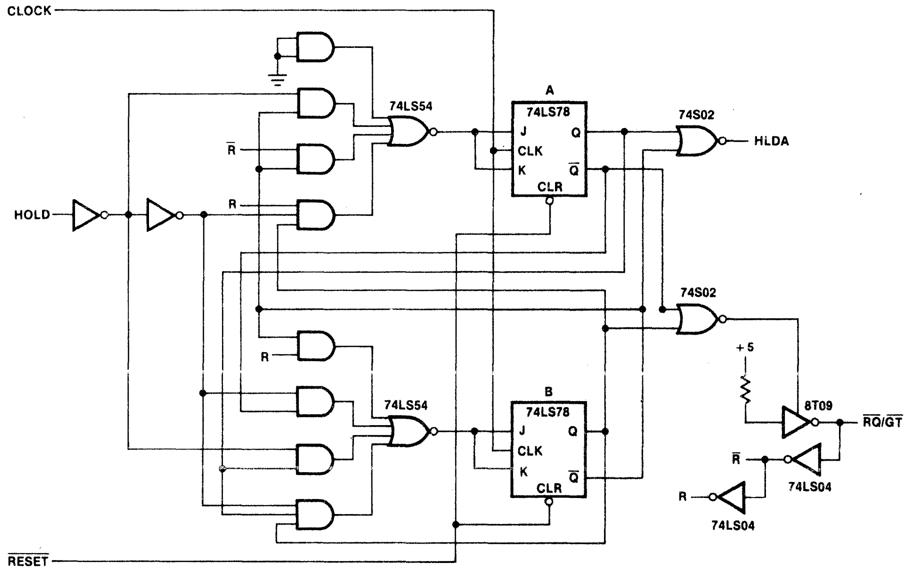


Figure 3G8a. HOLD/HLDA◄►$\overline{RQ}/\overline{GT}$ Conversion Circuit

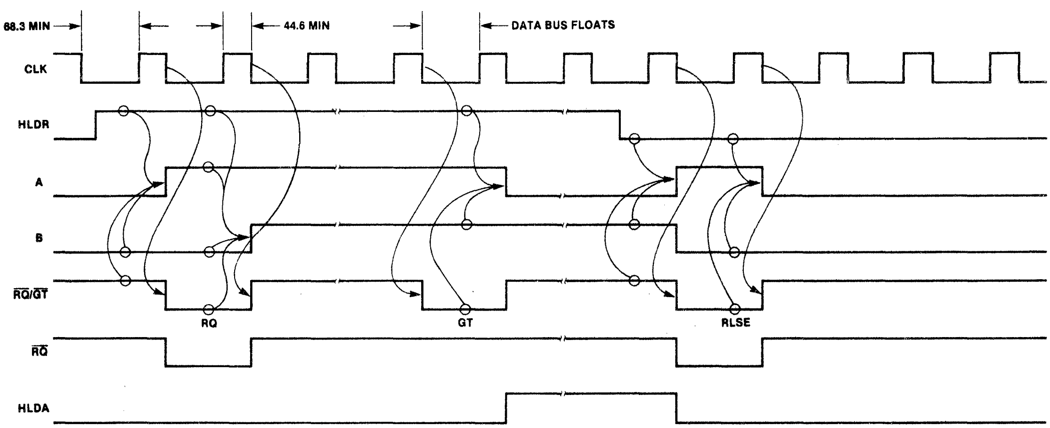


Figure 3G8b. HOLD/HLDA◄►$\overline{RQ}/\overline{GT}$ Conversion Timing

4. INTERFACING WITH I/O

The 8086 is capable of interfacing with 8- and 16-bit I/O devices using either I/O instructions or memory mapped I/O. The I/O instructions allow the I/O devices to reside in a separate I/O address space while memory mapped I/O allows the full power of the instruction set to be used for I/O operations. Up to 64K bytes of I/O mapped I/O may be defined in an 8086 system. To the programmer, the separate I/O address space is only accessible with INPUT and OUTPUT commands which transfer data between I/O devices and the AX (for 16-bit data transfers) or AL (for 8-bit data transfers) register. The first 256 bytes of the I/O space (0 to 255) are directly addressable by the I/O instructions while the entire 64K is accessible via register indirect addressing through the DX register. The later technique is particularly desirable for service procedures that handle more than one device by allowing the desired device address to be passed to the procedure as a parameter. I/O devices may be connected to the local CPU bus or the buffered system bus.

4A. Eight-Bit I/O

Eight-bit I/O devices may be connected to either the upper or lower half of the data bus. Assigning an equal number of devices to the upper and lower halves of the bus will distribute the bus loading. If a device is connected to the upper half of the data bus, all I/O addresses assigned to the device must be odd (A0 = 1). If the device is on the lower half of the bus, its addresses must be even (A0 = 0). The address assignment directs the eight-bit transfer to the upper (odd byte address) or lower (even byte address) half of the sixteen-bit data bus. Since A0 will always be a one or zero for a specific device, A0 cannot be used as an address input to select registers within a specific device. If a device on the upper half of the bus and one on the lower half are assigned addresses that differ only in A0 (adjacent odd and even addresses), A0 and $\overline{BHE}$ must be conditions of chip select decode to prevent a write to one device from erroneously performing a write to the other. Several techniques for generating I/O device chip selects are given in Figure 4A1.

The first technique (a) uses separate 8205's to generate chip selects for odd and even addressed byte peripherals. If a word transfer is performed to an even addressed device, the adjacent odd addressed I/O device is also selected. This allows accessing the devices individually with byte transfers or simultaneously as a 16-bit device with word transfers. Figure 4A1(b) restricts the chip selects to byte transfers, however a word transfer to an odd address will cause the 8086 to run two byte transfers that the decode technique will not detect. The third technique simply uses a single 8205 to generate odd and even device selects for byte transfers and will only select the even addressed eight-bit device on a word transfer to an even address.

If greater than 256 bytes of the I/O space or memory mapped I/O is used, additional decoding beyond what is shown in the examples may be necessary. This can be done with additional TTL, 8205's or bipolar PROMs (Intel's 3605A). The bipolar PROMs are slightly slower than multiple levels of TTL (50 ns vs 30 to 40 ns for TTL) but provide full decoding in a single package and allow inserting a new PROM to reconfigure the system I/O map without circuit board or wiring modifications (Fig. 4A2).

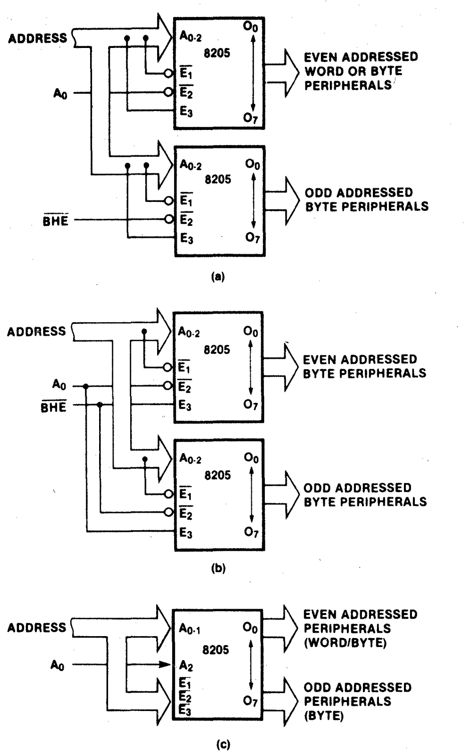


Figure 4A1. Techniques for I/O Device Chip Selects

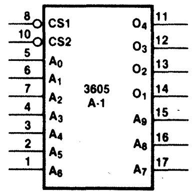


Figure 4A2. Bipolar PROM Decoder

One last technique for interfacing with eight-bit peripherals is considered in Figure 4A3. The sixteen-bit data bus is multiplexed onto an eight-bit bus to accommodate byte oriented DMA or block transfers to memory mapped eight-bit I/O. Devices connected to this interface may be assigned a sequence of odd and even addresses rather than all odd or even.

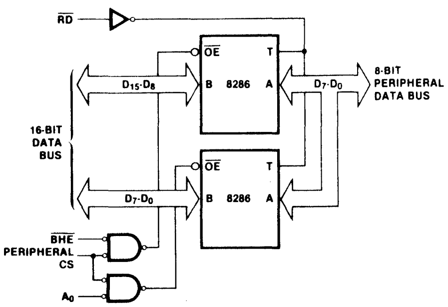
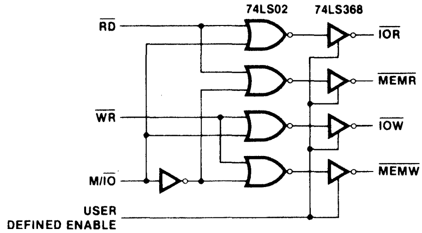


Figure 4A3. 16- to 8-Bit Bus Conversion



NOTE: IF IT IS NOT NECESSARY TO THREE-STATE THE COMMAND LINES, A DECODER (8205 OR 74S138) COULD BE USED. THE 74LS257 IS NOT RECOMMENDED SINCE THE OUTPUTS MAY EXPERIENCE VOLTAGE SPIKES WHEN ENTERING OR LEAVING THREE-STATE.

Figure 4C1. Decoding Memory and I/O $\overline{RD}$ and $\overline{WR}$ Commands for Minimum Mode 8086 Systems

4B. Sixteen-Bit I/O

For obvious reasons of efficient bus utilization and simplicity of device selection, sixteen-bit I/O devices should be assigned even addresses. To guarantee the device is selected only for word operations, A0 and $\overline{BHE}$ should be conditions of chip select code (Fig. 4B1).

Linear select techniques (Fig. 4C2) for I/O devices can only be used with devices that either reside in the I/O address space or require more than one active chip select (at least one low active and one high active). Devices with a single chip select input cannot use linear select if they are memory mapped. This is due to the assignment of memory address space FFFF0H-FFFFFH to reset startup and memory space 00000H-003FFH to interrupt vectors.

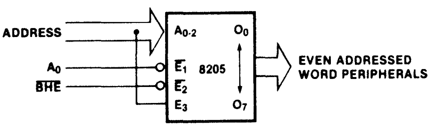


Figure 4B1. Sixteen-Bit I/O Decode

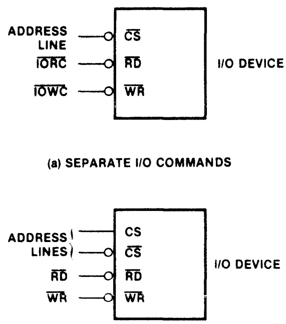


Figure 4C2. Linear Select for I/O

4C. General Design Considerations

MIN/MAX, MEMORY I/O MAPPED AND LINEAR SELECT

Since the minimum mode 8086 has common read and write commands for memory and I/O, if the memory and I/O address spaces overlap, the chip selects must be qualified by M/$\overline{IO}$ to determine which address space the devices are assigned to. This restriction on chip select decoding can be removed if the I/O and memory addresses in the system do not overlap and are properly decoded; all I/O is memory mapped; or $\overline{RD}$, $\overline{WR}$ and M/$\overline{IO}$ are decoded to provide separate memory and I/O read/write commands (Fig. 4C1). The 8288 bus controller in the maximum mode 8086 system generates separate I/O and memory commands in place of a M/$\overline{IO}$ signal. An I/O device is assigned to the I/O space or memory space (memory mapped I/O) by connection of either I/O or memory command lines to the command inputs of the device. To allow overlap of the memory and I/O address space, the device must not respond to chip select alone but must require a combination of chip select and a read or write command.

4D. Determining I/O Device Compatibility

This section presents a set of A.C. characteristics which represent the timing of the asynchronous bus interface of the 8086. The equations are expressed in terms of the CPU clock (when applicable) and are derived for minimum and maximum modes of the 8086. They represent the bus characteristics at the CPU.

The results can be used to determine I/O device requirements for operation on a single CPU local bus or buffered system bus. These values are not applicable to a Multibus system bus interface. The requirements for a Multibus system bus are available in the Multibus interface specification.

A list of bus parameters, their definition and how they relate to the A.C. characteristics of Intel peripherals are given in Table 4D1. Cycle dependent values of the parameters are given in Table 4D2. For each equation, if more than one signal path is involved, the equation reflects the worst case path.

ex. TAVRL(address valid before read active) =
(1) Address from CPU to $\overline{RD}$ active
(or)
(2) ALE (to enable the address through the address latches) to $\overline{RD}$ active

The worst case delay path is (1).

For the maximum mode 8086 configurations, TAVWLA, TWLWHA and TWLCLA are relative to the advanced write signal while TAVWL, TWLWH and TWLCL are relative to the normal write signal.

TABLE 4D1. PARAMETERS FOR PERIPHERAL COMPATIBILITY

TAVRL — Address stable before RD leading edge	(TAR)
TRHAX — Address hold after RD trailing edge	(TRA)
TRLRH — Read pulse width	(TRR)
TRLDV — Read to data valid delay	(TRD)
TRHDZ — Read trailing edge to data floating	(TDF)
TAVDV — Address to valid data delay	(TAD)
TRLRL — Read cycle time	(TRCYC)
TAVWL — Address valid before write leading edge	(TAW)
TAVWLA — Address valid before advanced write	(TAW)
TWHAX — Address hold after write trailing edge	(TWA)
TWLWH — Write pulse width	(TWW)
TWLWHA — Advanced write pulse width	(TWW)
TDVWH — Data set up to write trailing edge	(TDW)
TWHDX — Data hold from write trailing edge	(TWD)
TWLCL — Write recovery time	(TRV)
TWLCLA — Advanced write recovery time	(TRV)
TSVRL — Chip select stable before RD leading edge	(TAR)
TRHSX — Chip select hold after RD trailing edge	(TRA)
TSLDV — Chip select to data valid delay	(TRD)
TSVWL — Chip select stable before WR leading edge	(TAW)
TWHSX — Chip select hold after WR trailing edge	(TWA)
TSVWLA — Chip select stable before advanced write	(TAW)

Symbols in parentheses are equivalent parameters specified for Intel peripherals.

In the given list of equations, TWHDXB is the data hold time from the trailing edge of write for the minimum mode with a buffered data bus. For this equation, TCVCTX cannot be a minimum for data hold and a maximum for write inactive. The maximum difference is 50 ns giving the result TCLCH-50. If the reader wishes to verify the equations or derive others, refer to Section 3F for assistance with interpreting the 8086 bus timing diagrams.

Figure 4D1 shows four representative configurations and the compatible Intel peripherals (including wait states if required) for each configuration are given in Table 4D3. Configuration 1 and 2 are minimum mode demultiplexed bus 8086 systems without (1) and with (2) data bus transceivers. Configurations 3 and 4 are maximum mode systems with one (3) and two (4) levels of address and data buffering. The last configuration is characteristic of a multi-board system with bus buffers on each board. The 5 MHz parameter values for these configurations are given in Table 4D4 and demonstrate the relaxed device requirements for even a large complex configuration. The analysis assumes all components are exhibiting the specified worst case parameter values and are under the corresponding temperature, voltage and capacitive load conditions. If the capacitive loading on the 8282/83 or 8286/87 is less than the maximum, graphs of delay vs. capacitive loading in the respective data sheets should be used to determine the appropriate delay values.

TABLE 4D2. CYCLE DEPENDENT PARAMETER REQUIREMENTS FOR PERIPHERALS

(a) Minimum Mode

TAVRL = TCLCL + TCLRLmin − TCLAVmax = TCLCL − 100
TRHAX = TCLCL − TCLRHmax + TCLLHmin = TCLCL − 150
TRLRH = 2TCLCL − 60 = 2TCLCL − 60
TRLDV = 2TCLCL − TCLRLmax − TDVCLmin = 2TCLCL − 195
TRHDZ = TRHAVmin = 155 ns
TAVDV = 3TCLCL − TDVCLmin − TCLAVmax = 3TCLCL − 140
TRLRL = 4TCLCL = 4TCLCL
TAVWL = TCLCL + TCVCTVmin − TCLAVmax = TCLCL − 100
TWHAX = TCLCL + TCLLHmin − TCVCTXmax = TCLCL − 110
TWLWH = 2TCLCL − 40 = 2TCLCL − 40
TDVWH = 2TCLCL + TCVCTXmin − TCLDVmax = 2TCLCL − 100
TWHDX = TWHDZmin = 89
TWLCL = 4TCLCL = 4TCLCL
TWHDXB = TCLCHmin + (− TCVCTXmax + TCVCTXmin) = TCLCHmin − 50

Note: Delays relative to chip select are a function of the chip select decode technique used and are equal to: equivalent delay from address − chip select decode delay.

(b) Maximum Mode

TAVRL = TCLCL + TCLMLmin − TCLAVmax = TCLCL − 100
TRHAX = TCLCL − TCLMHmax + TCLLHmin = TCLCL − 40
TRLRH = 2TCLCL − TCLMLmax + TCLMHmin = 2TCLCL − 25
TRLDV = 2TCLCL − TCLMLmax − TDVCLmin = 2TCLCL − 65
TRHDZ = TRHAVmin = 155
TAVDV = 3TCLCL − TDVCLmin − TCLAVmax = 3TCLCL − 140
TRLRL = 4TCLCL = 4TCLCL
TAVWLA = TAVRL = TCLCL − 100
TAVWL = TAVRL + TCLCL = 2TCLCL − 100
TWHAX = TRHAX = TCLCL − 40
TWLWHA = TRLRH = 2TCLCL − 25
TWLWH = TRLRH − TCLCL = TCLCL − 25
TDVWH = 2TCLCL + TCLMHmin − TCLDVmax = 2TCLCL − 100
TWHDX = TCLCHmin − TCLMHmax + TCHDZmin = TCLCHmin − 30
TWLCL = 3TCLCL = 3TCLCL
TWLCLA = 4TCLCL = 4TCLCL

TABLE 4D3. COMPATIBLE PERIPHERALS (5 MHz 8086)

	Configuration			
	Minimum Mode		Maximum Mode	
	Unbuffered	Buffered	Buffered	Fully Buffered
8251A	✓	1W	✓	✓
8253-5	✓	1W	✓	✓
8255A-5	✓	1W	✓	✓
8257-5	✓	1W	✓	✓
8259A	✓	✓	✓	✓
8271	✓	1W	✓	✓
8273	✓	1W	✓	✓
8275	✓	1W	✓	✓
8279-5	✓	1W	✓	✓
8041A*	✓	1W	✓	✓
8741A	✓	1W	✓	✓
8291	✓	✓	✓	✓

*Includes other Intel peripherals based on the 8041A (i.e., 8292, 8294, 8295).

✓ implies full operation with no wait states.

W implies the number of wait states required.

TABLE 4D4. PERIPHERAL REQUIREMENTS FOR FULL SPEED OPERATION WITH 5 MHz 8086

	Configuration			
	Minimum Mode		Maximum Mode	
	Unbuffered	Buffered	Buffered	Fully Buffered
TAVRL	70	72	70	58
TRHAX	57	27	169	141
TRLRH	340	320	375	347
TRLDV	205	150	305	261
TRHDZ	155	158	382	360
TAVDV	430	400	400	372
TRLRL	800	770	800	772
TAVWL	70	72	270	258
TAVWLA	—	—	70	58
TWHAX	97	67	169	141
TWLWH	360	340	175	147
TWLWHA	—	—	375	347
TDVWH	300	339	270	258
TWHDX	88	15	95	13
TWLCL	800	772	600	572
TWLCLA	—	—	800	772
TSVRL	52	54	52	40
TRHSX	50	50	171	143
TSLDV	412	382	382	354
TSVWL	52	54	252	240
TWHSX	90	90	171	143
TSVWLA	—	—	52	40

— Not applicable.

Peripheral compatibility is determined from the equations given for the CPU by modifying them to account for additional delays from address latches and data transceivers in the configuration. Once the system configuration is selected, the system requirements can be determined at the peripheral interface and used to evaluate compatibility of the peripheral to the system. During this process, two areas must be considered. First, can the device operate at maximum bus bandwidth and if not, how many wait states are required. Second, are there any problems that cannot be resolved by wait states.

Examples of the first are TRLRH (read pulse width) and TRLDV (read access or $\overline{RD}$ active to output data valid). Consider address access time (valid address to valid data) for the maximum mode fully buffered configuration.

TAVDV = 3TCYC − 140 ns — address latch delay — address buffer delay — chip select decode delay — 2 transceiver delays

Assuming inverting latches, buffers and transceivers with 22 ns max delays (8283, 8287) and a bipolar PROM decode with 50 ns delay, the result is:

TAVDV = 322 ns @ 5 MHz

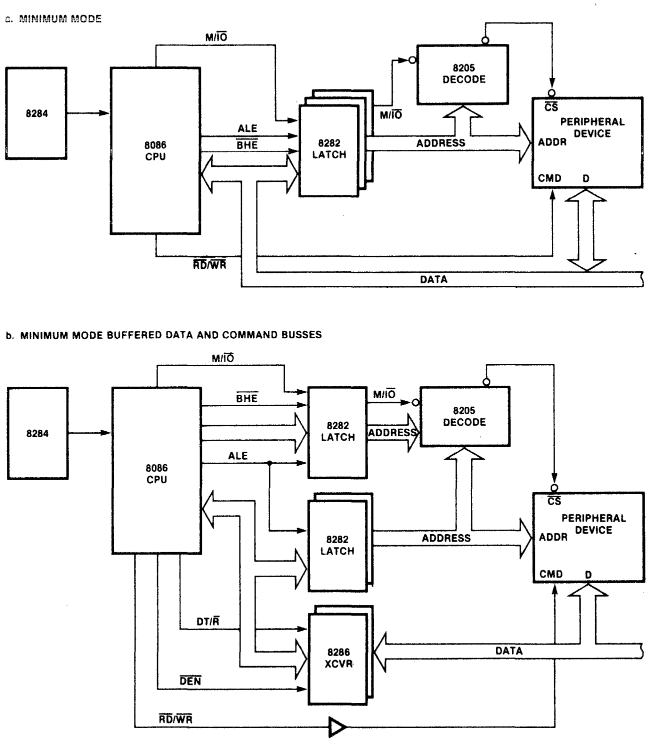
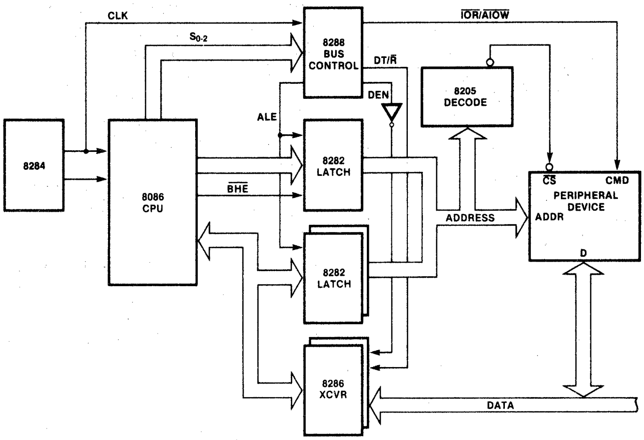


Figure 4D1. 8086 System Configurations

c. MAXIMUM MODE BUFFERED DATA BUS



NOTE: FOR OPTIMUM PERFORMANCE WITH INTEL PERIPHERALS, AIOW (ADVANCED WRITE) SHOULD BE USED.

d. MAXIMUM MODE DOUBLE BUFFERED SYSTEM

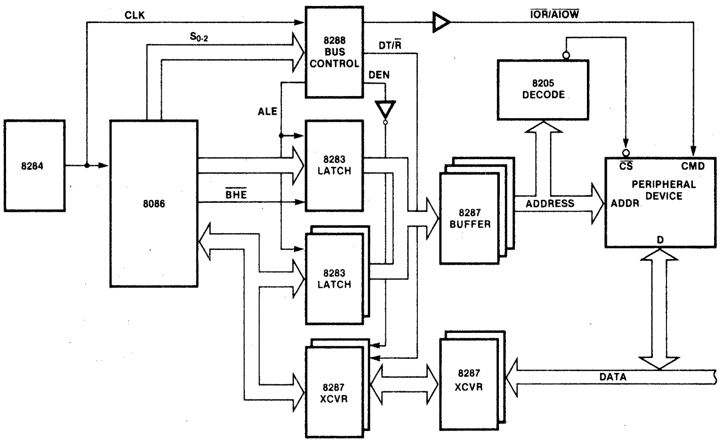


Figure 4D1. 8086 System Configurations (Con't)

The result gives the address to data valid delay required at the peripheral (in this configuration) to satisfy zero wait state CPU access time. If the maximum delay specified for the peripheral is less than the result, this parameter is compatible with zero wait state CPU operation. If not, wait states must be inserted until TAVDV + n * TCYC (n is the number of wait states) is greater than the peripherals maximum delay. If several parameters require wait states, either the largest number required should always be used or different transfer cycles can insert the maximum number required for that cycle.

The second area of concern includes TAVRL (address set up to read) and TWHDX (data hold after write). Incompatibilities in this area cannot be resolved by the insertion of wait states and may require either additional hardware, slowing down the CPU (if the parameter is related to the clock) or not using the device.

As an example consider address valid prior to advanced write low (TAVWLA) for the maximum mode fully buffered system.

TAVWLA = TCYC − 100 ns — address latch delay — address buffer delay — chip select decode delay + write buffer delay (minimum)

Assuming inverting latches and buffers with 22 ns delay (8283, 8287) and an 8205 address decoder with 18 ns delay

TAVWLA = 38 ns which is the time a 5 MHz 8086 system provides

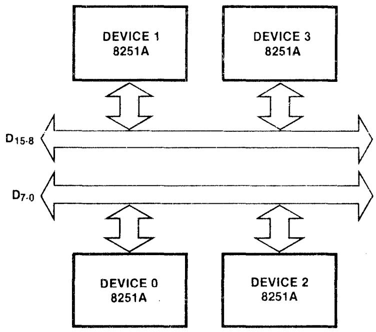
4E. I/O Examples

1. Consider an interrupt driven procedure for handling multiple communication lines. On receiving an interrupt from one of the lines, the invoked procedure polls the lines (reading the status of each) to determine which line to service. The procedure does not enable lines but simply services input and output requests until the associated output buffer is empty (for output requests) or until an input line is terminated (for the example only EOT is considered). On detection of the terminate condition, the routine will disable the line. It is assumed that other routines will fill a lines output buffer and enable the device to request output or empty the input buffer and enable the device to input additional characters.

The routine begins operation by loading CX with a count of the number of lines in the system and DX with the I/O address of the first line. The I/O addresses are assigned as shown in Figure 4E1 with 8251A's as the I/O devices. The status of each line is read to determine if it needs service. If yes, the appropriate routine is called to input or output a character. After servicing the line or if no service is needed, CX is decremented and DX is incremented to test the next line. After all lines have been tested and serviced, the routine terminates. If all interrupts from the lines are OR'd together, only one interrupt is used for all lines. If the interrupt is input to the CPU through an 8259A interrupt controller, the 8259A should be programmed in the level triggered mode to guarantee all line interrupts are serviced.

To service either an input or output request, the called routine transfers DX to BX, and shifts BX to form the offset for this device into the table of input or output buffers. The first entry in the buffer is an index to the next character position in the buffer and is loaded into the SI register. By specifying the base address of the table of buffers as a displacement into the data segment, the base + index + displacement addressing mode allows direct access to the appropriate memory location. 8086 code for part of this example is shown in Figure 4E2.

2. As a second example, consider using memory mapped I/O and the 8086 string primative instructions to perform block transfers between memory and I/O. By assigning a block of the memory address space (equivalent in size to the maximum block to be transferred to the I/O device) and decoding this address space to generate the I/O device's chip select, the block transfer capability is easily implemented. Figure 4E3 gives an interconnect for 16-bit I/O devices while Figure 4E4 incorporates the 16-bit bus to 8-bit bus multiplexing scheme to support 8-bit I/O devices. A code example to perform such a transfer is shown in Figure 4E5.



DEVICES ARE CONNECTED TO THE UPPER AND LOWER HALVES OF THE DATA BUS.

ADDRESS		
0	DEVICE 0	DATA
1	DEVICE 1	DATA
2	DEVICE 0	CONTROL/STATUS
3	DEVICE 1	CONTROL/STATUS
4	DEVICE 2	DATA
5	DEVICE 3	DATA
6	DEVICE 2	CONTROL/STATUS
7	DEVICE 3	CONTROL/STATUS
ETC.	"	"

Figure 4E1. Device Assignment

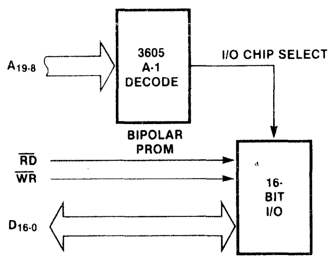
```
; THIS CODE DEMONSTRATES TESTING DEVICE
; STATUS FOR SERVICE, CONSTRUCTING THE
; APPROPRIATE LINE BUFFER ADDRESS FOR INPUT
; AND OUTPUT AND SERVICING AN INPUT
; REQUEST

                MASK EQU OFFFDH
CHECK_STATUS:   INPUT  AL, DX               ; GET 8251A STATUS.
                MOV    AH, AL
                TEST   AH, READ_OR_WRITE_STATUS
                JZ     NEXT_IO
                CALL   ADDRESS
                TEST   AH, READ STATUS
                JZ     WRITE_SERVICE
                CALL   READ
                TEST   AH, WRITE STATUS
                JZ     NEXT_IO
WRITE_SERVICE:  CALL   WRITE
NEXT_IO:        DEC    CX                   ; TEST IF DONE.
                JNC    EXIT                 ; YES, RESTORE & RETURN.
                AND    DX, MASK             ; REMOVE A1 AND
                ADD    DX, 3                ; INCREMENT ADDRESS.
                OR     DX, 2                ; SELECT STATUS FOR
                JMP    CHECK_STATUS         ; NEXT INPUT.

ADDRESS:        AND    DX, MASK             ; SELECT DATA.
                MOV    BH, DL               ; CONSTRUCT BUFFER
                INC    BH                   ; DISPLACEMENT FOR
                SHR    BH                   ; THIS DEVICE.
                XOR    BL, BL               ; BX IS THE DISPLACEMENT.
                RET

READ:           INPUT AL, DX                          ; READ CHARACTER.
                MOV SI, READ_BUFFERS [BX]             ; GET CHARACTER POINTER.
                MOV READ_BUFFERS [BX + SI], AL        ; STORE CHARACTER.
                INC READ_BUFFERS [BX]                 ; INCR CHARACTER POINTER.
                CMP AL, EOT                           ; END OF TRANSMISSION?
                JNZ CONT_READ
                CALL DISABLE READ                     ; YES, DISABLE RECEIVER.
                CONT_READ:   RET                      ; SEND MESSAGE THAT INPUT
                                                      ; IS READY.
```

Figure 4E2.



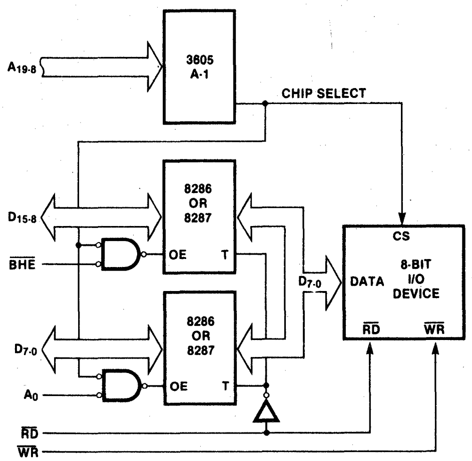
TRANSFER 256 BYTE BLOCKS TO THE I/O DEVICE

THE ADDRESS SPACE ASSIGNED TO THE I/O DEVICE IS

	A_{19} … A_8	A_7 … A_0
FROM	← BASE ADDRESS →	← 0's →
THRU	← BASE ADDRESS →	← 1's →

MEMORY DATA NEED NOT BE ALIGNED TO EVEN ADDRESS BOUNDARIES
I/O TRANSFERS MUST BE WORD TRANSFERS TO EVEN ADDRESS BOUNDARIES

Figure 4E3. Block Transfer to 16-Bit I/O Using 8086 String Primatives



ADDRESS ASSIGNMENT SAME AS PREVIOUS EXAMPLE. 16-BIT BUS IS MULTIPLEXED ONTO AN 8-BIT PERIPHERAL BUS.

Figure 4E4. Block Transfer to 8-Bit I/O Using 8086 String Primatives

```
          ; DEFINE THE I/O ADDRESS SPACE
            I/O SEGMENT
            ORG BLOCK_ADDRESS
I/O_BLOCK:  DW 128 DUP (?)
            I/O ENDS

          ; ASSUME THE DATA IS FROM THE CURRENT
          ; DATA SEGMENT
            CLD                          ; DF = FORWARD
            LES DI, I/O_BLOCK_ADDRESS    ; I/O BLOCK ADDRESS
                                         ; CONTAINS THE ADDRESS
                                         ; OF I/O BLOCK
            MOV CX, BLOCK_LENGTH
            MOV SI, SOURCE_ADDRESS
            MOVS I/O BLOCK               ; PERFORM WORD TRANSFERS

          ; END CODE EXAMPLE
```

NOTE THE CODE IS CAPABLE OF PERFORMING BYTE TRANSFERS BY CHANGING THE I/O BLOCK DEFINITION FROM 128 WORD TO 256 BYTES

Figure 4E5. Code for Block Transfers

5. INTERFACING WITH MEMORIES

Figure 5.1 is a general block diagram of an 8086 memory. The basic characteristics of the diagram are the partitioning of the 16-bit word memory into high and low 8-bit banks on the upper and lower halves of the data bus and inclusion of $\overline{\text{BHE}}$ and A0 in the selection of the banks. Specific implementations depend on the type of memory and the system configuration.

5A. ROM and EPROM

The easiest devices to interface to the system are ROM and EPROM. Their byte format provides a simple bus interface and since they are read only devices, A0 and $\overline{\text{BHE}}$ need not be included in their chip enable/select decoding (chip enable is similar to chip select but additionally determines if the device is in active or standby power mode). The address lines connected to the devices start with A1 and continue up to the maximum number the device can accept, leaving the remaining address lines for chip enable/select decoding. To connect the devices directly to the multiplexed bus, they must have output enables. The output enable is also necessary to avoid bus contention in other configurations. Figure 5A1 shows the bus connections for ROM and EPROM memories. No special decode techniques are required for generating chip enables/selects. Each valid decode selects one device on the upper and lower halves of bus to allow byte and word access. Byte access is achieved by reading the full word onto the bus with the 8086 only accepting the desired byte. For the minimum mode 8086, if $\overline{\text{RD}}$, $\overline{\text{WR}}$ and M/$\overline{\text{IO}}$ are not decoded to form separate commands for memory and I/O, and the I/O space overlaps the memory space assigned to the EPROM/ROM then M/$\overline{\text{IO}}$ (high active) must be a condition of chip enable/select decode. The output enable is controlled by the system memory read signal.

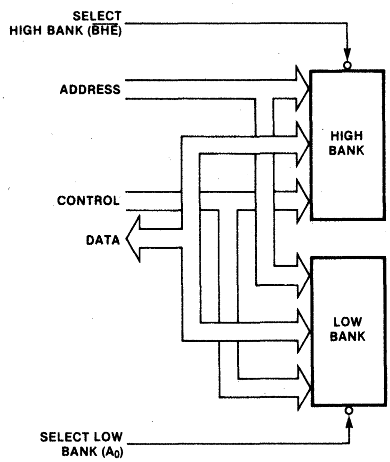
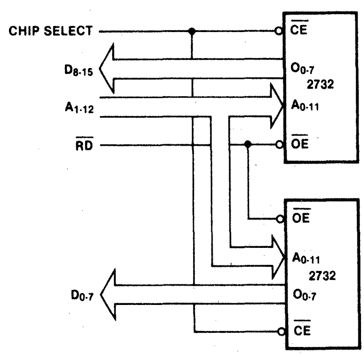


Figure 5.1. 8086 Memory Array



NOTE A_0 AND $\overline{\text{BHE}}$ ARE NOT USED.

Figure 5A1. EPROM/ROM Bus Interface

Static ROM's and EPROM's have only four parameters to evaluate when determining their compatibility to the system. The parameters, equations and evaluation techniques given in the I/O section are also applicable to these devices. The relationship of parameters is given in Table 5A1. TACC and TCE are related to the same equation and differ only by the delay associated with the chip enable/select decoder. As an example, consider a 2716 EPROM memory residing on the multiplexed bus of a minimum mode configuration:

TACC = 3TCLCL − 140 − address buffer delay = 430 ns
(8282 = 30 ns max delay)

TCE = TACC − decoder delay = 412 ns
(8205 decoder delay = 18 ns)

TOE = 2TCLCL − 195 = 205 ns

TDF = = 155 ns

TABLE 5A1. EPROM/ROM PARAMETERS

TOE — Output Enable to Valid Data ≡ TRLDV
TACC — Address to Valid Data ≡ TAVDV
TCE — Chip Enable to Valid Data ≡ TSLDV
TDF — Output Enable High to Output Float ≡ TRHDZ

The results are the times the system configuration requires of the component for full speed compatibility with the system. Comparing these times with 2716 parameter limits indicates the 2716-2 will work with no wait states while the 2716 will require one wait state. Table 5A2 demonstrates EPROM/ROM compatibility for the configurations presented in the I/O section. Before designing a ROM or EPROM memory system, refer to AP-30 for additional information on design techniques that give the system an upgrade path from 16K to 32K and 64K devices.

TABLE 5A2. COMPATIBLE EPROM/ROM (5 MHz 8086)

	Configuration			
	Minimum Mode		Maximum Mode	
	Unbuffered	Buffered	Buffered	Fully Buffered
2716-1	✓	✓	✓	✓
2716-2	✓	1W	1W	1W
2732	1W	1W	1W	1W
2332	✓	✓	✓	✓
2364	✓	✓	✓	✓

5B. Static RAM

Interfacing static RAM to the system introduces several new requirements to the memory design. A0 and $\overline{\text{BHE}}$ must be included in the chip select/chip enable decoding of the devices and write timing must be considered in the compatibility analysis.

For each device, the data bus connections must be restricted to either the upper or lower half of the data bus. Devices like the 2114 or 2142 must not straddle the upper and lower halves of the data bus (Fig. 5B1). To allow selecting either the upper byte, lower byte or full 16-bit word for a write operation, $\overline{\text{BHE}}$ must be a condition of decode for selecting the upper byte and A0 must be a condition of decode for selecting the lower byte. Figure 5B2 gives several selection techniques for devices with single chip selects and no output enables (2114, 2141, 2147). Figure 5B3 gives selection techniques for devices with chip selects and output enables.

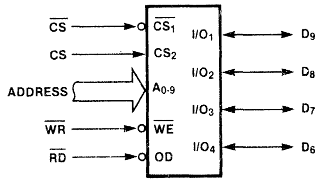


Figure 5B1. Incorrect Connection of 2142 Across Byte Boundaries

The first group requires inclusion of A0 and $\overline{\text{BHE}}$ to decode or enable the chip selects. Since these memories do not have output enables, read and write are used as enables for chip select generation to prevent bus contention. If read and write are not used to enable the chip selects, devices with common input/output pins (like the 2114) will be subjected to severe bus contention between chip select and write active. For devices with separate input/output lines (like 2141, 2147), the outputs can be externally buffered with the buffer enable controlled by read. This solution will only allow bus contention between memory devices in the array during chip select transition periods. These techniques are considered in more detail in Section 2C.

For devices with output enables (2142), write may be gated with $\overline{\text{BHE}}$ and A0 to provide upper and lower bank write strobes. This simplifies chip select decoding by eliminating $\overline{\text{BHE}}$ and A0 as a condition of decode. Although both devices are selected during a byte write operation, only one will receive a write strobe. No bus contention will exist during the write since a read command must be issued to enable the memory output drivers.

If multiple chip selects are available at the device, $\overline{\text{BHE}}$ and A0 may directly control device selection. This allows normal chip select decoding of the address space and direct connection of the read and write commands to the devices. Alternately, the multiple chip select inputs of the device could directly decode the address space (linear select) and be combined with the separate write strobe technique to minimize the control circuitry needed to generate chip selects.

As with the EPROM's and ROM's, if separate commands are not provided for memory and I/O in the minimum mode 8086 and the address spaces overlap, M/$\overline{\text{IO}}$ (high active) must be a condition of chip select decode. Also, the address lines connected to the memory devices must start with A1 rather than A0.

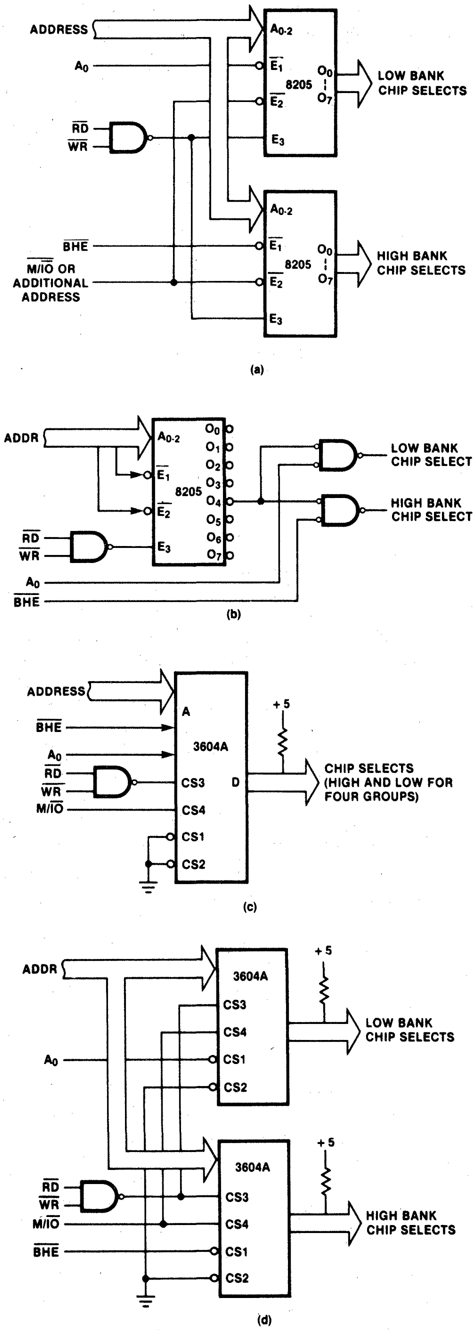


Figure 5B2. Generating Chip Selects for Devices without Output Enables

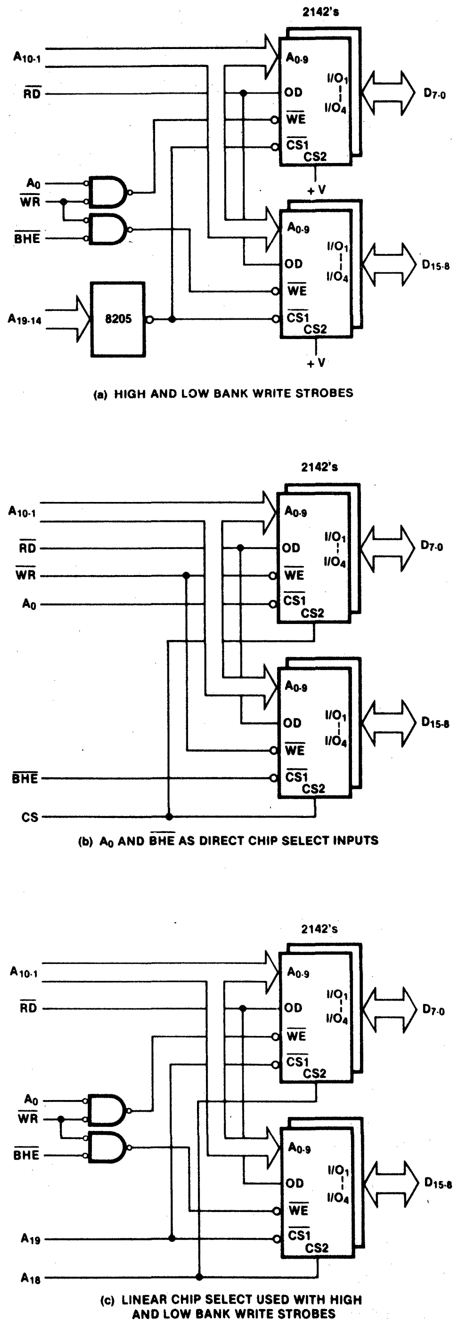


Figure 5B3. Chip Selection for Devices with Output Enables

For analysis of RAM compatibility, the write timing parameters listed in Table 5B1 may also need to be considered (depending on the RAM device being considered). The CPU clock relative timing is given in Table 5B2. The equations specify the device requirements at the CPU and provide a base for determining device requirements in other configurations. As an example consider the write timing requirements of a 2142 in a maximum mode buffered 8086 system (Figure 5B4). The 2142 write parameters that must be analyzed are TWA advanced write pulse width, TWR write release time, TDWA data to write time overlap and TDH data hold from write time.

TWA = 2TCLCL − TCLMLmax + TCLMHmin = 375 ns.
TWR = 2TCLCL − TCLMHmax + TCLLHmin + TSHOVmin = 170 ns.
TDWA = 2TCLCL − TCLDVmax + TCLMHmin − TIVOVmax = 265 ns.
TDH = TCLCH − TCLMHmax + TCHDXmin + TIVOVmin = 95 ns.

TABLE 5B1. TYPICAL WRITE TIMING PARAMETERS

TW	Write Pulse Width
TWR	Write Release (Address Hold From End of Write)
TDW	Data and Write Pulse Overlap
TDH	Data Hold From End of Write
TAW	Address Valid to End of Write
TCW	Chip Select to End of Write
TASW	Address Valid to Beginning of Write

TABLE 5B2. CYCLE DEPENDENT WRITE PARAMETERS FOR RAM MEMORIES

(a) Minimum Mode

TW = TWLWH = 2TCLCL − 60 = 340 ns
TWR = TCLCL − TCVCTXmax + TCLLHmin = 90 ns
TDW = 2TCLCL − TCLDVmax + TCVCTXmin = 300 ns
TDH = TWHDX = 88 ns
TAW = 3TCLCL − TCLAVmax + TCVCTXmin = 500 ns
TCW = TAW − Chip Select Decode
TASW = TCLCL − TCLAVmax + TCVCTXmin = 100 ns

(b) Maximum Mode

TW = TCLCL − TCLMLmax + TCLMHmin = 175 ns
TWR = TCLCL − TCLMHmax + TCLLHmin = 165 ns
TDW = TW = 175 ns
TDH = TCLCHmin − TCLMHmax + TCHDXmin = 93 ns
TAW = 3TCLCL − TCLAVmax + TCLMHmin = 500 ns
TCW = TAW − Chip Select Decode
TASW = 2TCLCL − TCLAVmax + TCLMLmin = 300 ns
TWA* = TW + TCLCL = 375 ns
TDWA* = 2TCLCL − TCLDVmax + TCLMHmin = 300 ns
TASWA* = TASW − TCLCL = 100 ns

*Relative to Advanced Write.

Comparing these results with the 2142 family indicates the standard 2142 write timing is fully compatible with this 8086 configuration. Read timing analysis is also necessary to completely determine compatibility of the devices.

5C. Dynamic RAM

Dynamic RAM is perhaps the most complex device to design into a system. To relieve the engineer of most of this burden, Intel provides the 8202 dynamic RAM controller as part of the 8086 family of peripheral devices. This section will discuss using the 8202 with the 8086 to build a dynamic memory system for an 8086 system. For additional information on the 8202, refer to the 8202 data sheet (9800873) and application note AP-45 Using the 8202 Dynamic RAM Controller (9800809A).

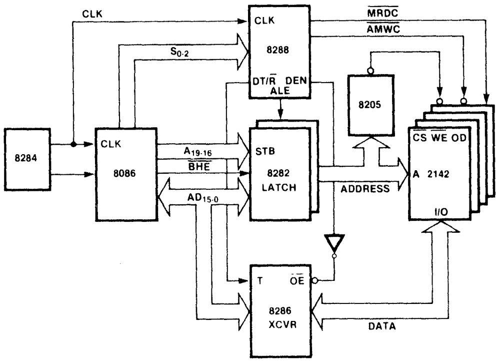


Figure 5B4. Sample Configuration for Compatibility Analysis Example

5.C.1 Standard 8086-8202 Interconnect

Figure 5.C.1.1 shows a standard interconnection for an 8202 into an 8086 system. The configuration accommodates 64K words (128K bytes) of dynamic RAM addressable as words or bytes. To access the RAM, the 8086 initiates a bus cycle with an address that selects the 8202 (via $\overline{PCS}$) and the appropriate transfer command ($\overline{MRDC}$ or $\overline{MWTC}$). If the 8202 is not performing a refresh cycle, the access starts immediately, otherwise, the 8086 must wait for completion of the refresh. $\overline{XACK}$ from the 8202 is connected to the 8284 RDY input to force the CPU to wait until the RAM cycle is completed before the CPU can terminate the bus cycle. This effectively synchronizes the asynchronous events of refresh and CPU bus cycles. The normal write command ($\overline{MWTC}$) is used rather than the advanced command ($\overline{AMWC}$) to guarantee the data is valid at the dynamic RAMS before the write command is issued. The gating of $\overline{WE}$ with A0 and $\overline{BHE}$ provides selective write strobes to the upper and lower banks of memory to allow byte and word write operations. The logic which generates the strobe for the data latches allows read data to propagate to the system as soon as the data is available and latches the data on the trailing edge of $\overline{CAS}$.

DETAILED TIMING

Read Cycle

For no wait state operation, the 8086 requires data to be valid from $\overline{MRDC}$ in:

2TCLCL − TCLML − TDVCL − buffer delays = 291 ns.

Since the 8202 is $\overline{CAS}$ access limited, we need only examine $\overline{CAS}$ access time. The 8202/2118 guarantees data valid from 8202 $\overline{RD}$ low to be:

(tph + 3tp + 100 ns) 8202 TCC delay + TCAC for the 2118

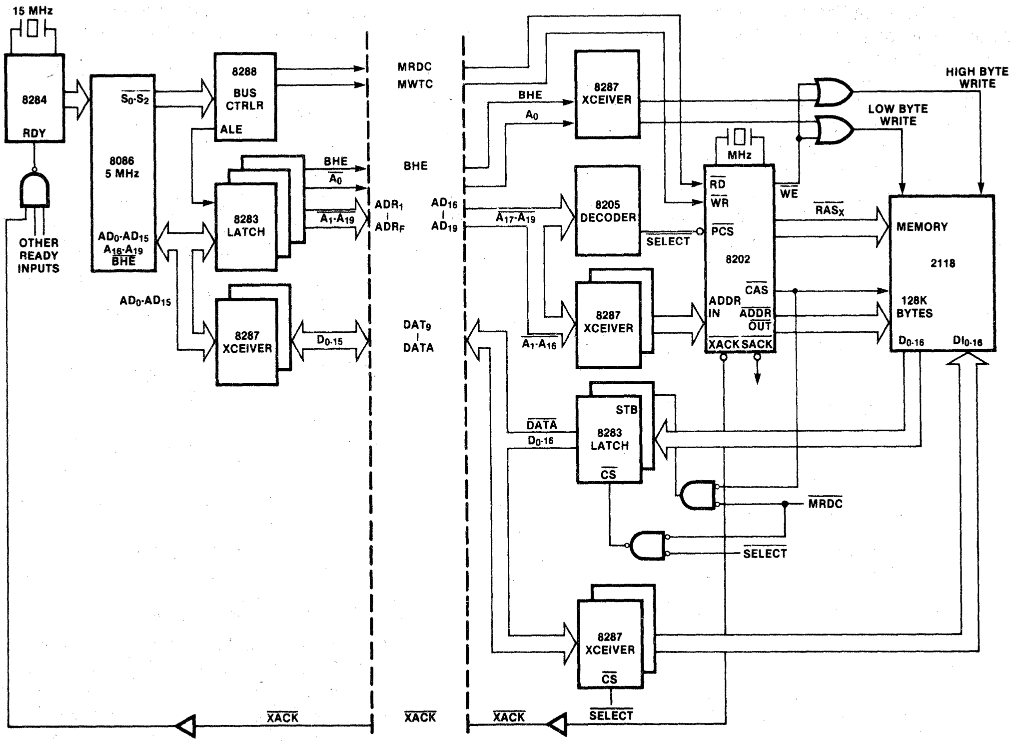


Figure 5C1.1. 5 MHz 8086/8202/128K Byte System — Double Data, Control and Address Buffering (Note: Bus driver on 8202 is not needed if less than 64K bytes are used)

For a 25 MHz 8202 and 2118-3, we get 297 ns which is insufficient for no wait state operation. If only 64K bytes are accessed, the 8202 requires only (tph + 3tp + 85 ns) giving 282 ns access and no wait states required. Refer to Figure 5.C.1.2 and 5C.1.3 for timing information on the 8202 and 2118.

Write Cycle

An important consideration for dynamic RAM write cycles is to guarantee data to the RAM is valid when both $\overline{\text{CAS}}$ and $\overline{\text{WE}}$ are active. For the 2118, if $\overline{\text{WE}}$ is valid prior to $\overline{\text{CAS}}$, the data setup is to $\overline{\text{CAS}}$ and if $\overline{\text{CAS}}$ is valid before $\overline{\text{WE}}$ (as would occur during a read modify write cycle) the data setup time is to $\overline{\text{WE}}$. For the 8202, the $\overline{\text{WR}}$ to $\overline{\text{CAS}}$ delay is analyzed to determine the data setup time to $\overline{\text{CAS}}$ inherently provided by the 8202 command to $\overline{\text{RAS}}/\overline{\text{CAS}}$ timing. The minimum delay from $\overline{\text{WR}}$ to $\overline{\text{CAS}}$ is:

$$\text{TCCmin} = \text{tph} + 2\text{tp} + 25 = 127 \text{ ns @ 25 MHz}$$

Subtracting buffer delays and data setup at the 2118, we have 83 ns to generate valid data after the write command is issued by the CPU (in this case the 8288). Since the 8086 will not guarantee valid data until TCLAVmax − TCLMLmin = 100 ns from the advanced write signal, the normal write signal is used. The normal write $\overline{\text{MWTC}}$ guarantees data is valid 100 ns before it is active. The worst case write pulse width is approximately 175 ns which is sufficient for all 2118's.

Synchronization

To force the 8086 to wait during refresh the $\overline{\text{XACK}}$ or $\overline{\text{SACK}}$ lines must be returned to the 8284 ready input. The maximum delay from $\overline{\text{RD}}$ to $\overline{\text{SACK}}$ (if the 8202 is not performing refresh) is TAC = tp + 40 = 80 ns. To prevent a wait state at the 8086, RDY must be valid at the 8284 TCLCHmin − TCLMLmax − TR1VCLmax = 48 ns after the command is active. This implies that under worst case conditions, one wait state will be inserted for every read cycle. Since $\overline{\text{MWTC}}$ does not occur until one clock later, two wait states may be inserted for writes.

The $\overline{\text{XACK}}$ from command delay will assert RDY TCC + TCX = (tph + 3tp + 100) + (5tp + 20) = 460 ns after the command. This will typically insert one or two wait states.

Unless 2118-3's are used in 64K byte or less memories, $\overline{\text{SACK}}$ must not be used since it does not guarantee a wait state. From the previous access time analysis we saw that other configurations required a wait state.

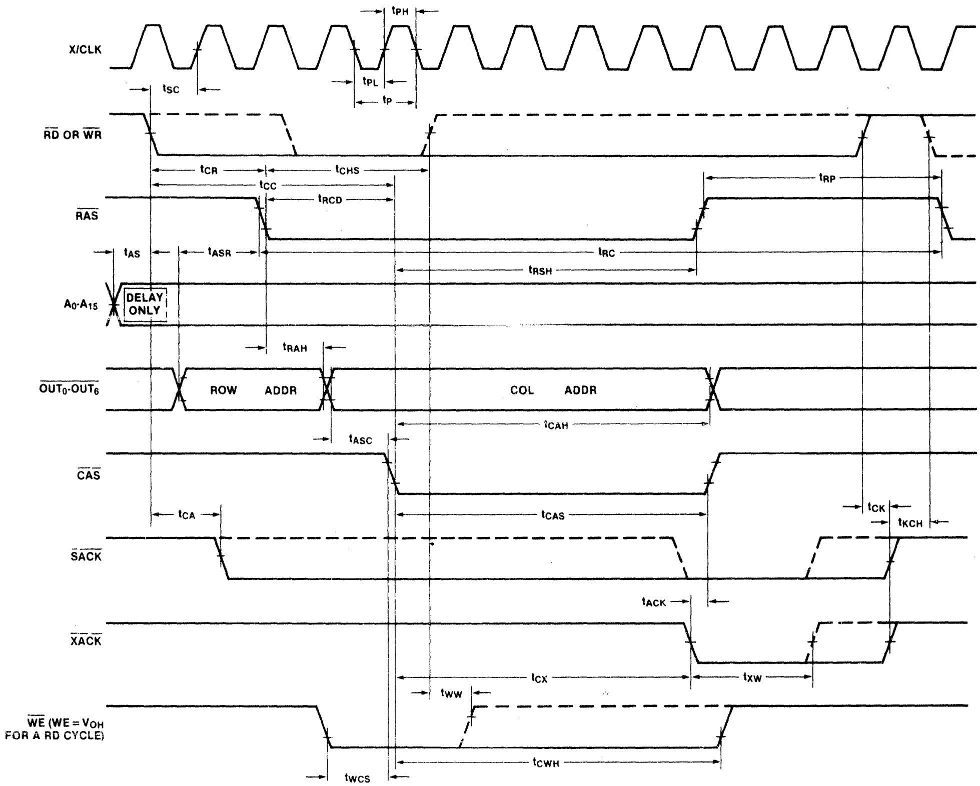


Figure 5C1.2. 8202 Timing Information

A.C. CHARACTERISTICS

$T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$

Measurements made with respect to $RAS_1 - RAS_4$, CAS, WE, $OUT_0 - OUT_6$ are at 2.4V and 0.8V. All other pins are measured at 1.5V.

Loading:

64 Devices

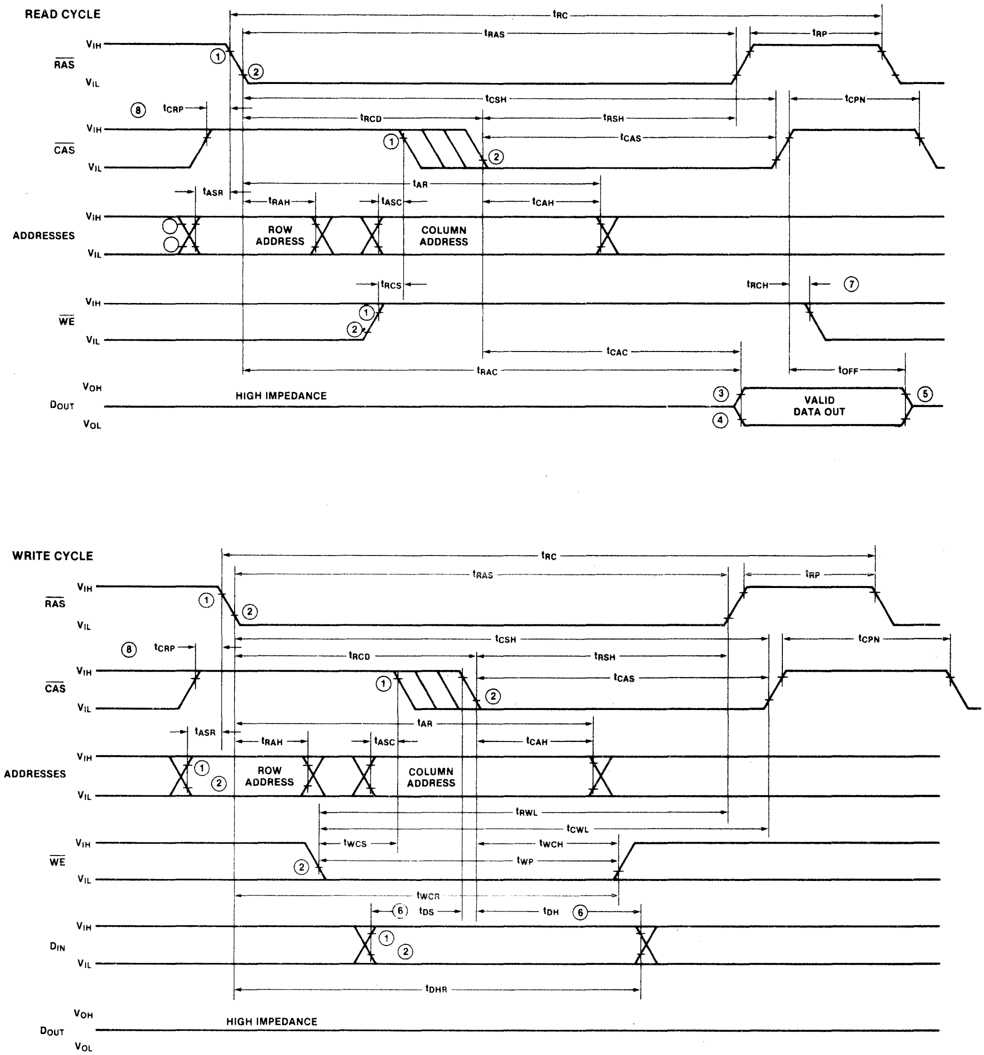
$\overline{SACK}$, $\overline{XACK}$	CL = 30 pF
$\overline{OUT_0} - \overline{OUT_6}$	CL = 320 pF
$\overline{RAS_1} - \overline{RAS_4}$	CL = 230 pF
$\overline{WE}$	CL = 450 pF
$\overline{CAS}$	CL = 640 pF

Symbol	Parameter	Min	Max	Units
t_P	Clock (Internal/External) Period (See Note 1)	40	54	ns
t_{RC}	Memory Cycle Time	$10\,t_P - 30$	$12\,t_P$	ns
t_{RAH}	Row Address Hold Time	$t_P - 10$		ns
t_{ASR}	Row Address Setup Time	t_{PH}		ns
t_{CAH}	Column Address Hold Time	$5\,t_P$		ns
t_{ASC}	Column Address Setup Time	$t_P - 35$		ns
t_{RCD}	$\overline{RAS}$ to $\overline{CAS}$ Delay Time	$2\,t_P - 10$	$2\,t_P + 45$	ns
t_{WCS}	$\overline{WE}$ Setup to $\overline{CAS}$	$t_P - 40$		ns
t_{RSH}	$\overline{RAS}$ Hold Time	$5\,t_P - 30$		ns
t_{CAS}	$\overline{CAS}$ Pulse Width	$5\,t_P - 30$		ns
t_{RP}	$\overline{RAS}$ Precharge Time (See Note 2)	$4\,t_P - 30$		ns
t_{WCH}	$\overline{WE}$ Hold Time to $\overline{CAS}$	$5\,t_P - 35$		ns
t_{REF}	Internally Generated Refresh to Refresh Time 64 Cycle 128 Cycle	 $548\,t_P$ $264\,t_P$	 $576\,t_P$ $288\,t_P$	 ns ns
t_{CR}	$\overline{RD}$, $\overline{WR}$ to $\overline{RAS}$ Delay	$t_{PH} + 30$	$t_{PH} + t_P + 75$	ns
t_{CC}	$\overline{RD}$, $\overline{WR}$ to $\overline{CAS}$ Delay	$t_{PH} + 2\,t_P + 25$	$t_{PH} + 3\,t_P + 100$	ns
t_{RFR}	REFRQ to $\overline{RAS}$ Delay	$1.5\,t_P + 30$	$2.5\,t_P + 100$	ns
t_{AS}	A_0—A_{15} to $\overline{RD}$, $\overline{WR}$ Setup Time (See Note 4)	0		ns
t_{CA}	$\overline{RD}$, $\overline{WR}$ to $\overline{SACK}$ Leading Edge		$t_P + 40$	ns
t_{CK}	$\overline{RD}$, $\overline{WR}$ to $\overline{XACK}$, $\overline{SACK}$ Trailing Edge Delay		30	ns
t_{KCH}	$\overline{RD}$, $\overline{WR}$ Inactive Hold to $\overline{SACK}$ Trailing Edge	10		ns
t_{SC}	$\overline{RD}$, $\overline{WR}$, $\overline{PCS}$ to X/CLK Setup Time (See Note 3)	15		ns
t_{CX}	$\overline{CAS}$ to $\overline{XACK}$ Time	$5\,t_P - 40$	$5\,t_P + 20$	ns
t_{ACK}	$\overline{XACK}$ Leading Edge to $\overline{CAS}$ Trailing Edge Time	10		ns
t_{XW}	$\overline{XACK}$ Pulse Width	$2\,t_P - 25$		ns
t_{LL}	REFRQ Pulse Width	20		ns
t_{CHS}	$\overline{RD}$, $\overline{WR}$, $\overline{PCS}$ Active Hold to $\overline{RAS}$	0		ns
t_{WW}	$\overline{WR}$ to $\overline{WE}$ Propagation Delay	8	50	ns
t_{AL}	S_1 to ALE Setup Time	40		ns
t_{LA}	S_1 to ALE Hold Time	$2\,t_P + 40$		ns
t_{PL}	External Clock Low Time	15		ns
t_{PH}	External Clock High Time	22		ns
t_{PH}	External Clock High Time for $V_{CC} = 5V \pm 5\%$	18		ns

Notes:

1. t_P minimum determines maximum oscillator frequency.
   t_P maximum determines minimum frequency to maintain 2 ms refresh rate and t_{RP} minimum.
2. To achieve the minimum time between the $\overline{RAS}$ of a memory cycle and the $\overline{RAS}$ of a refresh cycle, such as a transparent refresh, REFRQ should be pulsed in the previous memory cycle.
3. t_{SC} is not required for proper operation which is in agreement with the other specs, but can be used to synchronize external signals with X/CLK if it is desired.
4. If t_{AS} is less than 0 then the only impact is that t_{ASR} decreases by a corresponding amount.

Figure 5C1.2. 8202 Timing Information (Con't)



NOTES:
- 1,2. $V_{IH\ MIN}$ AND $V_{IL\ MAX}$ ARE REFERENCE LEVELS FOR MEASURING TIMING OF INPUT SIGNALS.
- 3,4. $V_{OH\ MIN}$ AND $V_{OL\ MAX}$ ARE REFERENCE LEVELS FOR MEASURING TIMING OF D_{OUT}.
- 5. t_{OFF} IS MEASURED TO $I_{OUT} < |I_{OL}|$.
- 6. t_{DS} AND t_{DH} ARE REFERENCED TO $\overline{CAS}$ OR $\overline{WE}$, WHICHEVER OCCURS LAST.
- ‡. t_{RCH} IS REFERENCED TO THE TRAILING EDGE OF $\overline{CAS}$ OR $\overline{RAS}$, WHICHEVER OCCURS FIRST.
- 8. t_{CRP} REQUIREMENT IS ONLY APPLICABLE FOR $\overline{RAD}/\overline{CAS}$ CYCLES PRECEDED BY A $\overline{CAS}$-ONLY CYCLE (i.e., FOR SYSTEMS WHERE $\overline{CAS}$ HAS NOT BEEN DECODED WITH $\overline{RAS}$).

Figure 5C1.3. 2118 Family Timing

A.C. CHARACTERISTICS[1,2,3]

$T_A = 0°C$ to $70°C$, $V_{DD} = 5V \pm 10\%$, $V_{SS} = 0V$, unless otherwise noted.

READ, WRITE, READ-MODIFY-WRITE AND REFRESH CYCLES

Symbol	Parameter	2118-3		2118-4		2118-7		Unit	Notes
		Min.	Max.	Min.	Max.	Min.	Max.		
t_{RAC}	Access Time From $\overline{RAS}$		100		120		150	ns	4,5
t_{CAC}	Access Time from $\overline{CAS}$		55		65		80	ns	4,5,6
t_{REF}	Time Between Refresh		2		2		2	ms	
t_{RP}	$\overline{RAS}$ Precharge Time	110		120		135		ns	
t_{CPN}	$\overline{CAS}$ Precharge Time (non-page cycles)	50		55		70		ns	
t_{CRP}	$\overline{CAS}$ to $\overline{RAS}$ Precharge Time	0		0		0		ns	
t_{RCD}	$\overline{RAS}$ to $\overline{CAS}$ Delay Time	25	45	25	55	25	70	ns	7
t_{RSH}	$\overline{RAS}$ Hold Time	70		85		105		ns	
t_{CSH}	$\overline{CAS}$ Hold Time	100		120		165		ns	
t_{ASR}	Row Address Set-Up Time	0		0		0		ns	
t_{RAH}	Row Address Hold Time	15		15		15		ns	
t_{ASC}	Column Address Set-Up Time	0		0		0		ns	
t_{CAH}	Column Address Hold Time	15		15		20		ns	
t_{AR}	Column Address Hold Time to $\overline{RAS}$	60		70		90		ns	
t_T	Transition Time (Rise and Fall)	3	50	3	50	3	50	ns	8
t_{OFF}	Output Buffer Turn Off Delay	0	45	0	50	0	60	ns	
READ AND REFRESH CYCLES									
T_{RC}	Random Read Cycle Time	235		270		320		ns	
t_{RAS}	$\overline{RAS}$ Pulse Width	115	10000	140	10000	175	10000	ns	
t_{CAS}	$\overline{CAS}$ Pulse Width	55	10000	65	10000	95	10000	ns	
t_{RCS}	Read Command Set-Up Time	0		0		0		ns	
t_{RCH}	Read Command Hold Time	0		0		0		ns	
WRITE CYCLE									
t_{RC}	Random Write Cycle Time	235		270		320		ns	
t_{RAS}	$\overline{RAS}$ Pulse Width	115	10000	140	10000	175	10000	ns	
t_{CAS}	$\overline{CAS}$ Pulse Width	55	10000	65	10000	95	10000	ns	
t_{WCS}	Write Command Set-Up Time	0		0		0		ns	9
t_{WCH}	Write Command Hold Time	25		30		45		ns	
t_{WCR}	Write Command Hold Time, to $\overline{RAS}$	70		85		115		ns	
t_{WP}	Write Command Pulse Width	25		30		50		ns	
t_{RWL}	Write Command to $\overline{RAS}$ Lead Time	60		65		110		ns	
t_{CWL}	Write Command to $\overline{CAS}$ Lead Time	45		50		100		ns	
t_{DS}	Data-In Set-Up Time	0		0		0		ns	
t_{DH}	Data-In Hold Time	25		30		45		ns	
t_{DHR}	Data-In Hold Time, to $\overline{RAS}$	70		85		115		ns	
READ-MODIFY-WRITE CYCLE									
t_{RWC}	Read-Modify-Write Cycle Time	285		320		410		ns	
t_{RRW}	RMW Cycle $\overline{RAS}$ Pulse Width	165	10000	190	10000	265	10000	ns	
t_{CRW}	RMW Cycle $\overline{CAS}$ Pulse Width	105	10000	120	10000	185	10000	ns	
t_{RWD}	$\overline{RAS}$ to $\overline{WE}$ Delay	100		120		150		ns	9
t_{CWD}	$\overline{CAS}$ to $\overline{WE}$ Delay	55		65		80		ns	9

NOTES:
1. All voltages referenced to V_{SS}.
2. Eight cycles are required after power-up or prolonged periods (greater than 2 ms) of $\overline{RAS}$ inactivity before proper device operation is achieved. Any 8 cycles which perform refresh are adequate for this purpose.
3. A.C. Characteristics assume $t_T = 5$ ns.
4. Assume that $t_{RCD} \leq t_{RCD}$ (max.). If t_{RCD} is greater than t_{RCD} (max.) then t_{RAC} will increase by the amount that t_{RCD} exceeds t_{RCD} (max.).
5. Load = 2 TTL loads and 100 pF.
6. Assumes $t_{RCD} \geq t_{RCD}$ (max.).
7. t_{RCD} (max.) is specified as a reference point only; if t_{RCD} is less than t_{RCD} (max.) access time is t_{RAC}, if t_{RCD} is greater than t_{RCD} (max.) access time is $t_{RCD} + t_{CAC}$.
8. t_T is measured between V_{IH} (min.) and V_{IL} (max.).
9. t_{WCS}, t_{CWD} and t_{RWD} are specified as reference points only. If $t_{WCS} \geq t_{WCS}$ (min.) the cycle is an early write cycle and the data out pin will remain high impedance throughout the entire cycle. If $t_{CWD} \geq t_{CWD}$ (min.) and $t_{RWD} \geq t_{RWD}$ (min.), the cycle is a read-modify-write cycle and the data out will contain the data read from the selected address. If neither of the above conditions is satisfied, the condition of the data out is indeterminate.

Figure 5C1.3. 2118 Family Timing (Con't)

5.C.2 Enhanced Operation

Two problems are evident from the previous investigation:

1) $\overline{SACK}$ timing from command will not allow reliable operation while $\overline{XACK}$ is not active early enough to prevent wait states.

2) The normal write command required to guarantee data setup is not enabled until the CPU has sampled READY thereby forcing multiple wait states during write operations.

The first problem could be resolved if an early command could be generated that would guarantee $\overline{SACK}$ was valid when READY was sampled and $\overline{SACK}$ to data valid satisfied the CPU requirements. Figure 5.C.2.1 is a circuit which provides an early read command derived from the maximum mode status. The early command is enabled from the trailing edge of ALE and disabled on the trailing edge of the normal command. The command provides an additional TCHCLmin − TCHLLmax + TCLMLmax − circuit delays = 53 ns of access time and time to generate RDY from the early command. If we go back to our previous equations, early command to valid data at the CPU is now:

TCHCLmin − TCHLLmax + 2TCLCL − TDVCLmax − buffer and circuit delays = 333 ns

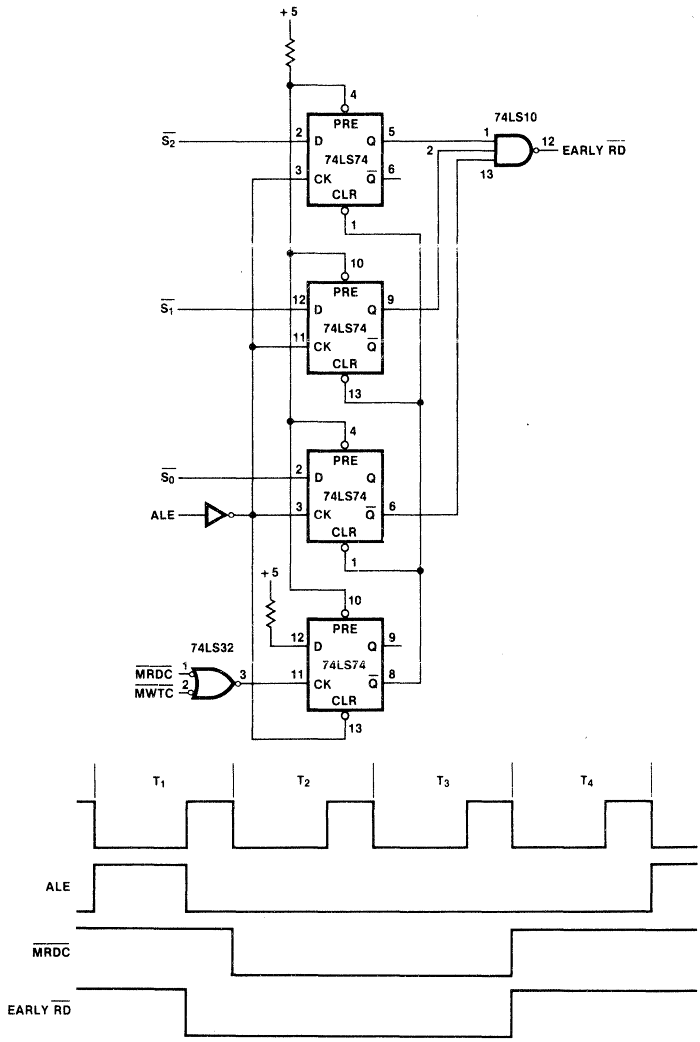


Figure 5C2.1. Early Read and Write Command Generation

We can now use the slowest 2118 which gives 8202 and 2118 access of 320 ns. Early command to RDY timing is TCLCL − TCHLLmax − circuit delays − TR1VCLmax = 115 ns and provides 35 ns of margin beyond the 8202 command to $\overline{SACK}$ delay.

The write timing of the 8202 and write data valid timing of the 8086 do not allow use of an early write command. However, if the 8202 clock is reduced from 25 MHz to 20 MHz and $\overline{WE}$ to the RAM's is gated with $\overline{CAS}$, the advanced write command ($\overline{AMWC}$) may be used. At 20 MHz the minimum command to $\overline{CAS}$ delay is 148 ns while the maximum data valid delay is 144 ns.

The reduced 8202 clock frequency still satisfies no wait state read operation from early read and will insert no more than one wait state for write (assuming no conflict with refresh). 20 MHz 8202 operation will however require using the 2118-4 to satisfy read access time.

Note that slowing the 8202 to 22.2 MHz guarantees valid data within 10 ns after $\overline{CAS}$ and allows using the 2118-7. Since this analysis is totally based on worst case minimum and maximum delays, the designer should evaluate the timing requirements of his specific implementation.

It should be noted that the 8202 $\overline{SACK}$ is equivalent to $\overline{XACK}$ timing if the cycle being executed was delayed by refresh. Delaying $\overline{SACK}$ until $\overline{XACK}$ time causes the CPU to enter wait states until the cycle is completed. If the cycle is a read cycle, the $\overline{XACK}$ timing guarantees data is valid at the CPU before RDY is issued to the CPU.

The use of the early command signals also solves a problem not mentioned previously. The cycle rate of the 8202 @ 20 MHz requires that commands (from leading edge to leading edge) be separated by a minimum of 695 ns. The maximum mode 8086 however may issue a read command 600 ns after the normal write command. For the early read command and advanced write command, 725 ns are guaranteed between commands.

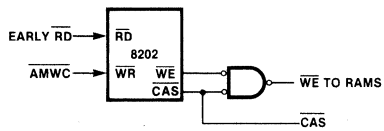


Figure 5C2.2. Delayed Write to Dynamic RAMs

APPENDIX I
BUS CONTENTION AND ITS EFFECT ON SYSTEM INTEGRITY

SYSTEM ARCHITECTURE

As higher performance microprocessors have become available, the architecture of microprocessor systems has been evolving, again placing demands on memory. For many years, system designers have been plagued with the problem of bus contention when connecting multiple memories to a common data bus. There have been various schemes for avoiding the problem, but device manufacturers have been unable to design internal circuits that would guarantee that one memory device would be "off" the bus before another device was selected. With small memories (512x8 and 1Kx8), it has been traditional to connect all the system address lines together and utilize the difference between t_{ACC} and t_{CO} to perform a decode to select the correct device (as shown in Figure 1).

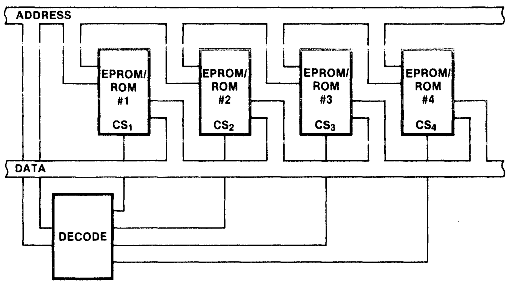


Figure 1. Single Control Line Architecture

With the 1702A, the chip select to output delay was only 100 ns shorter than the address access time; or to state it another way, the t_{ACC} time was 1000 ns while the t_{CO} time was 900 ns. The 1702A t_{ACC} performance of 1000 ns was suitable for the 4004 series microprocessors, but the 8080 processor required that the corresponding numbers be reduced to $t_{ACC} = 450$ ns and $t_{CO} = 120$ ns. This allowed a substantial improvement in performance over the 4004 series of microprocessors, but placed a substantial burden on the memory. The 2708 was developed to be compatible with the 8080 both in access time and power supply requirements. A portion of each 8080 machine cycle time had to be devoted to the architecture of the system decoding scheme used. This devoted portion of the machine cycle included the time required for the system controller (8224) to perform its function before the actual decode process could begin.

Let's pause here and examine the actual decode scheme that was used so we can understand how the control functions that a memory device requires are related to system architecture.

The 2708 can be used to illustrate the problem of having a single control line. The 2708 has only one read control function, chip select ($\overline{CS}$), which is very fast ($t_{CO} = 120$ ns) with respect to the overall access time ($t_{ACC} = 450$ ns) of the 2708. It is this time difference (330 ns) that is used to perform the decode function, as illustrated in Figure 2. The scheme works well and does not limit system performance, but it does lead to the possibility of bus contention.

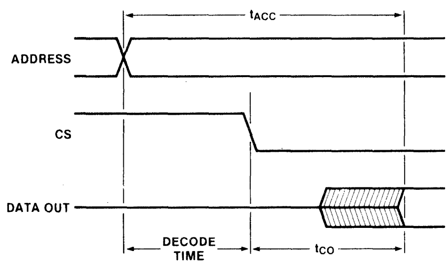
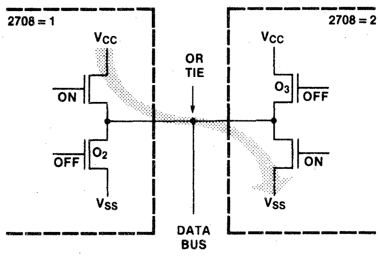


Figure 2. Single Line Control Architecture

BUS CONTENTION

There are actually two problems with the scheme described in the previous section. First, if one device in a multiple memory system has a relatively long deselect time, and a relatively fast decoder is used, it would be possible to have another device selected at the same time. If the two devices thus selected were reading opposite data; that is, device number one reading a HIGH and device number two reading a LOW, the output transistors of the two memory devices would effectively produce a short circuit, as Figure 3 illustrates. In this case, the current path is from V_{CC} on device number one to GND on device number two. This current is limited only by the "on" impedance of the MOS output transistors and can reach levels in excess of 200 mA per device. If the MOS transistors have a lot of "extra" margin, the current is usually not destructive; however, an instantaneous load of 400 mA can produce "glitches" on the V_{CC} supply—glitches large enough to cause standard TTL devices to drop bits or otherwise malfunction, thus causing incorrect address decode or generation.

The second problem with a single control line scheme is more subtle. As previously mentioned, there is only one control function available on the 2708 and any decoding scheme must use it out of necessity. In addition, any inadvertent changes in the state of the high order address lines that are inputs to the decoder will cause a change in the device that is selected. The result is the same as before—bus contention, only from a different source. The deselected device cannot get "off" the bus before the selected one is "on" the bus as the addresses rapidly change state. One approach to solving this problem would be to design (and specify as a maximum) devices with t_{DF} time less than t_{CO} time, thereby assuring that if one device is selected while another is simultaneously being deselected, there would be some small (20 ns) margin. Even with this solution, the user would not be protected from devices which have very fast t_{CO} times (t_{CO} is specified as a maximum).



RESULTS OF IMPROPER TIMING WHEN OR TYING MULTIPLE MEMORIES.

Figure 3. Results of Improper Timing when OR Tying Multiple Memories

The only sure solution appears to be the use of an external bus driver/transceiver that has an independent enable function. Then that function, not the "device selecting function," or addresses, could control the flow of data "on" and "off" the bus, and any contention problems would be confined to a particular card or area of a large card. In fact, many systems are implemented that way—the use of bus drivers is not at all uncommon in large systems where the drive requirements of long, highly capacitive interconnecting lines must be taken into consideration—it also may be the reason why more system designers were not aware of the bus contention problem until they took a previously large (multicard) system and, using an advanced micorprocessor and higher density memory devices, combined them all on one card, thereby eliminating the requirement for the bus drivers, but experiencing the problem of bus contention as described above.

THE MICROPROCESSOR/MEMORY INTERFACE

From the foregoing discussion, it becomes clear that some new concepts, both with regard to architecture and performance are required. A new generation of two control line devices is called for with general requirements as listed below:

1. Capability to control the data "on" and "off" the system bus, independent of the device selecting function identified above.

2. Access time compatible with the high performance microprocessors that are currently available.

Now let's examine the system architecture that is required to implement the two line control and prevent bus contention. This is shown in the form of a timing diagram (Figure 4). As before, addresses are used to generate the unique device selecting function, but a separate and independent Output Enable (OE) control is now used to gate data "on" and "off" the system data bus. With this scheme, bus contention is completely eliminated as the processor determines the time during which data must be present on the bus and then releases the bus by way of the Output Enable line, thus freeing the bus for use by other devices, either memories or peripheral devices. This type of architecture can be easily accomplished if the memory devices have two control functions, and the system is implemented according to the block diagram shown in Figure 5. It differs from the previous block diagram (shown in Figure 1) in that the control bus, which is connected to all memory Output Enable pins, provides separate and independent control over the data bus. In this way, the microprocessor is always in control of the system; while in the previous system, the microprocessor passed control to the particular memory device and then waited for data to become available. Another way to look at it is, with a single control line the sytem is always asynchronous with respect to microprocessor/memory communications. By using two control lines, the memory is synchronized to the processor.

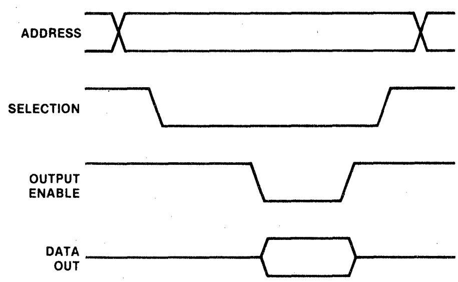


Figure 4. Two Control Line Architecture

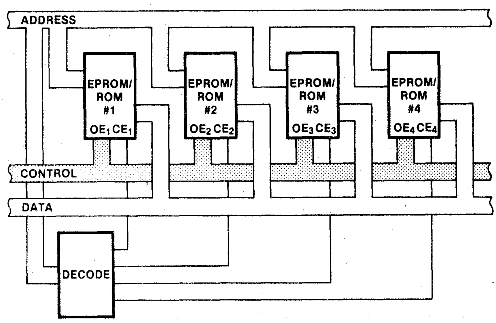


Figure 5. Two Control Line Architecture

intel®

APPLICATION NOTE

AP-113

February 1981

Getting Started With the Numeric Data Processor

Bill Rash
Microcomputer Applications

207865-001

INTRODUCTION

This is an application note on using numerics in Intel's iAPX 86 or iAPX 88 microprocessor family. The numerics implemented in the family provide instruction level support for high-precision integer and floating point data types with arithmetic operations like add, subtract, multiply, divide, square root, power, log and trigonometrics. These features are provided by members of the iAPX 86 or iAPX 88 family called numeric data processors.

Rather than concentrate on a narrow, specific application, the topics covered in this application note were chosen for generality across many applications. The goal is to provide sufficient background information so that software and hardware engineers can quickly move beyond needs specific to the numeric data processor and concentrate on the special needs of their application. The material is structured to allow quick identification of relevant material without reading all the material leading up to that point. Everyone should read the introduction to establish terminology and a basic background.

iAPX 86,88 BASE

The numeric data processor is based on an 8088 or 8086 microprocessor. The 8086 and 8088 are general purpose microprocessors, designed for general data processing applications. General applications need fast, efficient data movement and program control instructions. Actual arithmetic on data values is simple in general applications. The 8086 and 8088 fulfill these needs in a low cost, effective manner.

However, some applications need more powerful arithmetic instructions and data types than a general purpose data processor provides. The real world deals in fractional values and requires arithmetic operations like square root, sine, and logarithms. Integer data types and their operations like add, subtract, multiply, and divide may not meet the needs for accuracy, speed, and ease of use.

Such functions are not simple or inexpensive. The general data processor does not provide these features due to their cost to other less-complex applications that do not need such features. A special processor is required, one which is easy to use and has a high level of support in hardware and software.

The numeric data processor provides these features. It supports the data types and operations needed and allows use of all the current hardware and software support for the iAPX 86/10 and 88/10 microprocessors.

The iAPX 86 and iAPX 88 provide two implementations of a numeric data processor. Each offers different tradeoffs in performance, memory size, and cost.

One alternative uses a special hardware component, the 8087 numeric processor extension, while the other is based on software, the 8087 emulator. Both component and software emulator add the extra numerics data types and operations to the 8086 or 8088.

The component and its software emulator are completely compatible.

Nomenclature

Table one shows several possible configurations of the iAPX 86 and iAPX 88 microprocessor family. The choice of configuration will be decided by the needs of the application for cost and performance in the areas of general data processing, numerics, and I/O processing. The combination of an 8086 or 8088 with an 8087 is called an iAPX 86/20 or 88/20 numeric data processor. For applications requiring high I/O bandwidths and numeric performance, a combination of 8086, 8087 and 8089 is an iAPX 86/21 numerics and I/O data processor. The same system with an 8088 CPU for smaller size and lower cost, due to the smaller 8-bit wide system data bus, is referred to as an iAPX 88/21. Each 8089 in the system is designated in the units digit of the system designation. The term 86/2X or 88/2X refers to a numeric data processor with any number of 8089s.

Throughout this application note, I will use the terms NDP, numeric data processor, 86/2X, and 88/2X synonymously. Numeric processor extension and NPX are also synonymous for the functions of either the 8087 component or 8087 emulator. The term numeric instruction or numeric data type refers to an instruction or data type made available by the NPX. The term host will refer to either the 8086 or 8088 microprocessor.

Table 1. Components Used in i/APX 86,88 Configurations

System Name	8086	8087	8088	8089
iAPX 86/10	1			
iAPX 86/11	1			1
iAPX 86/12	1			2
iAPX 86/20	1	1		
iAPX 86/21	1	1		1
iAPX 86/22	1	1		2
iAPX 88/10			1	
iAPX 88/11			1	1
iAPX 88/12			1	2
iAPX 88/20		1	1	
iAPX 88/21		1	1	1
iAPX 88/22		1	1	2

NPX OVERVIEW

The 8087 is a coprocessor extension available to iAPX 86/1X or iAPX 88/1X maximum mode microprocessor systems. (See page 7). The 8087 adds hardware support for floating point and extended precision integer data types, registers, and instructions. Figure 1 shows the register set available to the NDP. On the next page, the seven data types available to numeric instructions are listed (Fig 2). Each data type has a load and store instruction. Independent of whether an 8087 or its emulator are used, the registers and data types all appear the same to the programmer.

All the numeric instructions and data types of the NPX are used by the programmer in the same manner as the general data types and instructions of the host.

The numeric data formats and arithmetic operations provided by the 8087 conform to the proposed IEEE Microprocessor Floating Point Standard. All the proposed IEEE floating point standard algorithms, exception detection, exception handling, infinity arithmetic and rounding controls are implemented.[1]

The numeric registers of the NPX are provided for fast, easy reference to values needed in numeric calculations. All numeric values kept in the NPX register file are held in the 80-bit temporary real floating point format which is the same as the 80-bit temporary real data type.

All data types are converted to the 80-bit register file format when used by the NPX. Load and store instructions automatically convert between the memory operand data type and the register file format for all numeric data types. The numeric load instruction specifies the format in which the memory operand is expected and which addressing mode to use.

All host base registers, index registers, segment registers, and addressing modes are available for locating numeric operands. In the same manner, the store instruction also specifies which data type to use and where the value is located when stored into memory.

Selecting Numeric Data Types

As figure 2 shows, the numeric data types are of different lengths and domains (real or integer). Each numeric data type is provided for a specific function, they are:

16-bit word integers	—Index values, loop counts, and small program control values
32-bit short integers	—Large integer general computation
64-bit long integers	—Extended range integer computation
18-digit packed decimal	—Commercial and decimal conversion arithmetic
32-bit short real	—Reduced range and accuracy is traded for reduced memory requirements
64-bit long real	—Recommended floating point variable type
80-bit temporary real	—Format for intermediate or high precision calculations

Referencing memory data types in the NDP is not restricted to load and store instructions. Some arithmetic operations can specify a memory operand in one of four possible data types. The numeric instructions compare, add, subtract, subtract reversed, multiply, divide, and divide reversed can specify a memory operand to be either a 16-bit integer, 32-bit integer, 32-bit real, or 64-bit real value. As with the load and store operations, the arithmetic instruction specifies the address and expected format of the memory operand.

The remaining arithmetic operations: square root, modulus, tangent, arctangent, logarithm, exponentiate, scale power, and extract power use only register operands.

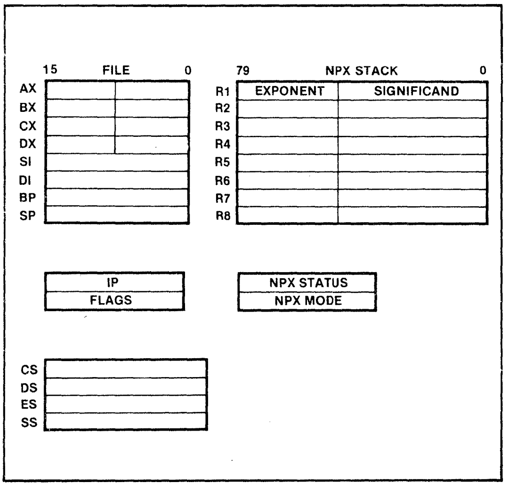


Figure 1. NDP Register Set for iAPX 86/20, 88/20

[1]"An Implementation Guide to a Proposed Standard for Floating Point" by Jerome Coonen in *Computer*, Jan. 1980 or the Oct. 1979 issue of *ACM SIGNUM*, for more information on the standard.

The register set of the host and 8087 are in separate components. Direct transfer of values between the two register sets in one instruction is not possible. To transfer values between the host and numeric register sets, the value must first pass through memory. The memory format of a 16-bit short integer used by the NPX is identical to that of the host, ensuring fast, easy transfers.

Since an 8086 or 8088 does not provide single instruction support for the remaining numeric data types, host programs reading or writing these data types must conform to the bit and byte ordering established by the NPX.

Writing programs using numeric instructions is as simple as with the host's instructions. The numeric instructions are simply placed in line with the host's instructions. They are executed in the same order as they appear in the instruction stream. Numeric instructions follow the same form as the host instructions. Figure 2 shows the ASM 86/88 representations for different numeric instructions and their similarity to host instructions.

```
FILD     VALUE
FIADD    TABLE [BX]
FADD     ST,ST(1)
```

8087 EMULATOR OVERVIEW

The NDP has two basic implementations, an 8087 component or with its software emulator (E8087). The decision to use the emulator or component has no effect on programs at the source level. At the source level, all instructions, data types, and features are used the same way.

The emulator requires all numeric instruction opcodes to be replaced with an interrupt instruction. This replacement is performed by the LINK86 program. Interrupt vectors in the host's interrupt vector table will point to numeric instruction emulation routines in the 8087 software emulator.

When using the 8087 emulator, the linker changes all the 2-byte wait-escape, nop-escape, wait-segment override, or nop-segment override sequences generated by an assembler or compiler for the 8087 component with a 2-byte interrupt instruction. Any remaining bytes of the numeric instruction are left unchanged.

DATA FORMATS	RANGE	PRECISION	MOST SIGNIFICANT BYTE (7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0)	
WORD INTEGER	10^4	16 BITS	I_{15} … I_0	TWO'S COMPLEMENT
SHORT INTEGER	10^9	32 BITS	I_{31} … I_0	TWO'S COMPLEMENT
LONG INTEGER	10^{19}	64 BITS	I_{63} … I_0	TWO'S COMPLEMENT
PACKED BCD	10^{18}	18 DIGITS	S | — | D_{17} D_{16} … D_1 D_0	
SHORT REAL	$10^{\pm 38}$	24 BITS	S | E_7 E_0 | F_1 … F_{23}	F_0 IMPLICIT
LONG REAL	$10^{\pm 308}$	53 BITS	S | E_{10} E_0 | F_1 … F_{52}	F_0 IMPLICIT
TEMPORARY REAL	$10^{\pm 4932}$	64 BITS	S | E_{14} E_0 | F_0 … F_{63}	

INTEGER: 1
PACKED BCD: $(-1)^S(D_{17} \ldots D_0)$
REAL: $(-1)^S(2^{E-BIAS})(F_0 \cdot F_1 \ldots)$
BIAS = 127 FOR SHORT REAL
1023 FOR LONG REAL
16383 FOR TEMP REAL

Figure 2. NPX Data Types

When the host encounters numeric and emulated instruction, it will execute the software interrupt instruction formed by the linker. The interrupt vector table will direct the host to the proper entry point in the 8087 emulator. Using the interrupt return address and CPU register set, the host will decode any remaining part of the numeric instruction, perform the indicated operation, then return to the next instruction following the emulated numeric instruction.

One copy of the 8087 emulator can be shared by all programs in the host.

The decision to use the 8087 or software emulator is made at link time, when all software modules are brought together. Depending on whether an 8087 or its software emulator is used, a different group of library modules are included for linking with the program.

If the 8087 component is used, the libraries do not add any code to the program, they just satisfy external references made by the assembler or compiler. Using the emulator will not increase the size of individual modules; however, other modules requiring about 16K bytes that implement the emulator will be automatically added.

Selecting between the emulator or the 8087 can be very easy. Different versions of submit files performing the link operation can be used to specify the different set of library modules needed. Figure 3 shows an example of two different submit files for the same program using the NPX with an 8087 or the 8087 emulator.

```
8087 BASED LINK/LOCATE COMMANDS

LINK86 :F1:PROG.OBJ, IO.LIB, 8087.LIB TO
       :F1:PROG.LNK
LOC86  :F1:PROG.LNK TO :F1:PROG

SOFTWARE EMULATOR BASED
LINK/LOCATE COMMANDS

LINK86 :F1:PROG.OBJ, IO.LIB, E8087.LIB,
        E8087 TO :F1:PROG.LNK
LOC86  :F1:PROG.LNK TO :F1:PROG
```

Figure 3. Submit File Example

iSBC 337™ MULTIMODULE™ Overview

The benefits of the NPX are not limited to systems which left board space for the 8087 component or memory space for its software emulator. Any maximum mode iAPX 86/1X or iAPX 88/1X system can be upgraded to a numeric processor. The iSBC 337 MULTIMODULE is designed for just this function. The iSBC 337 provides a socket for the host microprocessor and an 8087. A 40-pin plug is provided on the underside of the 337 to plug into the original host's socket, as shown in Figure 4. Two other pins on the underside of the MULTIMODULE allow easy connection to the 8087 INT and RQ/GT1 pins.

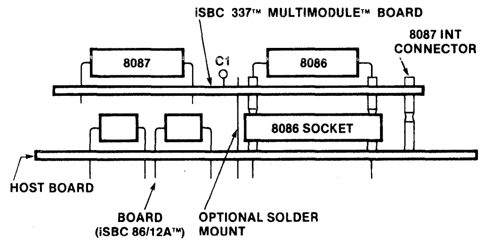


Figure 4. MULTIMODULE™ Math Mounting Scheme

CONSTRUCTING AN iAPX 86/2X OR iAPX 88/2X SYSTEM

This section will describe how to design a microprocessor system with the 8087 component. The discussion will center around hardware issues. However, some of the hardware decisions must be made based upon how the software will use the NPX. To better understand how the 8087 operates as a local bus master, we shall cover how the coprocessor interface works later in this section.

Wiring up the 8087

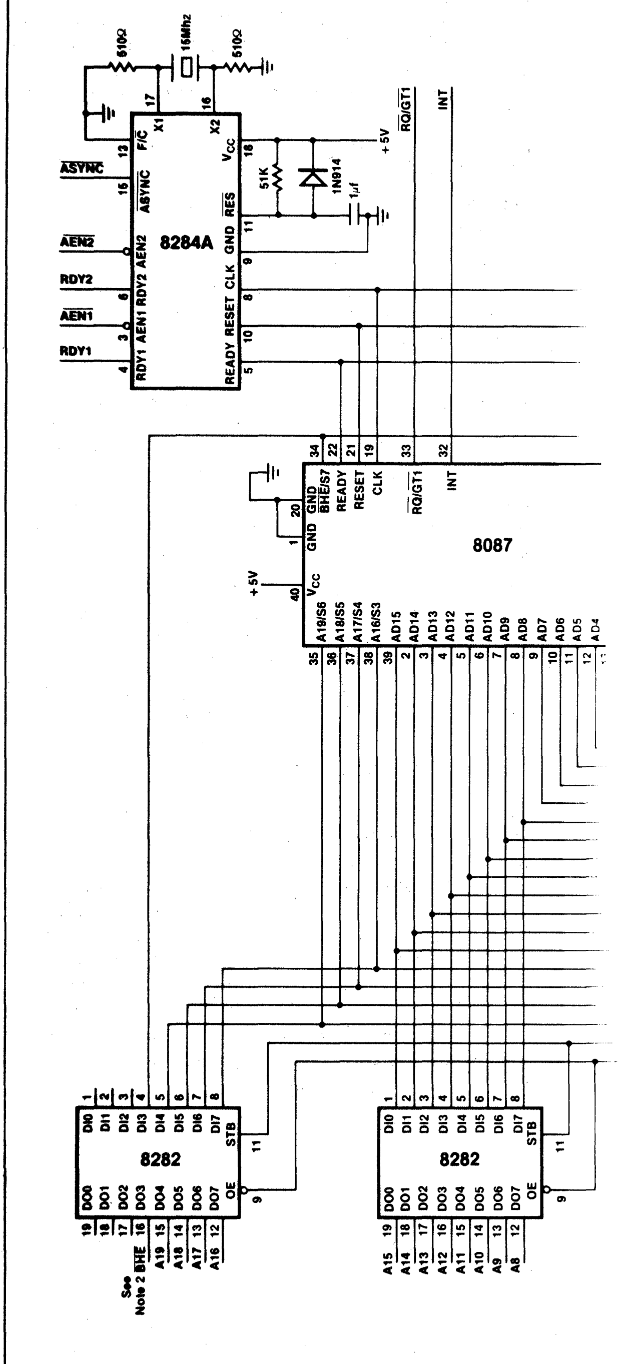
The 8087 can be designed into any 86/1X or 88/1X system operating in maximum mode. Such a system would be designated an 86/2X or 88/2X. Figure 5 shows the local bus interconnections for an iAPX 86/20 (or iAPX 88/20) system. The 8087 shares the maximum mode host's multiplexed address/data bus, status signals, queue status signals, ready status signal, clock and reset signal. Two dedicated signals, BUSY and INT, inform the host of current 8087 status. The 10K pull-down resistor on the BUSY signal ensures the host will always see a "not busy" status if an 8087 is not installed.

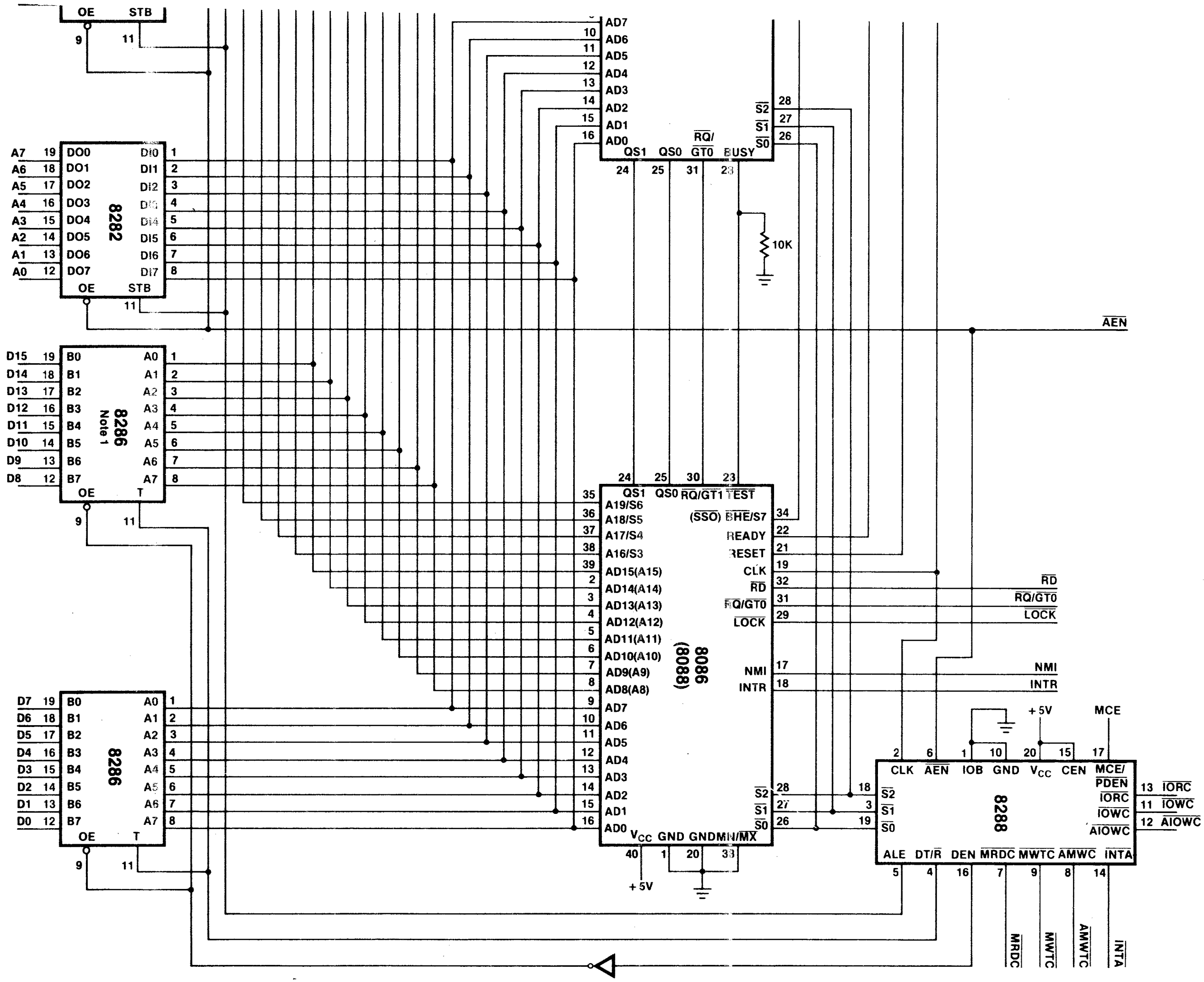
Adding the 8087 to your design has a minor effect on hardware timing. The 8087 has the exact same timing and equivalent DC and AC drive characteristics as a host or IOP on the local bus. All the local bus logic, such as clock, ready, and interface logic is shared.

The 8087 adds 15 pF to the total capacitive loading on the shared address/data and status signals. Like the 8086 or 8088, the 8087 can drive a total of 100 pF capacitive load above its own self load and sink 2.0 mA DC current on these pins. This AC and DC drive is sufficient for an 86/21 system with two sets of data transceivers, address latches, and bus controllers for two separate busses, an on-board bus and an off-board MULTIBUS™ using the 8289 bus arbiter.

Later in this section, what to do with the 8087 INT and RQ/GT pins, is covered.

It is possible to leave a prewired 40-pin socket on the board for the 8087. Adding the 8087 to such a system is as easy as just plugging it in. If a program attempts to execute any numeric instructions without the 8087 installed, they will be simply treated as NOP instructions by the host. Software can test for the existence of the 8087 by initializing it and then storing the control word. The program of Figure 6 illustrates this technique.





Note 1: Data Transceiver not present in 88/21 system
Note 2: BHE signal not necessary in 88/21 system

WHAT IS THE iAPX 86, 88 COPROCESSOR INTERFACE?

The idea of a coprocessor is based on the observation that hardware specially designed for a function is the fastest, smallest, and cheapest implementation. But, it is too expensive to incorporate all desired functions in general purpose hardware. Few applications could use all the functions. To build fast, small, economical systems, we need some way to mix and match components supporting specialized functions.

Purpose of the Coprocessor Interface

The coprocessor interface of the general purpose 8086 or 8088 microprocessor provides a way to attach specialized hardware in a simple, elegant, and efficient manner. Because the coprocessor hardware is specialized, it can perform its job much faster than any general purpose CPU of similar size and cost. The coprocessor interface simply requires connection to the host's local address/data, status, clock, ready, reset, test and request/grant signals. Being attached to the host's local bus gives the coprocessor access to all memory and I/O resources available to the host.

The coprocessor is independent of system configuration. Using the local bus as the connection point to the host isolates the coprocessor from the particular system configuration, since the timing and function of local bus signals are fixed.

Software's View of the Coprocessor

The coprocessor interface allows specialized hardware to appear as an integral part of the host's architecture controlled by the host with special instructions. When the host encounters these special instructions, both the host and coprocessor recognize them and work together to perform the desired function. No status polling loops or command stuffing sequences are required by software to operate the coprocessor.

More information is available to a coprocessor than simply an instruction opcode and a signal to begin execution. The host's coprocessor interface can read a value from memory, or identify a region of memory the coprocessor should use while performing its function. All the addressing modes of the host are available to identify memory based operands to the coprocessor.

Concurrent Execution of Host and Coprocessor

After the coprocessor has started its operation, the host may continue on with the program, executing it in parallel while the coprocessor performs the function started earlier. The parallel operation of the coprocessor does not normally affect that of the host unless the coprocessor must reference memory or I/O-based operands. When the host releases the local bus to the coprocessor, the host may continue to execute from its internal instruction queue. However, the host must stop when it also needs the local bus currently in use by the coprocessor. Except for the stolen memory cycle, the operation of the coprocessor is transparent to the host.

This parallel operation of host and coprocessor is called concurrent execution. Concurrent execution of instructions requires less total time then a strictly sequential execution would. System performance will be higher with concurrent execution of instructions between the host and coprocessor.

SYNCHRONIZATION

In exchange for the higher system performance made available by concurrent execution, programs must provide what is called synchronization between the host and coprocessor. Synchronization is necessary whenever the host and coprocessor must use information available from the other. Synchronization involves either the host or coprocessor waiting for the other to finish an operation currently in progress. Since the host executes the program, and has program control instructions like jumps, it is given responsibility for synchronization. To meet this need, a special host instruction exists to synchronize host operation with a coprocessor.

```
; The following algorithm detects the presence of the 8087 as well as the 80287 in a system. This will make it
; easier for ISVs to port their 8086-87 software to 286-287 systems.
;
;
        cc_cr                   equ     ODH         ; carriage return
        cc_lf                   equ     OAH         ; line feed

assume  cz:code, ds:data
;
code    segment                 public
start:
        mov     ax,data                 ;set data segment
        mov     ds,ax
;
; Test if 8087 is present in PC or PC/XT, or 80287 is in PC/AT
;
        fninit                          ;initialize coprocessor
        xor     ah,ah                   ;zero ah register and memory byte
        mov     byte ptr control+1,ah
        fnstcw  control                 ;store coprocessor's control word in memory
        mov     ah,byte ptr control+1
        cmp     ah,03h                  ;upper byte of control work will be 03 if
                                        ;8087 or 80287 coprocessor is present
        jne     no_coproc
;
coproc:
        mov     ah,09h                  ;print string — coprocessor present
        mov     dx,offset msg_yes
        int     21h
        jmp          done
;
no_coproc:
        mov     ah,09h                  ;print string—coprocessor not present
        mov     dx,offset msg_no
        int     21h
;
done:
        mov     ah,4CH                  ;terminate program
        int     21h
code    ends

data    segment                 public
control dw                      00
msg_yes db      cc_cr,cc_lf,
        db      'System has an 8087 or 80287',cc_cr, cc_lf, '$'
msg_no  db      cc_cr,cc_lf,
        db      'System does not have an 8087 or 80287',cc_cr, cc_lf, '$'
data    ends
        end     start                   ;start is the entry point
```

Figure 6. Test for Existence of an 8087

The host coprocessor synchronization instruction, called "WAIT", uses the TEST pin of the host. The coprocessor can signal that it is still busy to the host via this pin. Whenever the host executes a wait instruction, it will stop program execution while the TEST input is active. When the TEST pin becomes inactive, the host will resume program execution with the next instruction following the WAIT. While waiting on the TEST pin, the host can be interrupted at 5 clock intervals; however, after the TEST pin becomes inactive, the host will immediately execute the next instruction, ignoring any pending interrupts between the WAIT and following instruction.

COPROCESSOR CONTROL

The host has the responsibility for overall program control. Coprocessor operation is initiated by special instructions encountered by the host. These instructions are called "ESCAPE" instructions. When the host encounters an ESCAPE instruction, the coprocessor is expected to perform the action indicated by the instruction. There are 576 different ESCAPE instructions, allowing the coprocessor to perform many different actions.

The host's coprocessor interface requires the coprocessor to recognize when the host has encountered an ESCAPE instruction. Whenever the host begins executing a new instruction, the coprocessor must look to see if it is an ESCAPE instruction. Since only the host fetches instructions and executes them, the coprocessor must monitor the instructions being executed by the host.

Host Queue Tracking

The host can fetch an instruction at a variable length time before the host executes the instruction. This is a characteristic of the instruction queue of an 8086 or 8088 microprocessor. An instruction queue allows prefetching instructions during times when the local bus would be otherwise idle. The end benefit is faster execution time of host instructions for a given memory bandwidth.

The host does not externally indicate which instruction it is currently executing. Instead, the host indicates when it fetches an instruction and when, some time later, an opcode byte is decoded and executed. To identify the actual instruction the host fetched from its queue, the coprocessor must also maintain an instruction stream identical to the host's.

Instructions can be fetched in byte or word increments, depending on the type of host and the destination address of jump instructions executed by the host. When the host has filled its queue, it stops prefetching instructions. Instructions are removed from the queue a byte at a time for decoding and execution. When a jump occurs, the queue is emptied. The coprocessor follows these actions in the host by monitoring the host's bus status, queue status, and data bus signals. Figure 7 shows how the bus status signals and queue status signals are encoded.

IGNORING I/O PROCESSORS

The host is not the only local bus master capable of fetching instructions. An Intel 8089 IOP can generate instruction fetches on the local bus in the course of executing a channel program in system memory. In this case, the status signals S2, S1, and S0 generated by the IOP are identical to those of the host. The coprocessor must not interpret these instruction prefetches as going to the host's instruction queue. This problem is solved with a status signal called S6. The S6 signal identifies when the local bus is being used by the host. When the host is the local bus master, $S6=0$ during T2 and T3 of the memory cycle. All other bus masters must set $S6=1$ during T2 and T3 of their instruction prefetch cycles. Any coprocessor must ignore activity on the local bus when $S6=1$.

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Function
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	Halt
1	0	0	Code Fetch
1	0	1	Read Data Memory
1	1	0	Write Data Memory
1	1	1	Idle

QS1	QS0	Host Function	Coprocessor Activity
0	0	No Operation	No Queue Activity
0	1	First Byte	Decode Opcode Byte
1	0	Empty Queue	Empty Queue
1	1	Subsequent Byte	Flush Byte or if 2nd Byte of Escape Decode it

Figure 7.

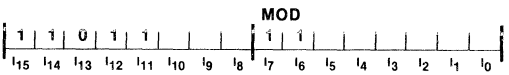
DECODING ESCAPE INSTRUCTIONS

To recognize ESCAPE instructions, the coprocessor must examine all instructions executed by the host. When the host fetches an instruction byte from its internal queue, the coprocessor must do likewise.

The queue status state, fetch opcode byte, identifies when an opcode byte is being examined by the host. At the same time, the coprocessor will check if the byte fetched from its internal instruction queue is an ESCAPE opcode. If the instruction is not an ESCAPE, the coprocessor will ignore it. The queue status signals for fetch subsequent byte and flush queue let the coprocessor track the host's queue without knowledge of the length and function of host instructions and addressing modes.

Escape Instruction Encoding

All ESCAPE instructions start with the high-order 5-bits of the instruction being 11011. They have two basic forms. The non-memory form, listed here, initiates some activity in the coprocessor using the nine available bits of the ESCAPE instruction to indicate which function to perform.



Memory reference forms of the ESCAPE instruction, shown in Figure 8, allow the host to point out a memory operand to the coprocessor using any host memory addressing mode. Six bits are available in the memory reference form to identify what to do with the memory operand. Of course, the coprocessor may not recognize all possible ESCAPE instructions, in which case it will simply ignore them.

Memory reference forms of ESCAPE instructions are identified by bits 7 and 6 of the byte following the ESCAPE opcode. These two bits are the MOD field of the 8086 or 8088 effective address calculation byte. They, together with the R/M field, bits 2 through 0, determine the addressing mode and how many subsequent bytes remain in the instruction.

Host's Response to an Escape Instruction

The host performs one of two possible actions when encountering an ESCAPE instruction: do nothing or calculate an effective address and read a word value beginning at that address. The host ignores the value of the word read. ESCAPE instructions change no registers in the host other than advancing IP. So, if there is no coprocessor, or the coprocessor ignores the ESCAPE instruction, the ESCAPE instruction is effectively a NOP to the host. Other than calculating a memory address and reading a word of memory, the host makes no other assumptions regarding coprocessor activity.

The memory reference ESCAPE instructions have two purposes: identify a memory operand and for certain instructions, transfer a word from memory to the coprocessor.

COPROCESSOR INTERFACE TO MEMORY

The design of a coprocessor is considerably simplified if it only requires reading memory values of 16 bits or less. The host can perform all the reads with the coprocessor latching the value as it appears on the data bus at the end of T3 during the memory read cycle. The coprocessor need never become a local bus master to read or write additional information.

If the coprocessor must write information to memory, or deal with data values longer than one word, then it must save the memory address and be able to become a local bus master. The read operation performed by the host in the course of executing the ESCAPE instruction places the 20-bit physical address of the operand on the address/data pins during T1 of the memory cycle. At this time the coprocessor can latch the address. If the coprocessor instruction also requires reading a value, it will appear on the data bus during T3 of the memory read. All other memory bytes are addressed relative to this starting physical address.

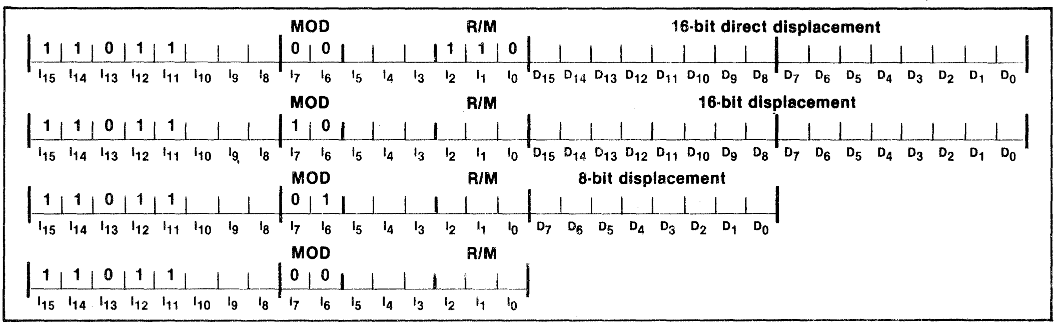


Figure 8. Memory Reference Escape Instruction Forms

Whether the coprocessor becomes a bus master or not, if the coprocessor has memory reference instruction forms, it must be able to identify the memory read performed by the host in the course of executing an ESCAPE instruction.

Identifying the memory read is straightforward, requiring all the following conditions to be met:

1) A MOD value of 00, 01, or 10 in the second byte of the ESCAPE instruction executed by the host.
2) This is the first data read memory cycle performed by the host after it encountered the ESCAPE instruction. In particular, the bus status signals S2–S0 will be 101 and S6 will be 0.

The coprocessor must continue to track the instruction queue of the host while it calculates the memory address and reads the memory value. This is simply a matter of following the fetch subsequent byte status commands occurring on the queue status pins.

HOST PROCESSOR DIFFERENCES

A coprocessor must be aware of the bus characteristics of the host processor. This determines how the host will read the word operand of a memory reference ESCAPE instruction. If the host is an 8088, it will always perform two byte reads at sequential addresses. But if the host is an 8086, it can either perform a single word read or two byte reads to sequential addresses.

The 8086 places no restrictions on the alignment of word operands in memory. It will automatically perform two byte operations for word operands starting at an odd address. The two operations are necessary since the two bytes of the operand exist in two different memory words. The coprocessor should be able to accept the two possible methods of reading a word value on the 8086.

A coprocessor can determine whether the 8086 will perform one or two memory cycles as part of the current ESCAPE instruction execution. The AD0 pin during T1 of the first memory read by the host tells if this is the only read to be performed as part of the ESCAPE instruction. If this pin is a 1 during T1 of the memory cycle, the 8086 will immediately follow this memory read cycle with another one at the next byte address.

Coprocessor Interface Summary

The host ESCAPE instructions, coprocessor interface, and WAIT instruction allow easy extension of the host's architecture with specialized processors. The 8087 is such a processor, extending the host's architecture as seen by the programmer. The specialized hardware provided by the 8087 can greatly improve system performance economically in terms of both hardware and software for numerics applications.

The next section examines how the 8087 uses the coprocessor interface of the 8086 or 8088.

8087 COPROCESSOR OPERATION

The 8086 or 8088 ESCAPE instructions provide 64 memory reference opcodes and 512 non-memory reference opcodes. The 8087 uses 57 of the memory reference forms and 406 of the non-memory reference forms. Figure 9 shows the ESCAPE instructions not used by the 8087.

I_{15}	I_{14}	I_{13}	I_{12}	I_{11}	I_{10}	I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0
1	1	0	1	1				1	1						

I_{10}	I_9	I_8	I_5	I_4	I_3	I_2	I_1	I_0	Available codes
0	0	1	0	1	0	0	0	1	1
0	0	1	0	1	0	0	1	—	2
0	0	1	0	1	0	1	—	—	4
0	0	1	1	0	0	0	1	—	2
0	0	1	1	0	0	1	1	—	2
0	0	1	1	0	1	1	1	1	1
0	0	1	1	1	0	1	0	1	1
0	0	1	1	1	1	0	1	1	1
0	0	1	1	1	1	1	1	—	2
0	1	1	1	0	0	1	0	1	1
0	1	1	1	0	0	1	1	—	2
0	1	1	1	0	1	—	—	—	8
0	1	1	1	1	—	—	—	—	16
1	0	1	1	—	—	—	—	—	32
1	1	1	1	0	0	0	0	1	1
1	1	1	1	0	0	0	1	0	1
1	1	1	1	0	0	1	—	—	4
1	1	1	1	0	1	—	—	—	8
1	1	1	1	1	—	—	—	—	16
									105 total

Available Non-Memory Reference Escape Instructions

I_{15}	I_{14}	I_{13}	I_{12}	I_{11}	I_{10}	I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0
1	1	0	1	1				MOD					R/M		

I_{10}	I_9	I_8	I_5	I_4	I_3
0	0	1	0	0	1
0	1	1	0	0	1
0	1	1	1	0	0
0	1	1	1	1	0
1	0	1	0	0	1
1	0	1	1	0	1
1	1	1	0	0	1

Available Memory Reference Escape Instructions

Figure 9.

Using the 8087 With Custom Coprocessors

Custom coprocessors, a designer may care to develop, should limit their use of ESCAPE instructions to those not used by the 8087 to prevent ambiguity about whether any one ESCAPE instruction is intended for a numerics or other custom coprocessor. Using any escape instruction for a custom coprocessor may conflict with opcodes chosen for future Intel coprocessors.

Operation of an 8087 together with other custom coprocessors is possible under the following constraints:

1) All 8087 errors are masked. The 8087 will update its opcode and instruction address registers for the unused opcodes. Unused memory reference instructions will also update the operand address value. Such changes in the 8087 make software-defined error handling impossible.
2) If the coprocessors provide a BUSY signal, they must be ORed together for connection to the host TEST pin. When the host executes a WAIT instruction, it does not know which coprocessor will be affected by the following ESCAPE instruction. In general, all coprocessors must be idle before executing the ESCAPE instruction.

Operand Addressing by the 8087

The 8087 has seven different memory operand formats. Six of them are longer than one word. All are an even number of bytes in length and are addressed by the host at the lowest address word.

When the host executes a memory reference ESCAPE instruction intended to cause a read operation by the 8087, the host always reads the low-order word of any 8087 memory operand. The 8087 will save the address and data read. To read any subsequent words of the operand, the 8087 must become a local bus master.

When the 8087 has the local bus, it increments the 20-bit physical address it saved to address the remaining words of the operand.

When the ESCAPE instruction is intended to cause a write operation by the 8087, the 8087 will save the address but ignore the data read. Eventually, it will get control of the local bus, then perform successive write, increment address operations writing the entire data value.

8087 OPERATION IN iAPX 86,88 SYSTEMS

The 8087 will work with either an 8086 or 8088 host. The identity of the host determines the width of the local bus path. The 8087 will identify the host and adjust its use of the data bus accordingly; 8 bits for an 8088 or 16 bits for an 8086. No strapping options are required by the 8087; host identification is automatic.

The 8087 identifies the host each time the host and 8087 are reset via the RESET pin. After the reset signal goes inactive, the host will begin instruction execution at memory address $FFFF0_{16}$.

If the host is an 8086 it will perform a word read at that address; an 8088 will perform a byte read.

The 8087 monitors pin 34 on the first memory cycle after power up. If an 8086 host is used, pin 34 will be the BHE signal, which will be low for that memory cycle. For an 8088 host, pin 34 will be the SS0 signal, which will be high during T1 of the first memory cycle. Based on this signal, the 8087 will then configure its data bus width to match that of the host local bus.

For 88/2X systems, pin 34 of the 8087 may be tied to V_{CC} if not connected to the 8088 SS0 pin.

The width of the data bus and alignment of data operands has no effect, except for execution time and number of memory cycles performed, on 8087 instructions. A numeric program will always produce the same results on an 86/2X or 88/2X with any operand alignment. All numeric operands have the same relative byte orderings independent of the host and starting address.

The byte alignment of memory operands can affect the performance of programs executing on an 86/2X. If a word operand, or any numeric operand, starts on an odd-byte address, more memory cycles are required to access the operand than if the operand started on an even address. The extra memory cycles will lower system performance.

The 86/2X will attempt to minimize the number of extra memory cycles required for odd-aligned operands. In these cases, the 8087 will perform first a byte operation, then a series of word operations, and finally a byte operation.

88/2X instruction timings are independent of operand alignment, since byte operations are always performed. However, it is recommended to align numeric operands on even boundaries for maximum performance in case the program is transported to an 86/2X.

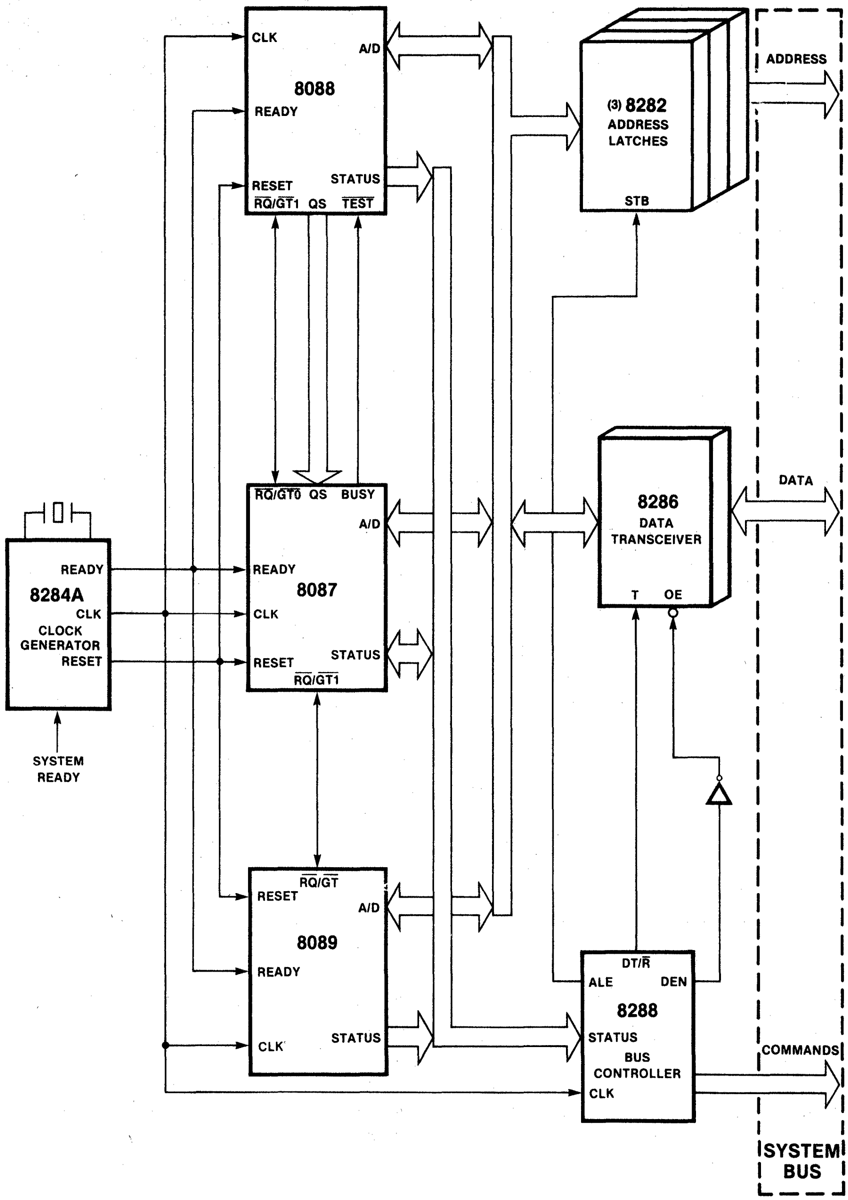


Figure 10. iAPX 88/21

RQ/GT CONNECTION

Two decisions must be made when connecting the 8087 to a system. The first is how to interconnect the RQ/GT signals of all local bus masters. The RQ/GT decision affects the response time to service local bus requests from other local bus masters, such as an 8089 IOP or other coprocessor. The interrupt connection affects the response time to service an interrupt request and how user-interrupt handlers are written. The implications of how these pins are connected concern both the hardware designer and programmer and must be understood by both.

The RQ/GT issue can be broken into three general categories, depending on system configuration: 86/20 or 88/20, 86/21 or 88/21, and 86/22 or 88/22. Remote operation of an IOP is not effected by the 8087 RQ/GT connection.

iAPX 86/20, 88/20

For an 86/20 or 88/20 just connect the RQ/GT0 pin of the 8087 to RQ/GT1 of the host (see Figure 5), and skip forward to the interrupt discussion on page 15.

iAPX 86/21, 88/21

For an 86/21 or 88/21, connect RQ/GT0 of the 8087 to RQ/GT1 of the host, connect RQ/GT of the 8089 to RQ/GT1 of the 8087 (see Figure 10, page 12), and skip forward to the interrupt discussion on page 15.

The RQ/GT1 pin of the 8087 exists to provide one I/O processor with a low maximum wait time for the local bus. The maximum wait times to gain control of the local bus for a device attached to RQ/GT1 of an 8087 for an 8086 or 8088 host are shown in Table 2. These numbers are all dependent on when the host will release the local bus to the 8087.

As Table 2 implies, three factors determine when the host will release the local bus:

1) What type of host is there, an 8086 or 8088?
2) What is the current instruction being executed?
3) How is the lock prefix being used?

An 8086 host will not release the local bus between the two consecutive byte operations performed for odd-aligned word operands. The 8088, in contrast, will never release the local bus between the two bytes of a word transfer, independent of its byte alignment.

Host operations such as acknowledging an interrupt will not release the local bus for several bus cycles.

Using a lock prefix in front of a host instruction prevents the host from releasing the local bus during the execution of that instruction.

8087 RQ/GT Function

The presence of the 8087 in the RQ/GT path from the IOP to the host has little effect on the maximum wait time seen by the IOP when requesting the local bus. The 8087 adds two clocks of delay to the basic time required by the host. This low delay is achieved due to a preemptive protocol implemented by the 8087 on RQ/GT1.

The 8087 always gives higher priority to a request for the local bus from a device attached to its RQ/GT1 pin than to a request generated internally by the 8087. If the 8087 currently owns the local bus and a request is made to its RQ/GT1 pin, the 8087 will finish the current memory cycle and release the local bus to the requestor. If the request from the devices arrives when the 8087 does not own the local bus, then the 8087 will pass the request on to the host via its RQ/GT0 pin.

Table 2. Worst Case Local Bus Request Wait Times in Clocks

System Configuration	No Locked Instructions	Only Locked Exchange	Other Locked Instructions
iAPX 86/21 even aligned words	15_1	35_1	max (15_1, *)
iAPX 86/21 odd aligned words	15_1	43_2	max (43_2, *)
iAPX 88/21	15_1	43_2	max (43_2, *)

Notes: 1. Add two clocks for each wait state inserted per bus cycle
2. Add four clocks for each wait state inserted per bus cycle
* Execution time of longest locked instruction

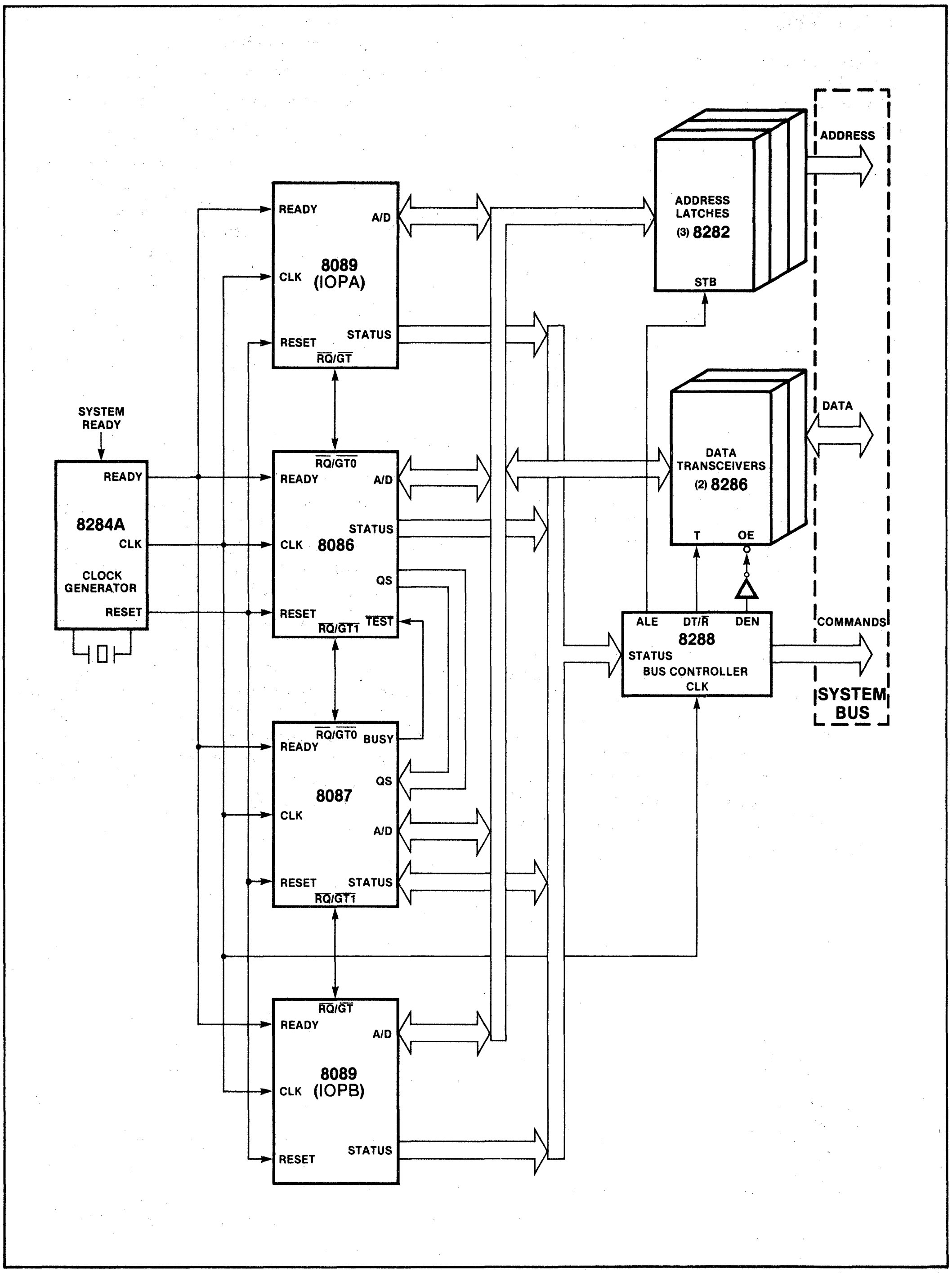


Figure 11. iAPX 86/22 System

IAPX 86/22, 88/22

An 86/22 system offers two alternates regarding to which IOP to connect an I/O device. Each IOP will offer a different maximum delay time to servide an I/O request. (See Fig. 11)

The second 8089 (IOPA) must use the RQ/GT0 pin of the host. With two IOPs the designer must decide which IOP services which I/O devices, determined by the maximum wait time allowed between when an I/O device requests IOP service and the IOP can respond. The maximum service delay times of the two IOPs can be very different. It makes little difference which of the two host RQ/GT pins are used.

The different wait times are due to the non-preemptive nature of bus grants between the two host RQ/GT pins. No communication of a need to use the local bus is possible between IOPA and the 8087/IOPB combination. Any request for the local bus by the IOPA must wait in the worst case for the host, 8087, and IOPB to finish their longest sequence of memory cycles. IOPB must wait in the worst case for the host and IOPA to finish their longest sequence of memory cycles. The 8087 has little effect on the maximum wait time of IOPB.

DELAY EFFECTS OF THE 8087

The delay effects of the 8087 on IOPA can be significant. When executing special instructions (FSAVE, FNSAVE, FRSTOR), the 8087 can perform 50 or 96 consecutive memory cycles with an 8086 or 8088 host, respectively. These instructions do not affect response time to local bus requests seen by an IOPB.

If the 8087 is performing a series of memory cycles while executing these instructions, and IOPB requests the local bus, the 8087 will stop its current memory activity, then release the local bus to IOPB.

The 8087 cannot release the bus to IOPA since it cannot know that IOPA wants to use the local bus, like it can for IOPB.

REDUCING 8087 DELAY EFFECTS

For 86/22 or 88/22 systems requiring lower maximum wait times for IOPA, it is possible to reduce the worst case bus usage of the 8087. If three 8087 instructions are never executed; namely FSAVE, FNSAVE, or FRSTOR, the maximum number of consecutive memory cycles performed by the 8087 is 10 or 16 for an 8086 or 8088 host respectively. The function of these instructions can be emulated with other 8087 instructions.

Appendix B shows an example of how these three instructions can be emulated. This improvment does have a cost, in the increased execution time of 427 or 747 additional clocks for an 8086 or 8088 respectively, for the equivalent save and restore operations. These operations appear in time-critical context-switching functions of an operating system or interrupt handler. This technique has no affect on the maximum wait time seen by IOPB or wait time seen by IOPA due to IOPB.

Which IOP to connect to which I/O device in an 86/22 or 88/22 system will depend on how quickly an I/O request by the device must be serviced by the IOP. This maximum time must be greater than the sum of the maximum delay of the IOP and the maximum wait time to gain control of the local bus by the IOP.

If neither IOP offers a fast enough response time, consider remote operation of the IOP.

8087 INT Connection

The next decision in adding the 8087 to an 8086 or 8088 system is where to attach the INT signal of the 8087. The INT pin of the 8087 provides an external indication of software-selected numeric errors. The numeric program will stop until something is done about the error. Deciding where to connect the INT signal can have important consequences on other interrupt handlers.

WHAT ARE NUMERIC ERRORS?

A numeric error occurs in the NPX whenever an operation is attempted with invalid operands or attempts to produce a result which cannot be represented. If an incorrect or questionable operation is attempted by a program, the NPX will always indicate the event. Examples of errors on the NPX are: 1/0, square root of −1, and reading from an empty register. For a detailed description of when the 8087 detects a numeric error, refer to the *Numerics Supplement*. (See Lit. Ref).

WHAT TO DO ABOUT NUMERIC ERRORS

Two possible courses of action are possible when a numeric error occurs. The NPX can itself handle the error, allowing numeric program execution to continue undisturbed, or software in the host can handle the error. To have the 8087 handle a numeric error, set its associated mask bit in the NPX control word. Each numeric error may be individually masked.

The NPX has a default fixup action defined for all possible numeric errors when they are masked. The default actions were carefully selected for their generality and safety.

For example, the default fixup for the precision error is to round the result using the rounding rules currently in effect. If the invalid error is masked, the NPX will generate a special value called indefinite as the result of any invalid operation.

NUMERIC ERRORS (CON'T)

Any arithmetic operation with an indefinite operand will always generate an indefinite result. In this manner, the result of the original invalid operation will propagate throughout the program wherever it is used.

When a questionable operation such as multiplying an unnormal value by a normal value occurs, the NPX will signal this occurrence by generating an unnormal result.

The required response by host software to a numeric error will depend on the application. The needs of each application must be understood when deciding on how to treat numeric errors. There are three attitudes towards a numeric error:

1) No response required. Let the NPX perform the default fixup.
2) Stop everything, something terrible has happened!
3) Oh, not again! But don't disrupt doing something more important.

SIMPLE ERROR HANDLING

Some very simple applications may mask all of the numeric errors. In this simple case, the 8087 INT signal may be left unconnected since the 8087 will never assert this signal. If any numeric errors are detected during the course of executing the program, the NPX will generate a safe result. It is sufficient to test the final results of the calculation to see if they are valid.

Special values like not-a-number (NAN), infinity, indefinite, denormals, and unnormals indicate the type and severity of earlier invalid or questionable operations.

SEVERE ERROR HANDLING

For dedicated applications, programs should not generate or use any invalid operands. Furthermore, all numbers should be in range. An operand or result outside this range indicates a severe fault in the system. This situation may arise due to invalid input values, program error, or hardware faults. The integrity of the program and hardware is in question, and immediate action is required.

In this case, the INT signal can be used to interrupt the program currently running. Such an interrupt would be of high priority. The interrupt handler responsible for numeric errors might perform system integrity tests and then restart the system at a known, safe state. The handler would not normally return to the point of error.

Unmasked numeric errors are very useful for testing programs. Correct use of synchronization, (Page 21), allows the programmer to find out exactly what operands, instruction, and memory values caused the error. Once testing has finished, an error then becomes much more serious.

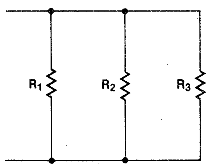
The *8086 Family Numerics Supplement* recommends masking all errors except invalid. (See Lit. Ref.). In this case the NPX will safely handle such errors as underflow, overflow, or divide by zero. Only truly questionable operations will disturb the numerics program execution.

An example of how infinities and divide by zero can be harmless occurs when calculating the parallel resistance of several values with the standard formula (Figure 12). If R1 becomes zero, the circuit resistance becomes 0. With divide by zero and precision masked, the NPX will produce the correct result.

NUMERIC EXCEPTION HANDLING

For some applications, a numeric error may not indicate a severe problem. The numeric error can indicate that a hardware resource has been exhausted, and the software must provide more. These cases are called exceptions since they do not normally arise.

Special host software will handle numeric error exceptions when they infrequently occur. In these cases, numeric exceptions are expected to be recoverable although not requiring immediate service by the host. In effect, these exceptions extend the functionality of the NDP. Examples of extensions are: normalized only arithmetic, extending the register stack to memory, or tracing special data values.



$$\text{Equivalent resistance} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}}$$

Figure 12. Infinity Arithmetic Example

HOST INTERRUPT OVERVIEW

The host has only two possible interrupt inputs, a non-maskable interrupt (NMI) and a maskable interrupt (INTR). Attaching the 8087 INT pin to the NMI input is not recommended. The following problems arise: NMI cannot be masked, it is usually reserved for more important functions like sanity timers or loss of power signal, and Intel supplied software for the NDP will not support NMI interrupts. The INTR input of the host allows interrupt masking in the CPU, using an Intel 8259A Programmable Interrupt Controller (PIC) to resolve multiple interrupts, and has Intel support.

NUMERIC INTERRUPT CHARACTERISTICS

Numeric error interrupts are different from regular instruction error interrupts like divide by zero. Numeric interrupts from the 8087 can occur long after the ESCAPE instruction that started the failing operation. For example, after starting a numeric multiply operation, the host may respond to an external interrupt and be in the process of servicing it when the 8087 detects an overflow error. In this case the interrupt is a result of some earlier, unrelated program.

From the point of view of the currently executing interrupt handler, numeric interrupts can come from only two sources: the current handler or a lower priority program.

To explicitly disable numeric interrupts, it is recommended that numeric interrupts be disabled at the 8087. The code example of Figure 13 shows how to disable any pending numeric interrupts then reenable them at the end of the handler. This code example can be safely placed in any routine which must prevent numeric interrupts from occurring. Note that the ESCAPE instructions act as NOPs if an 8087 is not present in the system. It is not recommended to use numeric mnemonics since they may be converted to emulator calls, which run comparatively slow, if the 8087 emulator used.

Interrupt systems have specific functions like fast response to external events or periodic execution of system routines. Adding an 8087 interrupt should not effect these functions. Desirable goals of any 8087 interrupt configuration are:

- Hide numeric interrupts from interrupt handlers that don't use the 8087. Since they didn't cause the numeric interrupt why should they be interrupted?
- Avoid adding code to interrupt handlers that don't use the 8087 to prevent interruption by the 8087.
- Allow other higher priority interrupts to be serviced while executing a numeric exception handler.
- Provide numeric exception handling for interrupt service routines which use the 8087.
- Avoid deadlock as described in a later section (page 24)

```
;
; Disable any possible numeric interrupt from the 8087. This code is safe to place in any
; procedure. If an 8087 is not present, the ESCAPE instructions will act as nops. These
; instructions are not affected by the TEST pin of the host. Using the 8087 emulator will not
; convert these instructions into interrupts. A word variable, called control, is required to hold
; the 8087 control word. Control must not be changed until it is reloaded into the 8087.
;
ESC   15, control                    ; (FNSTCW) Save current 8087 control word
NOP                                  ; Delay while 8087 saves current control
NOP                                  ; register value
ESC   28,cx                          ; (FNDISI) Disable any 8087 interrupts
                                     ; Set IEM bit in 8087 control register
                                     ; The contents of cx is irrelevant
                                     ; Interrupts can now be enabled

                          (Your Code Here)
;
; Reenable any pending interrupts in the 8087. This instruction does not disturb any 8087 instruction
; currently in progress since all it does is change the IEM bit in the control register.
;
TEST  control, 80H                   ; Look at IEM bit
JNZ   $+4                            ; If IEM=1 skip FNENI
ESC   28,ax                          ; (FNENI) reenable 8087 interrupts
```

Figure 13. Inhibit/Enable 8087 Interrupts

Recommended Interrupt Configurations

Five categories cover most uses of the 8087 interrupt in fixed priority interrupt systems. For each category, an interrupt configuration is suggested based on the goals mentioned above.

1. All errors on the 8087 are always masked. Numeric interrupts are not possible. Leave the 8087 INT signal unconnected.
2. The 8087 is the only interrupt in the system. Connect the 8087 INT signal directly to the host's INTR input. (See Figure 14 on page 19). A bus driver supplies interrupt vector 10_{16} for compatibility with Intel supplied software.
3. The 8087 interrupt is a stop everything event. Choose a high priority interrupt input that will terminate all numerics related activity. This is a special case since the interrupt handler may never return to the point of interruption (i.e. reset the system and restart rather than attempt to continue operation).
4. Numeric exceptions or numeric programming errors are expected and all interrupt handlers either don't use the 8087 or only use it with all errors masked. Use the lowest priority interrupt input. The 8087 interrupt handler should allow further interrupts by higher priority events. The PIC's priority system will automatically prevent the 8087 from disturbing other interrupts without adding extra code to them.
5. Case 4 holds except that interrupt handlers may also generate numeric interrupts. Connect the 8087 INT signal to multiple interrupt inputs. One input would still be the lowest priority input as in case 4. Interrupt handlers that may generate a numeric interrupt will require another 8087 INT connection to the next highest priority interrupt. Normally the higher priority numeric interrupt inputs would be masked and the low priority numeric interrupt enabled. The higher priority interrupt input would be unmasked only when servicing an interrupt which requires 8087 exception handling.

All of these configurations hide the 8087 from all interrupt handlers which do not use the 8087. Only those interrupt handlers that use the 8087 are required to perform any special 8087 related interrupt control activities.

A conflict can arise between the desired PIC interrupt input and the required interrupt vector of 10_{16} for compatibility with Intel software for numeric interrupts. A simple solution is to use more than one interrupt vector for numeric interrupts, all pointing at the same 8087 interrupt handler. Design the numeric interrupt handler such that it need not know what the interrupt vector was (i.e. don't use specific EOI commands).

If an interrupt system uses rotating interrupt priorities, it will not matter which interrupt input is used.

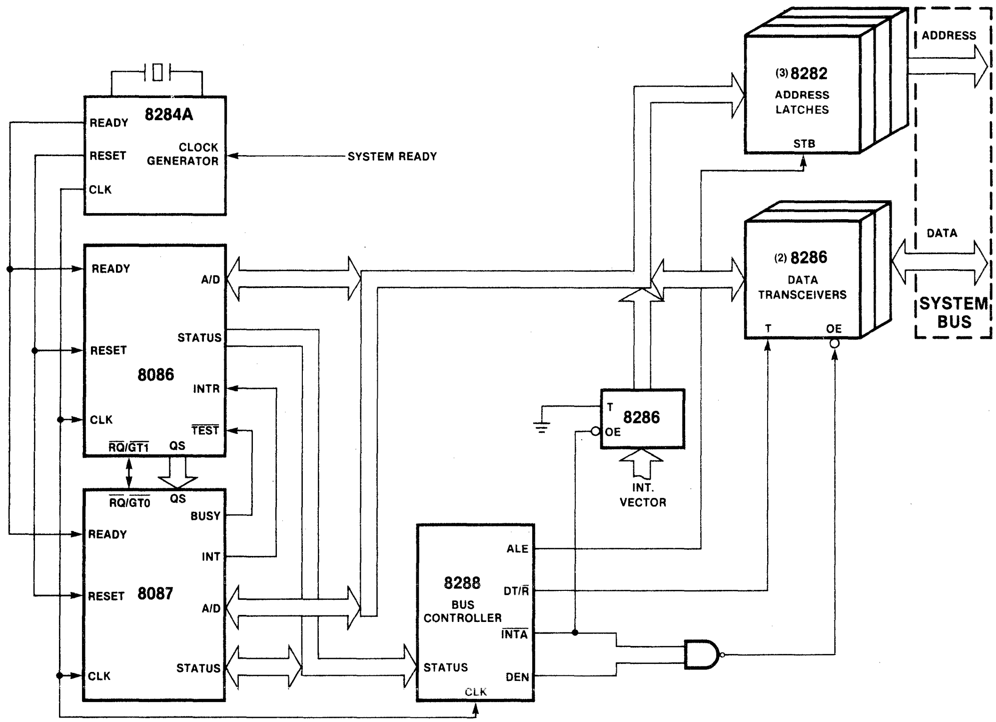


Figure 14. iAPX 86/20 With Numerics Interrupt Only

GETTING STARTED IN SOFTWARE

Now we are ready to run numeric programs. Developing numeric software will be a new experience to some programmers. This section of the application note is aimed at describing the programming environment and providing programming guidelines for the NPX. The term NPX is used to emphasize that no distinction is made between the 8087 component or an emulated 8087.

Two major areas of numeric software can be identified: systems software and applications software. Products such as iRMX™ 86 provide system software as an off-the-shelf product. Some applications use specially developed systems software optimized to their needs.

Whether the system software is specially tailored or common, they share issues such as using concurrency, maintaining synchronization between the host and 8087, and establishing programming conventions. Applications software directly performs the functions of the application. All applications will be concerned with initialization and general programming rules for the NPX. Systems software will be more concerned with context switching, use of the NPX by interrupt handlers, and numeric exception handlers.

How to Initialize the NPX

The first action required by the NPX is initialization. This places the NPX in a known state, unaffected by other activity performed earlier. This initialization is similar to that caused by the RESET signal of the 8087. All the error masks are set, all registers are tagged empty, the TOP field is set to 0, default rounding, precision, and infinity controls are set. The 8087 emulator requires more initialization than the component. Before the emulator may be used, all its interrupt vectors must be set to point to the correct entry points within the emulator.

To provide compatibility between the emulator and component in this special case, a call to an external procedure should be used before the first numeric instruction. In ASM86 the programmer must call the external function INIT87. (Fig. 15). For PLM86, the programmer must call the built-in function INIT$REAL$MATH$UNIT. PLM86 will call INIT87 when executing the INIT$REAL$MATH$UNIT built-in function.

The function supplied for INIT87 will be different, depending on whether the emulator library, called E8087.LIB, or component library, called 8087.LIB, were used at link time. INIT87 will execute either an FNINIT instruction for the 8087 or initialize the 8087 emulator interrupt vectors, as appropriate.

Concurrency Overview

With the NPX initialized, the next step in writing a numeric program is learning about concurrent execution within the NDP.

Concurrency is a special feature of the 8087, allowing it and the host to simultaneously execute different instructions. The 8087 emulator does not provide concurrency since it is implemented by the host.

The benefit of concurrency to an application is higher performance. All Intel high level languages automatically provide for and manage concurrency in the NDP. However, in exchange for the added performance, the assembly language programmer must understand and manage some areas of concurrency. This section is for the assembly language programmer or well-informed, high level language programmer.

Whether the 8087 emulator or component is used, care should be taken by the assembly language programmer to follow the rules described below regarding synchronization. Otherwise, the program may not function correctly with current or future alternatives for implementing the NDP.

Concurrency is possible in the NDP because both the host and 8087 have separate arithmetic and control units. The host and coprocessor automatically decide who will perform any single instruction. The existence of the 8087 as a separate unit is not normally apparent.

Numeric instructions, which will be executed by the 8087, are simply placed in line with the instructions for the host. Numeric instructions are executed in the same order as they are encountered by the host in its instruction stream. Since operations performed by the 8087 generally require more time than operations performed by the host, the host can execute several of its instructions while the 8087 performs one numeric operation.

```
IN PLM86:
     CALL INIT$REAL$MATH$UNIT;
IN ASM86:
          EXTRN                    INIT87:FAR
            •
            •
            •
            •
          CALL                     INIT87
```

Figure 15. 8087 Initialization

MANAGING CONCURRENCY

Concurrent execution of the host and 8087 is easy to establish and maintain. The activities of numeric programs can be split into two major areas: program control and arithmetic. The program control part performs activities like deciding what functions to perform, calculating addresses of numeric operands, and loop control. The arithmetic part simply performs the adds, subtracts, multiplies, and other operations on the numeric operands. The NPX and host are designed to handle these two parts separately and efficiently.

Managing concurrency is necessary because the arithmetic and control areas must converge to a well-defined state when starting another numeric operation. A well-defined state means all previous arithmetic and control operations are complete and valid.

Normally, the host waits for the 8087 to finish the current numeric operation before starting another. This waiting is called synchronization.

Managing concurrent execution of the 8087 involves three types of synchronization: instruction, data, and error. Instruction and error synchronization are automatically provided by the compiler or assembler. Data synchronization must be provided by the assembly language progammer or compiler.

Instruction Synchronization

Instruction synchronization is required because the 8087 can only perform one numeric operation at a time. Before any numeric operation is started, the 8087 must have completed all activity from previous instructions.

The WAIT instruction on the host lets it wait for the 8087 to finish all numeric activity before starting another numeric instruction. The assembler automatically provides for instruction synchronization since a WAIT instruction is part of most numeric instructions. A WAIT instruction requires 1 byte code space and 2.5 clocks average execution time overhead.

Instruction synchronization as provided by the assembler or a compiler allows concurrent operation in the NDP. An execution time comparison of NDP concurrency and non-concurrency is illustrated in Figure 16. The non-concurrent program places a WAIT instruction immediately after a multiply instruction ESCAPE instruction. The 8087 must complete the multiply operation before the host executes the MOV instruction on statement 2. In contrast, the concurrent example allows the host to calculate the effective address of the next operand while the 8087 performs the multiply. The execution time of the concurrent technique is the longest of the host's execution time from line 2 to 5 and the execution time of the 8087 for a multiply instruction. The execution time of the non-concurrent example is the sum of the execution times of statements 1 to 5.

```
;
; This code macro defines two instructions which do not allow any concurrency of execution with
; the host. A register version and memory version of the instruction is shown. It is assumed that the
; 8087 is always idle from the previous instruction. Allow space for emulator fixups.
;
R233 Record RF6:2, Mid3:3, RF7:3
CodeMacro NCMUL dst:T, src:F
RNfix 000B
R233 (11B, 001B, src)
RWfix
EndM

CodeMacro NCMUL memop:Mq
RNfixM 100B, memop
ModRM 001B, memop
RWfix
EndM
```

Statement	Concurrent		Non Concurrent	
1	FMUL	st(0), st(1)	NCMUL	st(0), st(1)
2	MOV	ax, size A	MOV	ax, size A
3	MUL	index	MUL	index
4	MOV	bx, ax	MOV	bx, ax
5	FMUL	A [bx]	NCMUL	A [bx]

Figure 16. Concurrent Versus Non-Concurrent Program

Data Synchronization

Managing concurrency requires synchronizing data references by the host and 8087.

Figure 17 shows four possible cases of the host and 8087 sharing a memory value. The second two cases require the FWAIT instruction shown for data synchronization. In the first two cases, the host will finish with the operand I before the 8087 can reference it. The coprocessor interface guarantees this. In the second two cases, the host must wait for the 8087 to finish with the memory operand before proceeding to reuse it. The FWAIT instruction in case 3 forces the host to wait for the 8087 to read I before changing it. In case 4, the FWAIT prevents the host from reading I before the 8087 sets its value.

Obviously, the programmer must recognize any form of the two cases shown above which require explicit data synchronization. Data synchronization is not a concern when the host and 8087 are using different memory operands during the course of one numeric instruction. Figure 16 shows such an example of the host performing activity unrelated to the current numeric instruction being executed by the 8087. Correct recognition of these cases by the programmer is the price to be paid for providing concurrency at the assembly language level.

Automatic Data Synchronization

Two methods exist to avoid the need for manual recognition of when data synchronization is needed: use a high level language which will automatically establish concurrency and manage it, or sacrifice some performance for automatic data synchronization by the assembler.

When a high level language is not adequate, the assembler can be changed to always place a WAIT instruction after the ESCAPE instruction. Figure 18 shows an example of how to change the ASM86 code macro for the FIST instruction to automatically place an FWAIT instruction after the ESCAPE instruction. The lack of any possible concurrent execution between the host and 8087 while the FIST instruction is executing is the price paid for automatic data synchronization.

An explicit FWAIT instruction for data synchronization, can be eliminated by using a subsequent numeric instruction. After this subsequent instruction has started execution, all memory references in earlier numeric instructions are complete. Reaching the next host instruction after the synchronizing numeric instruction indicates previous numeric operands in memory are available.

The data synchronization purpose of any FWAIT or numeric instruction must be well documented. Otherwise, a change to the program at a later time may remove the synchronizing numeric instruction, causing program failure, as:

```
FISTP   I
FMUL
MOV     AX, I    ; I is safe to use
```

```
Case 1:                     Case 3:
    MOV    I, 1                 FILD     I
    FILD   I                    FWAIT
                                MOV      I,5

Case 2:                     Case 4:
    MOV    AX, I                FISTP    I
    FISTP  I                    FWAIT
                                MOV      AX,I
```

Figure 17. Data Exchange Example

```
;
; This is a code macro to redefine the FIST
; instruction to prevent any concurrency
; while the instruction runs. A wait
; instruction is placed immediately after the
; escape to ensure the store is done
; before the program may continue. This
; code macro will work with the 8087
; emulator, automatically replacing the
; wait escape with a nop.
;
CodeMacro FIST memop: Mw
RfixM 111B, memop
ModRM 010B, memop
RWfix
EndM
```

Figure 18. Non-Concurrent FIST Instruction Code Macro

DATA SYNCHRONIZATION RULES EXCEPTIONS

There are five exceptions to the above rules for data synchronization. The 8087 automatically provides data synchronization for these cases. They are necessary to avoid deadlock (described on page 24). The instructions FSTSW/FNSTSW, FSTCW/FNSTCW, FLDCW, FRSTOR, and FLDENV do not require any waiting by the host before it may read or modify the referenced memory location.

The 8087 provides the data synchronization by preventing the host from gaining control of the local bus while these instructions execute. If the host cannot gain control of the local bus, it cannot change a value before the 8087 reads it, or read a value before the 8087 writes into it.

The coprocessor interface guarantees that, when the host executes one of these instructions, the 8087 will immediately request the local bus from the host. This request is timed such that, when the host finishes the read operation identifying the memory operand, it will always grant the local bus to the 8087 before the host may use the local bus for a data reference while executing a subsequent instruction. The 8087 will not release the local bus to the host until it has finished executing the numeric instruction.

Error Synchronization

Numeric errors can occur on almost any numeric instruction at any time during its execution. Page 15 describes how a numeric error may have many interpretations, depending on the application. Since the response to a numeric error will depend on the application, this section covers topics common to all uses of the NPX. We will review why error synchronization is needed and how it is provided.

Concurrent execution of the host and 8087 requires synchronization for errors just like data references and numeric instructions. In fact, the synchronization required for data and instructions automatically provides error synchronization.

However, incorrect data or instruction synchronization may not cause a problem until a numeric error occurs. A further complication is that a programmer may not expect his numeric program to cause numeric errors, but in some systems they may regularly happen. To better understand these points, let's look at what can happen when the NPX detects an error.

ERROR SYNCHRONIZATION FOR EXTENSIONS

The NPX can provide a default fixup for all numeric errors. A program can mask each individual error type to indicate that the NPX should generate a safe, reasonable result. The default error fixup activity is simply treated as part of the instruction which caused the error. No external indication of the error will be given. A flag in the numeric status register will be set to indicate that an error was detected, but no information regarding where or when will be available.

If the NPX performs its default action for all errors, then error synchronization is never exercised. But this is no reason to ignore error synchronization.

Another alternative exists to the NPX default fixup of an error. If the default NPX response to numeric errors is not desired, the host can implement any form of recovery desired for any numeric error detectable by the NPX. When a numeric error is unmasked, and the error occurs, the NPX will stop further execution of the numeric instruction. The 8087 will signal this event on the INT pin, while the 8087 emulator will cause interrupt 10_{16} to occur. The 8087 INT signal is normally connected to the host's interrupt system. Refer to page 18 for further discussion on wiring the 8087 INT pin.

Interrupting the host is a request from the NPX for help. The fact that the error was unmasked indicates that further numeric program execution under the arithmetic and programming rules of the NPX is unreasonable. Error synchronization serves to insure the NDP is in a well defined state after an unmasked numeric error occured. Without a well defined state, it is impossible to figure out why the error occured.

Allowing a correct analysis of the error is the heart of error synchronization.

NDP ERROR STATES

If concurrent execution is allowed, the state of the host when it recognizes the interrupt is undefined. The host may have changed many of its internal registers and be executing a totally different program by the time it is interrupted. To handle this situation, the NPX has special registers updated at the start of each numeric instruction to describe the state of the numeric program when the failed instruction was attempted. (See Lit. Ref. p. iii)

Besides programmer comfort, a well-defined state is important for error recovery routines. They can change the arithmetic and programming rules of the 8087. These changes may redefine the default fixup from an error, change the appearance of the NPX to the programmer, or change how arithmetic is defined on the NPX.

EXTENSION EXAMPLES

A change to an error response might be to automatically normalize all denormals loaded from memory. A change in appearance might be extending the register stack to memory to provide an "infinite" number of numeric registers. The arithmetic of the 8087 can be changed to automatically extend the precision and range of variables when exceeded. All these functions can be implemented on the NPX via numeric errors and associated recovery routines in a manner transparent to the programmer.

Without correct error synchronization, numeric subroutines will not work correctly in the above situations.

Incorrect Error Synchronization

An example of how some instructions written without error synchronization will work initially, but fail when moved into a new environment is:

```
FILD    COUNT
INC     COUNT
FSQRT
```

Three instructions are shown to load an integer, calculate its square root, then increment the integer. The coprocessor interface of the 8087 and synchronous execution of the 8087 emulator will allow this program to execute correctly when no errors occur on the FILD instruction.

But, this situation changes if the numeric register stack is extended to memory on an 8087. To extend the NPX stack to memory, the invalid error is unmasked. A push to a full register or pop from an empty register will cause an invalid error. The recovery routine for the error must recognize this situation, fixup the stack, then perform the original operation.

The recovery routine will not work correctly in the example. The problem is that there is no guarantee that COUNT will not be incremented before the 8087 can interrupt the host. If COUNT is incremented before the interrupt, the recovery routine will load a value of COUNT one too large, probably causing the program to fail.

Error Synchronization and WAITs

Error synchronization relies on the WAIT instructions required by instruction and data synchronization and the INT and BUSY signals of the 8087. When an unmasked error occurs in the 8087, it asserts the BUSY and INT signals. The INT signal is to interrupt the host, while the BUSY signal prevents the host from destroying the current numeric context.

The BUSY signal will never go inactive during a numeric instruction which asserts INT.

The WAIT instructions supplied for instruction synchronization prevent the host from starting another numeric instruction until the current error is serviced. In a like manner, the WAIT instructions required for data synchronization prevent the host from prematurely reading a value not yet stored by the 8087, or overwriting a value not yet read by the 8087.

The host has two responsibilities when handling numeric errors.1.) It must not disturb the numeric context when an error is detected, and 2.) it must clear the numeric error and attempt recovery from the error. The recovery program invoked by the numeric error may resume program execution after proper fixup, display the state of the NDP for programmer action, or simply abort the program. In any case, the host must do something with the 8087. With the INT and BUSY signals active, the 8087 cannot perform any useful work. Special instructions exist for controlling the 8087 when in this state. Later, an example is given of how to save the state of the NPX with an error pending. (See page 29)

Deadlock

An undesirable situation may result if the host cannot be interrupted by the 8087 when asserting INT. This situation, called deadlock, occurs if the interrupt path from the 8087 to the host is broken.

The 8087 BUSY signal prevents the host from executing further instructions (for instruction or data synchronization) while the 8087 waits for the host to service the exception. The host is waiting for the 8087 to finish the current numeric operation. Both the host and 8087 are waiting on each other. This situation is stable unless the host is interrupted by some other event.

Deadlock has varying affects on the NDP's performance. If no other interrupts in the system are possible, the NDP will wait forever. If other interrupts can arise, then the NDP can perform other functions, but the affected numeric program will remain "frozen".

SOLVING DEADLOCK

Finding the break in the interrupt path is simple. Look for disabled interrupts in the following places: masked interrupt enable in the host, explicitly masked interrupt request in the interrupt controller, implicitly masked interrupt request in the interrupt controller due to a higher priority interrupt in service, or other gate functions, usually in TTL, on the host interrupt signal.

DEADLOCK AVOIDANCE

Application programmers should not be concerned with deadlock. Normally, applications programs run with unmasked numeric errors able to interrupt them. Deadlock is not possible in this case. Traditionally, systems software or interrupt handlers may run with numeric interrupts disabled. Deadlock prevention lies in this domain. The golden rule to abide by is: "Never wait on the 8087 if an unmasked error is possible and the 8087 interrupt path may be broken."

Error Synchronization Summary

In summary, error synchronization involves protecting the state of the 8087 after an exception. Although not all applications may initially require error synchronization, it is just good programming practice to follow the rules. The advantage of being a "good" numerics programmer is generality of your program so it can work in other, more general environments.

Summary

Synchronization is the price for concurrency in the NDP. Intel high level language compilers will automatically provide concurrency and manage it with synchronization. The assembly language programmer can choose between using concurrency or not. Placing a WAIT instruction immediately after any numeric instruction will prevent concurrency and avoid synchronization concerns.

The rules given above are complete and allow concurrency to be used to full advantage.

Synchronization and the Emulator

The above discussion on synchronization takes on special meaning with the 8087 emulator. The 8087 emulator does not allow any concurrency. All numeric operand memory references, error tests, and wait for instruction completion occur within the emulator. As a result, programs which do not provide proper instruction, data, or error synchronization may work with the 8087 emulator while failing on the component.

Correct programs for the 8087 work correctly on the emulator.

Special Control Instructions of the NPX

The special control instructions of the NPX: FNINIT, FNSAVE, FNSTENV, FRSTOR, FLDENV, FLDCW, FNSTSW, FNSTCW, FNCLEX, FNENI, and FNDISI remove some of the synchronization requirements mentioned earlier. They are discussed here since they represent exceptions to the rules mentioned on page 21.

The instructions FNINIT, FNSAVE, FNSTENV, FNSTSW, FNCLEX, FNENI, and FNDISI do not wait for the current numeric instruction to finish before they execute. Of these instructions, FNINIT, FNSTSW, FNCLEX, FNENI and FNDISI will produce different results, depending on when they are executed relative to the current numeric instruction.

For example, FNCLEX will cause a different status value to result from a concurrent arithmetic operation, depending on whether is is executed before or after the error status bits are updated at the end of the arithmetic operation. The intended use of FNCLEX is to clear a known error status bit which has caused BUSY to be asserted, avoiding deadlock.

FNSTSW will safely, without deadlock, report the busy and error status of the NPX independent of the NDP interrupt status.

FNINIT, FNENI, and FNDISI are used to place the NPX into a known state independent of its current state. FNDISI will prevent an unmasked error from asserting BUSY without disturbing the current error status bits. Appendix A shows an example of using FNDISI.

The instructions FNSAVE and FNSTENV provide special functions. They allow saving the state of the NPX in a single instruction when host interrupts are disabled.

Several host and numeric instructions are necessary to save the NPX status if the interrupt status of the host is unknown. Appendix A and B show examples of saving the NPX state. As the *Numerics Supplement* explains, host interrupts must always be disabled when executing FNSAVE or FNSTENV.

The seven instructions FSTSW/FNSTSW, FSTCW/FNSTCW, FLDCW, FLDENV, and FRSTOR do not require explicit WAIT instructions for data synchronization. All of these instructions are used to interrogate or control the numeric context.

Data synchronization for these instructions is automatically provided by the coprocessor interface. The 8087 will take exclusive control of the memory bus, preventing the host from interfering with the data values before the 8087 can read them. Eliminating the need for a WAIT instruction avoids potential deadlock problems.

The three load instructions FLDCW, FLDENV, and FRSTOR can unmask a numeric error, activating the 8087 BUSY signal. Such an error was the result of a previous numeric instruction and is not related to any fault in the instruction.

Data synchronization is automatically provided since the host's interrupts are usually disabled in context switching or interrupt handling, deadlock might result if the host executed a WAIT instruction with its interrupts disabled after these instructions. After the host interrupts are enabled, an interrupt will occur if an unmasked error was pending.

PROGRAMMING TECHNIQUES

The NPX provides a stack-oriented register set with stack-oriented instructions for numeric operands. These registers and instructions are optimized for numeric programs. For many programmers, these are new resources with new programming options available.

Using Numeric Registers and Instructions

The register and instruction set of the NDP is optimized for the needs of numeric and general purpose programs. The host CPU provides the instructions and data types needed for general purpose data processing, while the 8087 provides the data types and instructions for numeric processing.

The instructions and data types recognized by the 8087 are different from the CPU because numeric program requirements are different from those of general purpose programs. Numeric programs have long arithmetic expressions where a few temporary values are used in a few statements. Within these statements, a single value may be referenced many times. Due to the time involved to transfer values between registers and memory, a significant speed optimization is possible by keeping numbers in the NPX register file.

In contrast, a general data processor is more concerned with addressing data in simple expressions and testing the results. Temporary values, constant across several instructions, are not as common nor is the penalty as large for placing them in memory.As a result it is simpler for compilers and programmers to manage memory based values.

NPX Register Usage

The eight numeric registers in the NDP are stack oriented. All numeric registers are addressed relative to a value called the TOP pointer, defined in the NDP status register. A register address given in an instruction is added to the TOP value to form the internal absolute address. Relative addressing of numeric registers has advantages analogous to those of relative addressing of memory operands.

Two modes are available for addressing the numeric registers. The first mode implicitly uses the top and optional next element on the stack for operands. This mode does not require any addressing bits in a numeric instruction. Special purpose instructions use this mode since full addressing flexibility is not required.

The other addressing mode allows any other stack element to be used together with the top of stack register. The top of stack or the other register may be specified as the destination. Most two-operand arithmetic instructions allow this addressing mode. Short, easy to develop numeric programs are the result.

Just as relative addressing of memory operands avoids concerns with memory allocation in other parts of a program, top relative register addressing allows registers to be used without regard for numeric register assignments in other parts of the program.

STACK RELATIVE ADDRESSING EXAMPLE

Consider an example of a main program calling a subroutine, each using register addressing independent of the other. (Fig. 19) By using different values of the TOP field, different software can use the same relative register addresses as other parts of the program, but refer to different physical registers.

```
MAIN_PROGRAM:
  FLD      A
  FADD     ST, ST(1)
  CALL     SUBROUTINE      ; Argument is in ST(0)
  FSTP     B

SUBROUTINE:
  FLD      ST              ; ST(0) = ST(1) = Argument
  FSQRT                    ; Main program ST(1) is
  FADD     C               ; safe in ST(2) here
  FMULP    ST(1), ST
  RET
```

Figure 19. Stack Relative Addressing Example

Of course, there is a limit to any physical resource. The NDP has eight numeric registers. Normally, programmers must ensure a maximum of eight values are pushed on the numeric register stack at any time. For time-critical inner loops of real-time applications, eight registers should contain all the values needed.

REGISTER STACK EXTENSION

This hardware limitation can be hidden by software. Software can provide "virtual" numeric registers, expanding the register stack size to 6000 or more.

The numeric register stack can be extended into memory via unmasked numeric invalid errors which cause an interrupt on stack overflow or underflow. The interrupt handler for the invalid error would manage a memory image of the numeric stack copying values into and out of memory as needed.

The NPX will contain all the necessary information to identify the error, failing instruction, required registers, and destination register. After correcting for the missing hardware resource, the original numeric operation could be repeated. Either the original numeric instruction could be single stepped or the affect of the instruction emulated by a composite of table-based numeric instructions executed by the error handler.

With proper data, error, and instruction synchronization, the activity of the error handler will be transparent to programs. This type of extension to the NDP allows programs to push and pop numeric registers without regard for their usage by other subroutines.

Programming Conventions

With a better understanding of the stack registers, let's consider some useful programming conventions. Following these conventions ensures compatibility with Intel support software and high level language calling conventions.

1) If the numeric registers are not extended to memory, the programmer must ensure that the number of temporary values left in the NPX stack and those registers used by the caller does not exceed 8. Values can be stored to memory to provide enough free NPX registers.
2) Pass the first seven numeric parameters to a subroutine in the numeric stack registers. Any extra parameters can be passed on the host's stack. Push the values on the register or memory stack in left to right order. If the subroutine does not need to allocate any more numeric registers, it can execute solely out of the numeric register stack. The eighth register can be used for arithmetic operations. All parameters should be popped off when the subroutine completes.
3) Return all numeric values on the numeric stack. The caller may now take advantage of the extended precision and flexible store modes of the NDP.
4) Finish all memory reads or writes by the NPX before exiting any subroutine. This guarantees correct data and error synchronization. A numeric operation based solely on register contents is safe to leave running on subroutine exit.
5) The operating mode of the NDP should be transparent across any subroutine. The operating mode is defined by the control word of the NDP. If the subroutine needs to use a different numeric operating mode than that of the caller, the subroutine should first save the current control word, set the new operating mode, then restore the original control word when completed.

PROGRAMMING EXAMPLES

The last section of this application note will discuss five programming examples. These examples were picked to illustrate NDP programming techniques and commonly used functions. All have been coded, assembled, and tested. However, no guarantees are made regarding their correctness.

The programming examples are: saving numeric context switching, save numeric context without FSAVE/FNSAVE, converting ASCII to floating point, converting floating point to ASCII, and trigonometric functions. Each example is listed in a different appendix with a detailed written description in the following text. The source code is available in machine readable form from the Intel Insite User's Library, "Interactive 8087 Instruction Interpreter," catalog item AA20.

The examples provide some basic functions needed to get started with the numeric data processor. They work with either the 8087 or the 8087 emulator with no source changes.

The context switching examples are needed for operating systems or interrupt handlers which may use numeric instructions and operands. Converting between floating point and decimal ASCII will be needed to input or output numbers in easy to read form. The trigonometric examples help you get started with sine or cosine functions and can serve as a basis for optimizations if the angle arguments always fall into a restricted range.

APPENDIX A

OVERVIEW

Appendix A shows deadlock-free examples of numeric context switching. Numeric context switching is required by interrupt handlers which use the NPX and operating system context switchers. Context switching consists of two basic functions, save the numeric context and restore it. These functions must work independent of the current state of the NPX.

Two versions of the context save function are shown. They use different versions of the save context instruction. The FNSAVE/FSAVE instructions do all the work of saving the numeric context. The state of host interrupts will decide which instruction to use.

Using FNSAVE

The FNSAVE instruction is intended to save the NPX context when host interrupts are disabled. The host does not have to wait for the 8087 to finish its current operation before starting this operation. Eliminating the instruction synchronization wait avoids any potential deadlock.

The 8087 Bus Interface Unit (BIU) will save this instruction when encountered by the host and hold it until the 8087 Floating point Execution Unit (FEU) finishes its current operation. When the FEU becomes idle, the BIU will start the FEU executing the save context operation.

The host can execute other non-numeric instructions after the FNSAVE while the BIU waits for the FEU to finish its current operation. The code starting at NO_INT_NPX_SAVE shows how to use the FNSAVE instruction.

When executing the FNSAVE instruction, host interrupts must be disabled to avoid recursions of the instruction. The 8087 BIU can hold only one FNSAVE instruction at a time. If host interrupts were not disabled, another host interrupt might cause a second FNSAVE instruction to be executed, destroying the previous one saved in the 8087 BIU.

It is not recommended to explicitly disable host interrupts just to execute an FNSAVE instruction. In general, such an operation may not be the best course of action or even be allowed.

If host interrupts are enabled during the NPX context save function, it is recommended to use the FSAVE instruction as shown by the code starting at NPX_SAVE. This example will always work, free of deadlock, independent of the NDP interrupt state.

Using FSAVE

The FSAVE instruction performs the same operation as FNSAVE but it uses standard instruction synchronization. The host will wait for the FEU to be idle before initiating the save operation. Since the host ignores all interrupts between completing a WAIT instruction and starting the following ESCAPE instruction, the FEU is ready to immediately accept the operation (since it is not signalling BUSY). No recursion of the save context operation in the BIU is possible. However, deadlock must be considered since the host executes a WAIT instruction.

To avoid deadlock when using the FSAVE instruction, the 8087 must be prevented from signalling BUSY when an unmasked error exists.

The Interrupt Enable Mask (IEM) bit in the NPX control word provides this function. When IEM = 1, the 8087 will not signal BUSY or INT if an unmasked error exists. The NPX instruction FNDISI will set the IEM independent of any pending errors without causing deadlock or any other errors. Using the FNDISI and FSAVE instructions together with a few other glue instructions allows a general NPX context save function.

Standard data and instruction synchronization is required after executing the FNSAVE/FSAVE instruction. The wait instruction following an FNSAVE/FSAVE instruction is always safe since all NPX errors will be masked as part of the instruction execution. Deadlock is not possible since the 8087 will eventually signal not busy, allowing the host to continue on.

PLACING THE SAVE CONTEXT FUNCTION

Deciding on where to save the NPX context in an interrupt handler or context switcher is dependent on whether interrupts can be enabled inside the function. Since interrupt latency is measured in terms of the maximum time interrupts are disabled, the maximum wait time of the host at the data synchronizing wait instruction after the FNSAVE or the FSAVE instruction is important if host interrupts are disabled while waiting.

The wait time will be the maximum single instruction execution time of the 8087 plus the execution time of the save operation. This maximum time will be approximately 1300 or 1500 clocks, depending on whether the host is an 8086 or 8088, respectively. The actual time will depend on how much concurrency of execution between the host and 8087 is provided. The greater the concurrency, the lesser the maximum wait time will be.

If host interrupts can be enabled during the context save function, it is recommended to use the FSAVE instruction for saving the numeric context in the interruptable section. The FSAVE instruction allows instruction and data synchronizing waits to be interruptable. This technique removes the maximum execution time of 8087 instructions from system interrupt latency time considerations.

It is recommended to delay starting the numeric save function as long as possible to maintain the maximum amount of concurrent execution between the host and the 8087.

Using FRSTOR

Restoring the numeric context with FRSTOR does not require a data synchronizing wait afterwards since the 8087 automatically prevents the host from interfering with the memory load operation.

The code starting with NPX_RESTORE illustrates the restore operation. Error synchronization is not necessary since the FRSTOR instruction itself does not cause errors, but the previous state of the NPX may indicate an error.

If further numeric instructions are executed after the FRSTOR, and the error state of the new NPX context is unknown, deadlock may occur if numeric exceptions cannot interrupt the host.

NPX_save

```
;
; General purpose save of NPX context. This function will work independent of the interrupt state of
; the NDP. Deadlock can not occur. 47 words of memory are required by the variable save_area.
; Register ax is not transparent across this code.
;

NPX_save:
     FNSTCW    save_area         ; Save IEM bit status
     NOP                         ; Delay while 8087 saves control register
     FNDISI                      ; Disable 8087 BUSY signal
     MOV       ax, save_area     ; Get original control word
     FSAVE     save_area         ; Save NPX context, the host can be safely interrupted while
                                 ; waiting for the 8087 to finish. Deadlock is not possible since
     FWAIT                       ; IEM = 1.Wait for save to finish. Put original control word into
     MOV       save_area, ax     ; NPX context area. All done
```

no_int_NPX_save

```
;
; Save the NPX context with host interrupts disabled. No deadlock is possible. 47 words of memory
; are required by the variable save_area.
;

no_int_NPX_save:
     FNSAVE    save_area         ; Save NPX context. Wait for save to finish, no deadlock
     FWAIT                       ; is possible. Interrupts may be enabled now, all done
```

NPX_restore

```
;
; Restore the NPX context saved earlier. No deadlock is possible if no further numeric instructions
; are executed until the 8087 numeric error interrupt is enabled. The variable save_area is assumed
; to hold an NPX context saved earlier. It must be 47 words long.
;

NPX_restore:
     FRSTOR    save_area         ; Load new NPX context
```

APPENDIX B

OVERVIEW

Appendix B shows alternative techniques for switching the numeric context without using the FSAVE/FNSAVE or FRSTOR instructions. These alternative techniques are slower than those of Appendix A but they reduce the worst case continuous local bus usage of the 8087.

Only an iAPX 86/22 or iAPX 88/22 could derive any benefit from this alternative. By replacing all FSAVE/FNSAVE instructions in the system, the worst case local bus usage of the 8087 will be 10 or 16 consecutive memory cycles for an 8086 or 8088 host, respectively.

Instead of saving and loading the entire numeric context in one long series of memory transfers, these routines use the FSTENV/FNSTENV/FLDENV instructions and separate numeric register load/store instructions. Using separate load/store instructions for the numeric registers forces the 8087 to release the local bus after each numeric load/store instruction. The longest series of back-to-back memory transfers required by these instructions are 8/12 memory cycles for an 8086 or 8088 host, respectively. In contrast, the FSAVE/FNSAVE/FRSTOR instructions perform 50/94 back-to-back memory cycles for an 8086 or 8088 host.

Compatibility With FSAVE/FNSAVE

This function produces a context area of the same format produced by FSAVE/FNSAVE instructions. Other software modules expecting such a format will not be affected. All the same interrupt and deadlock considerations of FSAVE and FNSAVE also apply to FSTENV and FNSTENV. Except for the fact that the numeric environment is 7 words rather than the 47 words of the numeric context, all the discussion of Appendix A also applies here.

The state of the NPX registers must be saved in memory in the same format as the FSAVE/FNSAVE instructions. The program example starting at the label SMALL_BLOCK_NPX_SAVE illustrates a software loop that will store their contents into memory in the same top relative order as that of FSAVE/FNSAVE.

To save the registers with FSTP instructions, they must be tagged valid, zero, or special. This function will force all the registers to be tagged valid, independent of their contents or old tag, and then save them. No problems will arise if the tag value conflicts with the register's content for the FSTP instruction. Saving empty registers insures compatibility with the FSAVE/FNSAVE instructions. After saving all the numeric registers, they will all be tagged empty, the same as if an FSAVE/FNSAVE instruction had been executed.

Compatibility With FRSTOR

Restoring the numeric context reverses the procedure described above, as shown by the code starting at SMALL_BLOCK_NPX_RESTORE. All eight regissters are reloaded in the reverse order. With each register load, a tag value will be assigned to each register. The tags assigned by the register load does not matter since the tag word will be overwritten when the environment is reloaded later with FLDENV.

Two assumptions are required for correct operation of the restore function: all numeric registers must be empty and the TOP field must be the same as that in the context being restored. These assumptions will be satisfied if a matched set of pushes and pops were performed between saving the numeric context and reloading it.

If these assumptions cannot be met, then the code example starting at NPX_CLEAN shows how to force all the NPX registers empty and set the TOP field of the status word.

small_block_NPX_save

```
;
; Save the NPX context independent of NDP interrupt state. Avoid using the FSAVE instruction to
; limit the worst case memory bus usage of the 8087. The NPX context area formed will appear the
; same as if an FSAVE instruction had written into it. The variable save_area will hold the NPX
; context and must be 47 words long. The registers ax, bx, and cx will not be transparent.
;
small_block_NPX_save:
    FNSTCW  save_area           ; Save current IEM bit
    NOP                         ; Delay while 8087 saves control register
    FNDISI                      ; Disable 8087 BUSY signal
    MOV     ax, save_area       ; Get original control word
    MOV     cx, 8               ; Set numeric register count
    XOR     bx, bx              ; Tag field value for stamping all registers as valid
    FSTENV  save_area           ; Save NPX environment
    FWAIT                       ; Wait for the store to complete
    XCHG    save_area+4, bx     ; Get original tag value and set new tag value
    FLDENV  save_area           ; Force all register tags as valid. BUSY is still masked. No data
    MOV     save_area, ax       ; synchronization needed. Put original control word into NPX
    MOV     save_area+4, bx     ; environment. Put original tag word into NPX environment
    XOR     bx, bx              ; Set initial register index
reg_store_loop:
    FSTP    saved_reg [bx]      ; Save register
    ADD     bx, type saved_reg  ; Bump pointer to next register
    LOOP    reg_store_loop
                                ; All done
```

NPX_clean

```
;
; Force the NPX into a clean state with TOP matching the TOP field stored in the NPX context and all
; numeric registers tagged empty. Save_area must be the NPX environment saved earlier.
; Temp_env is a 7 word temporary area used to build a prototype NPX environment. Register ax will
; not be transparent.
;
NPX_clean:
    FINIT                       ; Put NPX into known state
    MOV     ax, save_area+2     ; Get original status word
    AND     ax, 3800H           ; Mask out the top field
    FSTENV  temp_env            ; Format a temporary environment area with all registers
                                ; stamped empty and TOP field=0.
    FWAIT                       ; Wait for the store to finish.
    OR      temp_env+2, ax      ; Put in the desired TOP value.
    FLDENV  temp_env            ; Setup new NPX environment.
                                ; Now enter small_block_NPX_restore
```

small_block_NPX_restore

```
;
; Restore the NPX context without using the FRSTOR instruction. Assume the NPX context is in the
; same form as that created by an FSAVE/FNSAVE instruction, all the registers are empty, and that
; the TOP field of the NPX matches the TOP field of the NPX context. The variable save_area must
; be an NPX context save area, 47 words long. The registers bx and cx will not be transparent.
;
small_block_NPX_restore:
        MOV     cx, 8                       ; Set register count
        MOV     bx, type saved_reg*7        ; Starting offset of ST(7)
reg_load_loop:
        FLD     saved_reg [bx]              ; Get the register
        SUB     bx, type saved_reg          ; Bump pointer to next register
        LOOP    reg_load_loop
        FLDENV  save_area                   ; Restore NPX context
                                            ; All done
```

APPENDIX C

OVERVIEW

Appendix C shows how floating point values can be converted to decimal ASCII character strings. The function can be called from PLM/86, PASCAL/86, FORTRAN/86, or ASM/86 functions.

Shortness, speed, and accuracy were chosen rather than providing the maximum number of significant digits possible. An attempt is made to keep integers in their own domain to avoid unnecessary conversion errors.

Using the extended precision real number format, this routine achieves a worst case accuracy of three units in the 16th decimal position for a non-integer value or integers greater than 10^{18}. This is double precision accuracy. With values having decimal exponents less than 100 in magnitude, the accuracy is one unit in the 17th decimal position.

Higher precision can be achieved with greater care in programming, larger program size, and lower performance.

Function Partitioning

Three separate modules implement the conversion. Most of the work of the conversion is done in the module FLOATING_TO_ASCII. The other modules are provided separately since they have a more general use. One of them, GET_POWER_10, is also used by the ASCII to floating point conversion routine. The other small module, TOS_STATUS, will identify what, if anything, is in the top of the numeric register stack.

Exception Considerations

Care is taken inside the function to avoid generating exceptions. Any possible numeric value will be accepted. The only exceptions possible would occur if insufficient space exists on the numeric register stack.

The value passed in the numeric stack is checked for existence, type (NAN or infinity), and status (unnormal, denormal, zero, sign). The string size is tested for a minimum and maximum value. If the top of the register stack is empty, or the string size is too small, the function will return with an error code.

Overflow and underflow is avoided inside the function for very large or very small numbers.

Special Instructions

The functions demonstrate the operation of several numeric instructions, different data types, and precision control. Shown are instructions for automatic conversion to BCD, calculating the value of 10 raised to an integer value, establishing and maintaining concurrency, data synchronization, and use of directed rounding on the NPX.

Without the extended precision data type and built-in exponential function, the double precision accuracy of this function could not be attained with the size and speed of the shown example.

The function relies on the numeric BCD data type for conversion from binary floating point to decimal. It is not difficult to unpack the BCD digits into separate ASCII decimal digits. The major work involves scaling the floating point value to the comparatively limited range of BCD values. To print a 9-digit result requires accurately scaling the given value to an integer between 10^8 and 10^9. For example, the number +0.123456789 requires a scaling factor of 10^9 to produce the value +123456789.0 which can be stored in 9 BCD digits. The scale factor must be an exact power of 10 to avoid to changing any of the printed digit values.

These routines should exactly convert all values exactly representable in decimal in the field size given. Integer values which fit in the given string size, will not be scaled, but directly stored into the BCD form. Non-integer values exactly representable in decimal within the string size limits will also be exactly converted. For example, 0.125 is exactly representable in binary or decimal. To convert this floating point value to decimal, the scaling factor will be 1000, resulting in 125. When scaling a value, the function must keep track of where the decimal point lies in the final decimal value.

DESCRIPTION OF OPERATION

Converting a floating point number to decimal ASCII takes three major steps: identifying the magnitude of the number, scaling it for the BCD data type, and converting the BCD data type to a decimal ASCII string.

Identifying the magnitude of the result requires finding the value X such that the number is represented by $I*10^X$, where $1.0 <= I < 10.0$. Scaling the number requires multiplying it by a scaling factor 10^S, such that the result is an integer requiring no more decimal digits than provided for in the ASCII string.

Once scaled, the numeric rounding modes and BCD conversion put the number in a form easy to convert to decimal ASCII by host software.

Implementing each of these three steps requires attention to detail. To begin with, not all floating point values have a numeric meaning. Values such as infinity, indefinite, or Not A Number (NAN) may be encountered by the conversion routine. The conversion routine should recognize these values and identify them uniquely.

Special cases of numeric values also exist. Denormals, unnormals, and pseudo zero all have a numeric value but should be recognized since all of them indicate that precision was lost during some earlier calculations.

Once it has been determined that the number has a numeric value, and it is normalized setting appropriate unnormal flags, the value must be scaled to the BCD range.

Scaling the Value

To scale the number, its magnitude must be determined. It is sufficient to calculate the magnitude to an accuracy of 1 unit, or within a factor of 10 of the given value. After scaling the number, a check will be made to see if the result falls in the range expected. If not, the result can be adjusted one decimal order of magnitude up or down. The adjustment test after the scaling is necessary due to inevitable inaccuracies in the scaling value.

Since the magnitude estimate need only be close, a fast technique is used. The magnitude is estimated by multiplying the power of 2, the unbiased floating point exponent, associated with the number by $\log_{10}2$. Rounding the result to an integer will produce an estimate of sufficient accuracy. Ignoring the fraction value can introduce a maximum error of 0.32 in the result.

Using the magnitude of the value and size of the number string, the scaling factor can be calculated. Calculating the scaling factor is the most inaccurate operation of the conversion process. The relation $10^X = 2**(X*\log_2 10)$ is used for this function. The exponentiate instruction (F2XM1) will be used.

Due to restrictions on the range of values allowed by the F2XM1 instruction, the power of 2 value will be split into integer and fraction components. The relation $2**(I+F) = 2**I * 2**F$ allows using the FSCALE instruction to recombine the 2**F value, calculated through F2XM1, and the 2**I part.

Inaccuracy in Scaling

The inaccuracy of these operations arises because of the trailing zeroes placed into the fraction value when stripping off the integer valued bits. For each integer valued bit in the power of 2 value separated from the fraction bits, one bit of precision is lost in the fraction field due to the zero fill occurring in the least significant bits.

Up to 14 bits may be lost in the fraction since the largest allowed floating point exponent value is $2^{14} - 1$.

AVOIDING UNDERFLOW AND OVERFLOW

The fraction and exponent fields of the number are separated to avoid underflow and overflow in calculating the scaling values. For example, to scale 10^{-4932} to 10^8 requires a scaling factor of 10^{4950} which cannot be represented by the NPX.

By separating the exponent and fraction, the scaling operation involves adding the exponents separate from multiplying the fractions. The exponent arithmetic will involve small integers, all easily represented by the NPX.

FINAL ADJUSTMENTS

It is possible that the power function (Get_Power_10) could produce a scaling value such that it forms a scaled result larger than the ASCII field could allow. For example, scaling 9.9999999999999999e4900 by 1.00000000000000010e-4883 would produce 1.00000000000000009e18. The scale factor is within the accuracy of the NDP and the result is within the conversion accuracy, but it cannot be represented in BCD format. This is why there is a post-scaling test on the magnitude of the result. The result can be multiplied or divided by 10, depending on whether the result was too small or too large, respectively.

Output Format

For maximum flexibility in output formats, the position of the decimal point is indicated by a binary integer called the power value. If the power value is zero, then the decimal point is assumed to be at the right of the right-most digit. Power values greater than zero indicate how many trailing zeroes are not shown. For each unit below zero, move the decimal point to the left in the string.

The last step of the conversion is storing the result in BCD and indicating where the decimal point lies. The BCD string is then unpacked into ASCII decimal characters. The ASCII sign is set corresponding to the sign of the original value.

```
LINE    SOURCE

$title(Convert a floating point number to ASCII)
                name    floating_to_ascii
                public  floating_to_ascii
                extrn   get_power_10:near,tos_status:near
;
;         This subroutine will convert the floating point number in the
;       top of the 8087 stack to an ASCII string and separate power of 10
;       scaling value (in binary).  The maximum width of the ASCII string
;       formed is controlled by a parameter which must be > 1.  Unnormal values,
;       denormal values, and psuedo zeroes will be correctly converted.
;       A returned value will indicate how many binary bits of
;       precision were lost in an unnormal or denormal value.  The magnitude
;       (in terms of binary power) of a psuedo zero will also be indicated.
;       Integers less than 10**18 in magnitude are accurately converted if the
;       destination ASCII string field is wide enough to hold all the
;       digits.  Otherwise the value is converted to scientific notation.
;
;         The status of the conversion is identified by the return value,
;       it can be:
;
;               0       conversion complete, string_size is defined
;               1       invalid arguments
;               2       exact integer conversion, string_size is defined
;               3       indefinite
;               4       + NAN (Not A Number)
;               5       - NAN
;               6       + Infinity
;               7       - Infinity
;               8       psuedo zero found, string_size is defined
;
;         The PLM/86 calling convention is:
;
; floating_to_ascii:
;       procedure (number,denormal_ptr,string_ptr,size_ptr,field_size,
;               power_ptr) word external;
;       declare (denormal_ptr,string_ptr,power_ptr,size_ptr) pointer;
;       declare field_size word, string_size based size_ptr word;
;       declare number real;
;       declare denormal integer based denormal_ptr;
;       declare power integer based power_ptr;
;       end floating_to_ascii;
;
;         The floating point value is expected to be on the top of the NPX
;       stack.  This subroutine expects 3 free entries on the NPX stack and
;       will pop the passed value off when done.  The generated ASCII string
;       will have a leading character either '-' or '+' indicating the sign
;       of the value.  The ASCII decimal digits will immediately follow.
;       The numeric value of the ASCII string is (ASCII STRING.)*10**POWER.
```

```
;       If the given number was zero, the ASCII string will contain a sign
;       and a single zero chacter.  The value string_size indicates the total
;       length of the ASCII string including the sign character.  String(0) will
;       always hold the sign.  It is possible for string_size to be less than
;       field_size.  This occurs for zeroes or integer values.  A psuedo zero
;       will return a special return code.  The denormal count will indicate
;       the power of two originally associated with the value.  The power of
;       ten and ASCII string will be as if the value was an ordinary zero.
;
;         This subroutine is accurate up to a maximum of 18 decimal digits for
;       integers.  Integer values will have a decimal power of zero associated
;       with them.  For non integers, the result will be accurate to within 2
;       decimal digits of the 16th decimal place (double precision).  The
;       exponentiate instruction is also used for scaling the value into the
;       range acceptable for the BCD data type.  The rounding mode in effect
;       on entry to the subroutine is used for the conversion.
;
;         The following registers are not transparent:
;
;               ax bx cx dx si di flags
;


;
;         Define the stack layout.
;
bp_save         equu    word ptr [bp]
es_save         equ     bp_save + size bp_save
return_ptr      equ     es_save + size es_save
power_ptr       equ     return_ptr + size return_ptr
field_size      equ     power_ptr + size power_ptr
size_ptr        equ     field_size + size field_size
string_ptr      equ     size_ptr + size size_ptr
denormal_ptr    equ     string_ptr + size string_ptr

parms_size      equ     size power_ptr + size field_size + size size_ptr +
&                       size string_ptr + size denormal_ptr
;
;         Define constants used
;
BCD_DIGITS      equ     18              ; Number of digits in bcd_value
WORD_SIZE       equ     2
BCD_SIZE        equ     10
MINUS           equ     1               ; Define return values
NAN             equ     4               ; The exact values chosen here are
INFINITY        equ     6               ; important.  They must correspond to
INDEFINITE      equ     3               ; the possible return values and be in
PSUEDO_ZERO     equ     8               ; the same numeric order as tested by
INVALID         equ     -2              ; the program.
ZERO            equ     -4
DENORMAL        equ     -6
UNNORMAL        equ     -8
NORMAL          equ     0
EXACT           equ     2
;
;         Define layout of temporary storage area.
;
status          equ     word ptr [bp-WORD_SIZE]
power_two       equ     status - WORD_SIZE
power_ten       equ     power_two - WORD_SIZE
bcd_value       equ     tbyte ptr power_ten - BCD_SIZE
bcd_byte        equ     byte ptr bcd_value
fraction        equ     bcd_value

local_size      equ     size status + size power_two + size power_ten
&                       + size bcd_value
;
;         Allocate stack space for the temporaries so the stack will be big enough
;
stack           segment stack 'stack'
                db      (local_size+6) dup (?)


stack           ends
```

```
cgroup          group   code
code            segment public 'code'
                assume  cs:cgroup
                extrn   power_table:qword
;
;       Constants used by this function.
;
                even                    ; Optimize for 16 bits
const10         dw      10              ; Adjustment value for too big BCD
;
;       Convert the C3,C2,C1,C0 encoding from tos_status into meaningful bit
;     flags and values.
;
status_table    db      UNNORMAL, NAN, UNNORMAL + MINUS, NAN + MINUS,
&                       NORMAL, INFINITY, NORMAL + MINUS, INFINITY + MINUS,
&                       ZERO, INVALID, ZERO + MINUS, INVALID,
&                       DENORMAL, INVALID, DENORMAL + MINUS, INVALID

floating_to_ascii proc

        call    tos_status              ; Look at status of ST(0)
        mov     bx,ax                   ; Get descriptor from table
        mov     al,status_table[bx]
        cmp     al,INVALID              ; Look for empty ST(0)
        jne     not_empty
;
;       ST(0) is empty!  Return the status value.
;
        ret     parms_size
;
;       Remove infinity from stack and exit.
;
found_infinity:

        fstp    st(0)                   ; OK to leave fstp running
        jmp     short exit_proc
;
;       String space is too small!  Return invalid code.
;
small_string:

        mov     al,INVALID

exit_proc:

        mov     sp,bp                   ; Free stack space
        pop     bp                      ; Restore registers
        pop     es
        ret     parms_size
;
;       ST(0) is NAN or indefinite.  Store the value in memory and look
;     at the fraction field to separate indefinite from an ordinary NAN.
;
NAN_or_indefinite:

        fstp    fraction                ; Remove value from stack for examination
        test    al,MINUS                ; Look at sign bit
        fwait                           ; Insure store is done
        jz      exit_proc               ; Can't be indefinite if positive
```

```
        mov     bx,0C000H               ; Match against upper 16 bits of fraction
        sub     bx,word ptr fraction+6  ; Compare bits 63-48
        or      bx,word ptr fraction+4  ; Bits 32-47 must be zero
        or      bx,word ptr fraction+2  ; Bits 31-16 must be zero
        or      bx,word ptr fraction    ; Bits 15-0 must be zero
        jnz     exit_proc

        mov     al,INDEFINITE           ; Set return value for indefinite value
        jmp     exit_proc
;
;         Allocate stack space for local variables and establish parameter
;       addressibility.
;
not_empty:

        push    es                      ; Save working register
        push    bp
        mov     bp,sp                   ; Establish stack addressibility
        sub     sp,local_size

        mov     cx,field_size           ; Check for enough string space
        cmp     cx,2
        jl      small_string

        dec     cx                      ; Adjust for sign character
        cmp     cx,BCD_DIGITS           ; See if string is too large for BCD
        jbe     size_ok

        mov     cx,BCD_DIGITS           ; Else set maximum string size

size_ok:

        cmp     al,INFINITY             ; Look for infinity
        jge     found_infinity          ; Return status value for + or - inf.

        cmp     al,NAN                  ; Look for NAN or INDEFINITE
        jge     NAN_or_indefinite
;
;         Set default return values and check that the number is normalized.
;
        fabs                            ; Use positive value only
                                        ; sign bit in al has true sign of value
        mov     dx,ax                   ; Save return value for later
        xor     ax,ax                   ; Form 0 constant
        mov     di,denormal_ptr         ; Zero denormal count
        mov     word ptr [di],ax
        mov     bx,power_ptr            ; Zero power of ten value
        mov     word ptr [bx],ax
        cmp     dl,ZERO                 ; Test for zero
        jae     real_zero               ; Skip power code if value is zero

        cmp     dl,DENORMAL             ; Look for a denormal value
        jae     found_denormal          ; Handle it specially

        fxtract                         ; Separate exponent from significand
        cmp     dl,UNNORMAL             ; Test for unnormal value
        jb      normal_value

        sub     dl,UNNORMAL-NORMAL      ; Return normal status with correct sign
;
;         Normalize the fraction, adjust the power of two in ST(1) and set
;       the denormal count value.
;
;       Assert: 0 <= ST(0) < 1.0
;
        fld1                            ; Load constant to normalize fraction

normalize_fraction:

        fadd    st(1),st                ; Set integer bit in fraction
        fsub                            ; Form normalized fraction in ST(0)
        fxtract                         ; Power of two field will be negative
                                        ; of denormal count
        fxch                            ; Put denormal count in ST(0)
```

```
        fist    word ptr [di]           ; Put negative of denormal count in memory
        faddp   st(2),st                ; Form correct power of two in st(1)
                                        ; OK to use word ptr [di] now
        neg     word ptr [di]           ; Form positive denormal count
        jnz     not_psuedo_zero
;
;         A psuedo zero will appear as an unnormal number.  When attempting
;       to normalize it, the resultant fraction field will be zero.  Performing
;       an fxtract on zero will yield a zero exponent value.
;
        fxch                            ; Put power of two value in st(0)
        fistp   word ptr [di]           ; Set denormal count to power of two value
                                        ; Word ptr [di] is not used by convert
                                        ; integer, OK to leave running
        sub     dl,NORMAL-PSUEDO_ZERO   ; Set return value saving the sign bit
        jmp     convert_integer         ; Put zero value into memory
;
;         The number is a real zero, set the return value and setup for
;       conversion to BCD.
;
real_zero:

        sub     dl,ZERO-NORMAL          ; Convert status to normal value
        jmp     convert_integer         ; Treat the zero as an integer
;
;         The number is a denormal.  FXTRACT will not work correctly in this
;       case.  To correctly separate the exponent and fraction, add a fixed
;       constant to the exponent to guarantee the result is not a denormal.
;
found_denormal:

        fld1                            ; Prepare to bump exponent
        fxch
        fprem                           ; Force denormal to smallest representable
                                        ; extended real format exponent
        fxtract                         ; This will work correctly now
;
;         The power of the original denormal value has been safely isolated.
;       Check if the fraction value is an unnormal.
;
        fxam                            ; See if the fraction is an unnormal
        fstsw   status                  ; Save status for later
        fxch                            ; Put exponent in ST(0)
        fxch    st(2)                   ; Put 1.0 into ST(0), exponent in ST(2)
        sub     dl,DENORMAL-NORMAL      ; Return normal status with correct sign
        test    status,4400H            ; See if C3=C2=0 impling unnormal or NAN
        jz      normalize_fraction      ; Jump if fraction is an unnormal

        fstp    st(0)                   ; Remove unnecessary 1.0 from st(0)
;
;         Calculate the decimal magnitude associated with this number to
;       within one order.  This error will always be inevitable due to
;       rounding and lost precision.  As a result, we will deliberately fail
;       to consider the LOG10 of the fraction value in calculating the order.
;       Since the fraction will always be 1 <= F < 2, its LOG10 will not change
;       the basic accuracy of the function.  To get the decimal order of magnitude,
;       simply multiply the power of two by LOG10(2) and truncate the result to
;       an integer.
;
normal_value:
not_psuedo_zero:

        fstp    fraction                ; Save the fraction field for later use
        fist    power_two               ; Save power of two
        fldlg2                          ; Get LOG10(2)
                                        ; Power_two is now safe to use
        fmul                            ; Form LOG10(of exponent of number)
        fistp   power_ten               ; Any rounding mode will work here
;
;         Check if the magnitude of the number rules out treating it as
;       an integer.
;
;       CX has the maximum number of decimal digits allowed.
```

```
;
        fwait                               ; Wait for power_ten to be valid
        mov     ax,power_ten                ; Get power of ten of value
        sub     ax,cx                       ; Form scaling factor necessary in ax
        ja      adjust_result               ; Jump if number will not fit
;
;         The number is between 1 and 10**(field_size).
;       Test if it is an integer.
;
        fild    power_two                   ; Restore original number
        mov     si,dx                       ; Save return value
        sub     dl,NORMAL-EXACT             ; Convert to exact return value
        fld     fraction
        fscale                              ; Form full value, this is safe here
        fst     st(1)                       ; Copy value for compare
        frndint                             ; Test if its an integer
        fcomp                               ; Compare values
        fstsw   status                      ; Save status
        test    status,4000H                ; C3=1 implies it was an integer
        jnz     convert_integer

        fstp    st(0)                       ; Remove non integer value
        mov     dx,si                       ; Restore original return value
;
;         Scale the number to within the range allowed by the BCD format.
;       The scaling operation should produce a number within one decimal order
;       of magnitude of the largest decimal number representable within the
;       given string width.
;
;         The scaling power of ten value is in ax.
;
adjust_result:

        mov     word ptr [bx],ax            ; Set initial power of ten return value
        neg     ax                          ; Subtract one for each order of
                                            ; magnitude the value is scaled by
        call    get_power_10                ; Scaling factor is returned as exponent
                                            ; and fraction
        fld     fraction                    ; Get fraction
        fmul                                ; Combine fractions
        mov     si,cx                       ; Form power of ten of the maximum
        shl     si,1                        ; BCD value to fit in the string
        shl     si,1                        ; Index in si
        shl     si,1
        fild    power_two                   ; Combine powers of two
        faddp   st(2),st
        fscale                              ; Form full value, exponent was safe
        fstp    st(1)                       ; Remove exponent
;
;         Test the adjusted value against a table of exact powers of ten.
;       The combined errors of the magnitude estimate and power function can
;       result in a value one order of magnitude too small or too large to fit
;       correctly in the BCD field.  To handle this problem, pretest the
;       adjusted value, if it is too small or large, then adjust it by ten and
;       adjust the power of ten value.
;
test_power:

        fcom    power_table[si]+type power_table; Compare against exact power
                                            ; entry.  Use the next entry since cx
                                            ; has been decremented by one
        fstsw   status                      ; No wait is necessary
        test    status,4100H                ; If C3 = C0 = 0 then too big
        jnz     test_for_small

        fidiv   const10                     ; Else adjust value
        and     dl,not EXACT                ; Remove exact flag
        inc     word ptr [bx]               ; Adjust power of ten value
        jmp     short in_range              ; Convert the value to a BCD integer

test_for_small:

        fcom    power_table[si]             ; Test relative size
        fstsw   status                      ; No wait is necessary
```

```
          test    status,100H             ; If C0 = 0 then st(0) >= lower bound
          jz      in_range                ; Convert the value to a BCD integer

          fimul   const10                 ; Adjust value into range
          dec     word ptr [bx]           ; Adjust power of ten value

in_range:

          frndint                         ; Form integer value
;
;         Assert: 0 <= TOS <= 999,999,999,999,999,999
;         The TOS number will be exactly representable in 18 digit BCD format.
;
convert_integer:

          fbstp   bcd_value               ; Store as BCD format number
;
;           While the store BCD runs, setup registers for the conversion to
;         ASCII.
;
          mov     si,BCD_SIZE-2           ; Initial BCD index value
          mov     cx,0f04h                ; Set shift count and mask
          mov     bx,1                    ; Set initial size of ASCII field for sign
          mov     di,string_ptr           ; Get address of start of ASCII string
          mov     ax,ds                   ; Copy ds to es
          mov     es,ax
          cld                             ; Set autoincrement mode
          mov     al,'+'                  ; Clear sign field
          test    dl,MINUS                ; Look for negative value
          jz      positive_result

          mov     al,'-'

positive_result:

          stosb                           ; Bump string pointer past sign
          and     dl,not MINUS            ; Turn off sign bit
          fwait                           ; Wait for fbstp to finish
;
;           Register usage:
;                                     ah:     BCD byte value in use
;                                     al:     ASCII character value
;                                     dx:     Return value
;                                     ch:     BCD mask = 0fh
;                                     cl:     BCD shift count = 4
;                                     bx:     ASCII string field width
;                                     si:     BCD field index
;                                     di:     ASCII string field pointer
;                                     ds,es:  ASCII string segment base
;
;           Remove leading zeroes from the number.
;
skip_leading_zeroes:

          mov     ah,bcd_byte[si]         ; Get BCD byte
          mov     al,ah                   ; Copy value
          shr     al,cl                   ; Get high order digit
          and     al,ch                   ; Set zero flag
          jnz     enter_odd               ; Exit loop if leading non zero found

          mov     al,ah                   ; Get BCD byte again
          and     al,ch                   ; Get low order digit
          jnz     enter_even              ; Exit loop if non zero digit found

          dec     si                      ; Decrement BCD index
          jns     skip_leading_zeroes
;
;           The significand was all zeroes.
;
          mov     al,'0'                  ; Set initial zero
          stosb
          inc     bx                      ; Bump string length
          jmp     short exit_with_value
```

```
;
;          Now expand the BCD string into digit per byte values 0-9.
;
digit_loop:
          mov      ah,bcd_byte[si]          ; Get BCD byte
          mov      al,ah
          shr      al,cl                    ; Get high order digit
enter_odd:
          add      al,'0'                   ; Convert to ASCII
          stosb                             ; Put digit into ASCII string area
          mov      al,ah                    ; Get low order digit
          and      al,ch
          inc      bx                       ; Bump field size counter
enter_even:
          add      al,'0'                   ; Convert to ASCII
          stosb                             ; Put digit into ASCII area
          inc      bx                       ; Bump field size counter
          dec      si                       ; Go to next BCD byte
          jns      digit_loop
;
;          Conversion complete.  Set the string size and remainder.
;
exit_with_value:
          mov      di,size_ptr
          mov      word ptr [di],bx
          mov      ax,dx                    ; Set return value
          jmp      exit_proc
floating_to_ascii         endp
code                      ends
                          end
```

ASSEMBLY COMPLETE, NO ERRORS FOUND

```
LINE      SOURCE

$title(Calculate the value of 10**ax)
;
;          This subroutine will calculate the value of 10**ax.
;         All 8086 registers are transparent and the value is returned on
;         the TOS as two numbers, exponent in ST(1) and fraction in ST(0).
;         The exponent value can be larger than the maximum representable
;         exponent.  Three stack entries are used.
;
                    name     get_power_10
                    public   get_power_10,power_table
stack               segment stack   'stack'
                    dw       4 dup (?)              ; Allocate space on the stack

stack               ends
cgroup              group    code
code                segment public 'code'
                    assume   cs:cgroup
;
;          Use exact values from 1.0 to 1e18.
;
                    even                            ; Optimize 16 bit access
power_table         dq       1.0,1e1,1e2,1e3
```

```
                dq      1e4,1e5,1e6,1e7

                dq      1e8,1e9,1e10,1e11

                dq      1e12,1e13,1e14,1e15

                dq      1e16,1e17,1e18


get_power_10    proc

        cmp     ax,18                   ; Test for 0 <= ax < 19
        ja      out_of_range

        push    bx                      ; Get working index register
        mov     bx,ax                   ; Form table index
        shl     bx,1
        shl     bx,1
        shl     bx,1
        fld     power_table[bx]         ; Get exact value
        pop     bx                      ; Restore register value
        fxtract                         ; Separate power and fraction
        ret                             ; OK to leave fxtract running
;
;         Calculate the value using the exponentiate instruction.
;       The following relations are used:
;               10**x = 2**(log2(10)*x)
;               2**(I+F) = 2**I * 2**F
;               if st(1) = I and st(0) = 2**F then fscale produces 2**(I+F)
;
out_of_range:

        fldl2t                          ; TOS = LOG2(10)
        push    bp                      ; Establish stack addressibility
        mov     bp,sp
        push    ax                      ; Put power (P) in memory
        push    ax                      ; Allocate space for status
        fimul   word ptr [bp-2]         ; TOS,X = LOG2(10)*P = LOG2(10**P)
        fnstcw  word ptr [bp-4]         ; Get current control word
                                        ; Control word is a static value
        mov     ax,word ptr [bp-4]      ; Get control word, no wait necessary
        and     ax,not 0C00H            ; Mask off current rounding field
        or      ax,0400H                ; Set round to negative infinity
        xchg    ax,word ptr [bp-4]      ; Put new control word in memory
                                        ; old control word is in ax
        fld1                            ; Set TOS = -1.0
        fchs
        fld     st(1)                   ; Copy power value in base two
        fldcw   word ptr [bp-4]         ; Set new control word value
        frndint                         ; TOS = I: -inf < I <= X, I is an integer
        mov     word ptr [bp-4],ax      ; Restore original rounding control
        fldcw   word ptr [bp-4]
```

```
             fxch    st(2)                       ; TOS = X, ST(1) = -1.0, ST(2) = I
             pop     ax                          ; Remove original control word
             fsub    st,st(2)                    ; TOS,F = X-I: 0 <= TOS < 1.0
             pop     ax                          ; Restore power of ten
             fscale                              ; TOS = F/2: 0 <= TOS < 0.5
             f2xm1                               ; TOS = 2**(F/2) - 1.0
             pop     bp                          ; Restore stack
             fsubr                               ; Form 2**(F/2)
             fmul    st,st(0)                    ; Form 2**F
             ret                                 ; OK to leave fmul running

get_power_10         endp
code                 ends
                     end
```

ASSEMBLY COMPLETE, NO ERRORS FOUND

```
LINE    SOURCE

$title(Determine TOS register contents)
;
;        This subroutine will return a value from 0-15 in ax corresponding
;       to the contents of 8087 TOS.  All registers are transparent and no
;       errors are possible.  The return value corresponds to c3,c2,c1,c0
;       of FXAM instruction.
;
        name    tos_status
        public  tos_status

stack           segment stack 'stack'
                dw      3 dup (?)       ; Allocate space on the stack


stack           ends

cgroup          group   code
code            segment public 'code'
                assume  cs:cgroup
tos_status      proc

        fxam                    ; Get register contents status
        push    ax              ; Allocate space for status value
        push    bp              ; Establish stack addressibility
        mov     bp,sp
        fstsw   word ptr [bp+2] ; Put tos status in memory
        pop     bp              ; Restore registers
        pop     ax              ; Get status value, no wait necessary
        mov     al,ah           ; Put bit 10-8 into bits 2-0
        and     ax,4007h        ; Mask out bits c3,c2,c1,c0
        shr     ah,1            ; Put bit c3 into bit 11
        shr     ah,1
        shr     ah,1
        or      al,ah           ; Put c3 into bit 3
        mov     ah,0            ; Clear return value
        ret

tos_status      endp
code            ends
                end
```

ASSEMBLY COMPLETE, NO ERRORS FOUND

APPENDIX D

OVERVIEW

Appendix D shows a function for converting ASCII input strings into floating point values. The returned value can be used by PLM/86, PASCAL/86, FORTRAN/86, or ASM/86. The routine will accept a number in ASCII of standard FORTRAN formats. Up to 18 decimal digits are accepted and the conversion accuracy is the same as for converting in the other direction. Greater accuracy can also be achieved with similar tradeoffs, as mentioned earlier.

Description of Operation

Converting from ASCII to floating point is less complex numerically than going from floating point to ASCII. It consists of four basic steps: determine the size in decimal digits of the number, build a BCD value corresponding to the number string if the decimal point were at the far right, calculate the exponent value, and scale the BCD value. The first three steps are performed by the host software. The fourth step is mainly performed by numeric operations.

The complexity in this function arises due to the flexible nature of the input values it will recognize. Most of the code simply determines the meaning of each character encountered. Two separate number inputs must be recognized, mantissa and exponent values. Performing the numerics operations is very straightforward.

The length of the number string is determined first to allow building a BCD number from low digits to high digits. This technique guarantees that an integer will be converted to its exact BCD integer equivalent.

If the number is a floating point value, then the digit string can be scaled appropriately. If a decimal point occurs within the string, the scale factor must be decreased by one for each digit the decimal point is moved to the right. This factor must be added to any exponent value specified in the number.

ACCURACY CONSIDERATIONS

All the same considerations for converting floating point to ASCII apply to calculating the scaling factor. The accuracy of the scale factor determines the accuracy of the result.

The exponents and fractions are again kept separate to prevent overflows or underflows during the scaling operations.

LINE SOURCE

```
$title(ASCII to floating point conversion)
;
;       Define the publicly known names.
;
                name    ascii_to_floating
                public  ascii_to_floating
                extrn   get_power_10:near
;
;         This function will convert an ASCII character string to a floating
;       point representation.  Character strings in integer or scientific form
;       will be accepted.  The allowed format is:
;
;       [+,-][digit(s)][.][digit(s)][E,e][+,-][digit(s)]
;
;         Where a digit must have been encountered before the exponent
;       indicator 'E' or'e'.  If a '+', '-', or '.' was encountered, then at
;       least one digit must exist before the optional exponent field.  A value
;       will always be returned in the 8087 stack.  In case of invalid numbers,
;       values like indefinite or infinity will be returned.
;
;         The first character not fitting within the format will terminate the
;       conversion.  The address of the terminating character will be returned
;       by this subroutine.
;
;         The result will be left on the top of the NPX stack.  This
;       subroutine expects 3 free NPX stack registers.  The sign of the result
;       will correspond to any sign characters in the ASCII string.  The rounding
;       mode in effect at the time the subroutine was called will be used for
;       the conversion from base 10 to base 2.  Up to 18 significant decimal
;       digits may appear in the number.  Leading zeroes, trailing zeroes, or
;       exponent digits do not count towards the 18 digit maximum.  Integers
;       or exactly representable decimal numbers of 18 digits or less will be
;       exactly converted.  The technique used constructs a BCD number
```

```
;       representing the significant ASCII digits of the string with the decimal
;       point removed.
;
;         An attempt is made to exactly convert relatively small integers or
;       small fractions.  For example the values: .06125, 123456789012345678,
;       1e17, 1.23456e5, and 125e-3 will be exactly converted to floating point.
;         The exponentiate instruction is used to scale the generated BCD vaslue
;       to very large or very small numbers.  The basic accuracy of this function
;       determines the accuracy of this subroutine.  For very large or very small
;       numbers, the accuracy of this function is 2 units in the 16th decimal
;       place or double precision.  The range of decimal powers accepted is
;       10**-4930 to 10**4930.
;
;         The PLM/86 calling format is:
;
;       ascii_to_floating:
;               procedure (string_ptr,end_ptr,status_ptr) real external;
;               declare (string_ptr,end_ptr,status_ptr) pointer;
;               declare end based end_ptr pointer;
;               declare status based status_ptr word;
;               end;
;
;         The status value has 6 possible states:
;
;               0       A number was found.
;               1       No number was found, return indefinite.
;               2       Exponent was expected but none found, return indefinite.
;               3       Too many digits were found, return indefinite.
;               4       Exponent was too big, return a signed infinity.
;
;         The following registers are used by this subroutine:
;
;       ax      bx      cx      dx      si      di
;


;
;         Define constants.
;
LOW_EXPONENT    equ     -4930                   ; Smallest allowed power of 10
HIGH_EXPONENT   equ     4930                    ; Largest allowed power of 10
WORD_SIZE       equ     2
BCD_SIZE        equ     10
;
;         Define the parameter layouts involved:
;
bp_save         equ     word ptr [bp]
return_ptr      equ     bp_save + size bp_save
status_ptr      equ     return_ptr + size return_ptr
end_ptr         equ     status_ptr + size status_ptr
string_ptr      equ     end_ptr + size end_ptr

parms_size      equ     size status_ptr + size end_ptr + size string_ptr
;
;         Define the local variable data layouts
;
power_ten       equ     word ptr [bp- WORD_SIZE] ; power of ten value
bcd_form        equ     tbyte ptr power_ten - BCD_SIZE; BCD representation

local_size      equ     size power_ten + size bcd_form
;
;         Define common expressions used
;
bcd_byte        equ     byte ptr bcd_form       ; Current byte in the BCD form
bcd_count       equ     (type(bcd_form)-1)*2    ; Number of digits in BCD form
bcd_sign        equ     byte ptr bcd_form + 9   ; Address of BCD sign byte
bcd_sign_bit    equ     80H
;
;         Define return values.
;
NUMBER_FOUND    equ     0               ; Number was found
NO_NUMBER       equ     1               ; No number was found
NO_EXPONENT     equ     2               ; No exponent was found when expected
TOO_MANY_DIGITS equ     3               ; Too many digits were found
EXPONENT_TOO_BIG equ    4               ; Exponent was too big
```

```
;
;           Allocate stack space to insure enough exists at run time.
;
stack               segment stack 'stack'
                    db      (local_size+4) dup (?)


stack               ends

cgroup              group   code
code                segment public 'code'
                    assume  cs:cgroup
;
;           Define some of the possible return values.
;
                    even                        ; Optimize 16 bit access
indefinite          dd      0FFC00000R          ; Single precision real for indefinite
infinity            dd      07FF80000R          ; Single precision real for +infinity

ascii_to_floating proc

            fldz                                ; Prepare to zero BCD value
            push    bp                          ; Save callers stack environment
            mov     bp,sp                       ; Establish stack addressibility
            sub     sp,local_size               ; Allocate space for local variables
;
;           Get any leading sign character to form initial BCD template.
;
            mov     si,string_ptr               ; Get starting address of the number
            xor     dx,dx                       ; Set initial decimal digit count
            cld                                 ; Set autoincrement mode
;
;           Register usage:
;
;                   al:     Current character value being examined
;                   cx:     Digit count before the decimal point
;                   dx:     Total digit count
;                   si:     Pointer to character string
;
;           Look for an initial sign and skip it if found.
;
            lodsb                               ; Get first character
            cmp     al,'+'                      ; Look for a sign
            jz      scan_leading_digits

            cmp     al,'-'
            jnz     enter_leading_digits        ; If not "-" test current character

            fchs                                ; Set TOS = -0
;
;           Count the number of digits appearing before an optional decimal point.
;
scan_leading_digits:

            lodsb                               ; Get next character

enter_leading_digits:

            call    test_digit                  ; Test for digit and bump counter
            jnc     scan_leading_digits
;
;           Look for a possible decimal point and start fbstp operation.
;           The fbstp zeroes out the BCD value and sets the correct sign.
;
            fbstp   bcd_form                    ; Set initial sign and value of BCD number
            mov     cx,dx                       ; Save count of digits before decimal point
            cmp     al,'.'
            jnz     test_for_digits
;
;           Count the number of digits appearing after the decimal point.
;
scan_trailing_digits:

            lodsb                               ; Look at next character
```

```
        call    test_digit              ; Test for digit and bump counter
        jnc     scan_trailing_digits
;
;         There must be at least one digit counted at this point.
;
test_for_digits:

        dec     si                      ; Put si back on terminating character
        or      dx,dx                   ; Test digit count
        jz      no_number_found         ; Jump if no digits were found

        push    si                      ; Save pointer to terminator
        dec     si                      ; Backup pointer to last digit
;
;         Check that the number will fit in the 18 digit BCD format.
;       CX becomes the initial scaling factor to account for the implied
;       decimal point.
;
        sub     cx,dx                   ; For each digit to the right of the
                                        ; decimal point, subtract one from the
                                        ; initial scaling power
        neg     dx                      ; Use negative digit count so the
                                        ; test_digit routine can count dx up
                                        ; to zero
        cmp     dx,-bcd_count           ; See if too many digits found
        jb      test_for_unneeded_digits
;
;         Setup initial register values for scanning the number right to left
;       while building the BCD value in memory.
;
form_bcd_value:

        std                             ; Set autodecrement mode
        mov     power_ten,cx            ; Set initial power of ten
        xor     di,di                   ; Clear BCD number index
        mov     cl,4                    ; Set digit shift count
        fwait                           ; Ensure BCD store is done
        jmp     enter_digit_loop
;
;         No digits were encountered before testing for the exponent.
;       Restore the string pointer and return an indefinite value.
;
no_number_found:

        mov     ax,NO_NUMBER            ; Set return status
        fld     indefinite              ; Return an indefinite numeric value
        jmp     exit
;
;         Test for a number of the form ???00000.
;
test_terminating_point:

        lodsb                           ; Get last character
        cmp     al,'.'                  ; Look for decimal point
        jz      enter_power_zeroes      ; Skip forward if found

        inc     si                      ; Else bump pointer back
        jmp     short enter_power_zeroes
;
;         Too many decimal digits encountered.  Attempt to remove leading and
;       trailing digits to bring the total into the bounds of the BCD format.
;
test_for_unneeded_digits:

        std                             ; Set autodecrement mode
        or      cx,cx                   ; See if any digits appeared to the
                                        ; right of the decimal point
        jz      test_terminating_point  ; Jump if none exist

        dec     dx                      ; Adjust digit counter for loop
;
;         Scan backwards from the right skipping trailing zeroes.
;       If the end of the number is encountered, dx=0, the string consists of
;       all zeroes!
```

```
;
skip_trailing_zeroes:

        inc     dx                          ; Bump digit count
        jz      look_for_exponent           ; Jump if string of zeroes found!

        lodsb                               ; Get next character
        inc     cx                          ; Bump power value for each trailing
        cmp     al,'0'                      ; zero dropped
        jz      skip_trailing_zeroes

        dec     cx                          ; Adjust power counter from loop
        cmp     al,'.'                      ; Look for decimal point
        jnz     scan_leading_zeroes         ; Skip forward if none found

        dec     dx                          ; Adjust counter for the decimal point
;
;         The string is of the form: ????.0000000
;       See if any zeroes exist to the left of the decimal point.
;
enter_power_zeroes:

        dec     dx                          ; Adjust digit counter for loop

skip_power_zeroes:

        inc     dx                          ; Bump digit count
        jz      look_for_exponent

        lodsb                               ; Get next character
        inc     cx                          ; Bump power value for each trailing
        cmp     al,'0'                      ; zero dropped
        jz      skip_power_zeroes

        dec     cx                          ; Adjust power counter from loop
;
;         Scan the leading digits from the left to see if they are zeroes.
;
scan_leading_zeroes:

        lea     di,byte ptr [si+1]          ; Save new end of number pointer
        cld                                 ; Set autoincrement mode
        mov     si,string_ptr               ; Set pointer to the start
        lodsb                               ; Look for sign character
        cmp     al,'+'
        je      skip_leading_zeroes

        cmp     al,'-'
        jne     enter_leading_zeroes
;
;         Drop leading zeroes.  None of them affect the power value in cx.
;       We are guarenteed at least one non zero digit to terminate the loop.
;
skip_leading_zeroes:

        lodsb                               ; Get next character

enter_leading_zeroes:

        inc     dx                          ; Bump digit count
        cmp     al,'0'                      ; Look for a zero
        jz      skip_leading_zeroes

        dec     dx                          ; Adjust digit count from loop
        cmp     al,'.'                      ; Look for 000.??? form
        jnz     test_digit_count
;
;         Number is of the form 000.????
;       Drop all leading zeroes with no effect on the power value.
;
skip_middle_zeroes:

        inc     dx                          ; Remove the digit
        lodsb                               ; Get next character
```

```
        cmp     al,'0'
        jz      skip_middle_zeroes

        dec     dx                      ; Adjust digit count from loop
;
;         All superflous zeroes are removed.  Check if all is well now.
;
test_digit_count:

        cmp     dx,-bcd_count
        jb      too_many_digits_found

        mov     si,di                   ; Restore string pointer
        jmp     form_bcd_value

too_many_digits_found:

        fld     indefinite              ; Set return numeric value
        mov     ax,TOO_MANY_DIGITS      ; Set return flag
        pop     si                      ; Get last address
        jmp     exit
;
;         Build BCD form of the decimal ASCII string from right to left with
;       trailing zeroes and decimal point removed.  Note that the only non
;       digit possible is a decimal point which can be safely ignored.
;       Test_digit will correctly count dx back towards zero to terminate
;       the BCD build function.
;
get_digit_loop:

        lodsb                           ; Get next character
        call    test_digit              ; Check if digit and bump digit count
        jc      get_digit_loop          ; Skip the decimal point if found

        shl     al,cl                   ; Put digit into high nibble
        or      ah,al                   ; Form BCD byte in ah
        mov     bcd_byte[di],ah         ; Put into BCD string
        inc     di                      ; Bump BCD pointer
        or      dx,dx                   ; Check if digit is available
        jz      look_for_exponent

enter_digit_loop:

        lodsb                           ; Get next character
        call    test_digit              ; Check if digit
        jc      enter_digit_loop        ; Skip the decimal point

        mov     ah,al                   ; Save digit
        or      dx,dx                   ; Check if digit is available
        jnz     get_digit_loop

        mov     bcd_byte[di],ah         ; Save last odd digit
;
;         Look for an exponent indicator.
;
look_for_exponent:

        pop     si                      ; Restore string pointer
        cld                             ; Set autoincrement direction
        mov     di,power_ten            ; Get current power of ten
        lodsb                           ; Get next character
        cmp     al,'e'                  ; Look for exponent indication
        je      exponent_found

        cmp     al,'E'
        jne     convert
;
;         An exponent is expected, get its numeric value.
;
exponent_found:

        lodsb                           ; Get next character
        xor     di,di                   ; Clear power variable
        mov     cx,di                   ; Clear exponent sign flag and digit flag
```

```
        cmp     al,'+'                  ; Test for positive sign
        je      skip_power_sign

        cmp     al,'-'                  ; Test for negative sign
        jne     enter_power_loop
;
;         The exponent is negative.
;
        inc     ch                      ; Set exponent sign flag

skip_power_sign:
;
;         Register usage:
;
;               al:     exponent character being examined
;               bx:     return value
;               ch:     exponent sign flag      0 positive, 1 negative
;               cl:     digit flag      0 no digits found, 1 digits found
;               dx:     not usable since test_digit increments it
;               si:     string pointer
;               di:     binary value of exponent
;
;         Scan off exponent digits until a non-digit is encountered.
;
power_loop:

        lodsb                           ; Get next character

enter_power_loop:

        mov     ah,0                    ; Clear ah since ax is added to later
        call    test_digit              ; Test tor a digit
        jc      form_power_value        ; Exit loop if not

        mov     cl,1                    ; Set power digit flag
        sal     di,1                    ; old*2
        add     ax,di                   ; old*2+digit
        sal     di,1                    ; old*4
        sal     di,1                    ; old*8
        add     di,ax                   ; old*10+digit
        cmp     di,HIGH_EXPONENT+bcd_count; Check if exponent is too big
        jna     power_loop
;
;         The exponent is too large.
;
exponent_overflow:

        mov     ax,EXPONENT_TOO_BIG     ; Set return value
        fld     infinity                ; Return infinity
        test    bcd_sign,bcd_sign_bit   ; Return correctly signed infinity
        jz      exit                    ; Jump if not

        fchs                            ; Return -infinity
        jmp     short exit
;
;         No exponent was found.
;
no_exponent_found:

        dec     si                      ; Put si back on terminating character
        mov     ax,NO_EXPONENT          ; Set return value
        fld     indefinite              ; Set number to return
        jmp     short exit
;
;         The string examination is complete.  Form the correct power of ten.
;
form_power_value:

        dec     si                      ; Backup string pointer to terminating
                                        ; character
        rcr     ch,1                    ; Test exponent sign flag
        jnc     positive_exponent

        neg     di                      ; Force exponent negative
```

```
positive_exponent:

        rcr     cl,1                    ; Test exponent digit flag
        jnc     no_exponent_found       ; If zero then no exponent digits were
                                        ; found
        add     di,power_ten            ; Form the final power of ten value
        cmp     di,LOW_EXPONENT         ; Check if the value is in range
        js      exponent_overflow       ; Jump if exponent is too small

        cmp     di,HIGH_EXPONENT
        jg      exponent_overflow

        inc     si                      ; Adjust string pointer
;
;         Convert the base 10 number to base 2.
;       Note:   10**exp = 2**(exp*log2(10))
;
;       di has binary power of ten value to scale the BCD value with.
;
convert:

        dec     si                      ; Bump string pointer back to last character
        mov     ax,di                   ; Set power of ten to calculate
        or      ax,ax                   ; Test for positive or negative value
        js      get_negative_power
;
;         Scale the BCD value by a value >= 1.
;
        call    get_power_10            ; Get the adjustment power of ten
        fbld    bcd_form                ; Get the digits to use
        fmul                            ; Form converged result
        jmp     short done
;
;         Calculate a power of ten value > 1 then divide the BCD value with
;       it.  This technique is more exact than multiplying the BCD value by
;       a fraction since no negative power of ten can be exactly represented
;       in binary floating point.  Using this technique will guarentee exact
;       conversion of values like .5 and .0625.
;
get_negative_power:

        neg     ax                      ; Force positive power
        call    get_power_10            ; Get the adjustment power of ten
        fbld    bcd_form                ; Get the digits to use
        fdivr                           ; Divide fractions
        fxch                            ; Negate scale factor
        fchs
        fxch
;
;         All done, set return values.
;
done:

        fscale                          ; Update exponent of the result
        mov     ax,NUMBER_FOUND         ; Set return value
        fstp    st(1)                   ; Remove the scale factor

exit:

        mov     di,status_ptr           ; Set status of the conversion
        mov     word ptr [di],ax
        mov     di,end_ptr              ; Set ending string address
        mov     word ptr [di],si
        mov     sp,bp                   ; Deallocate local storage area
        pop     bp                      ; Restore caller's environment
        fwait                           ; Insure all loads from memory are done
        ret     parms_size
;
;         Test if the character in al is an ASCII digit.
;       If so then convert to binary, bump cx, and clear the carry flag.
;       Else leave as is and set the carry flag.
```

```
;
test_digit:
        cmp     al,'9'              ; See if a digit
        ja      not_digit

        cmp     al,'0'
        jb      not_digit
;
;         Character is a digit.
;
        inc     dx                  ; Bump digit count
        sub     al,'0'              ; Convert to binary and clear carry flag
        ret
;
;         Character is not a digit.
;
not_digit:
        stc                         ; Leave as is and set the carry flag
        ret

ascii_to_floating endp
code            ends
                end

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

APPENDIX E

OVERVIEW

Appendix E contains three trigonometric functions for sine, cosine, and tangent. All accept a valid angle argument between -2^{62} and $+2^{62}$. They may be called from PLM/86, PASCAL/86, FORTRAN/86 or ASM/86 functions.

They use the partial tangent instruction together with trigonometric identities to calculate the result. They are accurate to within 16 units of the low 4 bits of an extended precision value. The functions are coded for speed and small size, with tradeoffs available for greater accuracy.

FPTAN and FPREM

These trigonometric functions use the FPTAN instruction of the NPX. FPTAN requires that the angle argument be between 0 and PI/4 radians, 0 to 45 degrees. The FPREM instruction is used to reduce the argument down to this range. The low three quotient bits set by FPREM identify which octant the original angle was in.

One FPREM instruction iteration can reduce angles of 10^{18} radians or less in magnitude to PI/4! Larger values can be reduced, but the meaning of the result is questionable since any errors in the least significant bits of that value represent changes of 45 degrees or more in the reduced angle.

Cosine Uses Sine Code

To save code space, the cosine function uses most of the sine function code. The relation sin $(|A| + PI/2) = \cos(A)$ is used to convert the cosine argument into a sine argument. Adding PI/2 to the angle is performed by adding 010_2 to the FPREM quotient bits identifying the argument's octant.

It would be very inaccurate to add PI/2 to the cosine argument if it was very much different from PI/2.

Depending on which octant the argument falls in, a different relation will be used in the sine and tangent functions. The program listings show which relations are used.

For the tangent function, the ratio produced by FPTAN will be directly evaluated. The sine function will use either a sine or cosine relation depending on which octant the angle fell into. On exit these functions will normally leave a divide instruction in progress to maintain concurrency.

If the input angles are of a restricted range, such as from 0 to 45 degrees, then considerable optimization is possible since full angle reduction and octant identification is not necessary.

All three functions begin by looking at the value given to them. Not a number (NAN), infinity, or empty registers must be specially treated. Unnormals need to be converted to normal values before the FPTAN instruction will work correctly. Denormals will be converted to very small unnormals which do work correctly for the FPTAN instruction. The sign of the angle is saved to control the sign of the result.

Within the functions, close attention was paid to maintain concurrent execution of the 8087 and host. The concurrent execution will effectively hide the execution time of the decision logic used in the program.

```
SOURCE

$title(8087 Trignometric Functions)

                        public          sine,cosine,tangent
                        name            trig_functions

$include (:f1:8087.anc)
;
;               Define 8087 word packing in the environment area.
;
cw_87                   record  res871:3,infinity_control:1,rounding_control:2,
&                               precision_control:2,error_enable:1,res872:1,
&                               precision_mask:1,underflow_mask:1,overflow_mask:1,
&                               zero_divide_mask:1,denormal_mask:1,invalid_mask:1

sw_87                   record  busy:1,cond3:1,top:3,cond2:1,cond1:1,cond0:1,
&                               error_pending:1,res873:1,precision_error:1,
&                               underflow_error:1,overflow_error:1,zero_divide_error:1,
&                               denormal_error:1,invalid_error:1

tw_87                   record  reg7_tag:2,reg6_tag:2,reg5_tag:2,reg4_tag:2,
&                               reg3_tag:2,reg2_tag:2,reg1_tag:2,reg0_tag:2

low_ip_87               record  low_ip:16

high_ip_op_87           record  hi_ip:4,res874:1,opcode_87:11

low_op_87               record  low_op:16

high_op_87              record  hi_op:4,res875:12

environment_87          struc                           ; 8087 environemnt layout
env87_cw                dw              ?
env87_sw                dw              ?
env87_tw                dw              ?
env87_low_ip            dw              ?
env87_hip_op            dw              ?
env87_low_op            dw              ?
env87_hop               dw              ?
environment_87          ends
;
;               Define 8087 related constants.
;
TOP_VALUE_INC           equu            sw_87 <0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0>

VALID_TAG               equ             0       ; Tag register values
ZERO_TAG                equ             1
SPECIAL_TAG             equ             2
EMPTY_TAG               equ             3
REGISTER_MASK           equ             7

;
;               Define local variable areas.
;
stack                   segment stack 'stack'

local_area              struc
swl                     dw      ?               ; 8087 status value
local_area              ends

                        db      size local_area+4       ; Allocate stack space
stack                   ends

code                    segment public 'code'
                        assume  cs:code,ss:stack
;
;               Define local constants.
;
status                  equ     [bp].swl                ; 8087 status value location

                        even

pi_quarter              dt      3FFEC90FDAA22168C235R   ; PI/4
```

```
indefinite      dd      0FFC00000R              ; Indefinite special value
;
;         This subroutine calculates the sine or cosine of the angle, given in
;       radians.  The angle is in ST(0), the returned value will be in ST(0).
;       The result is accurate to within 7 units of the least significant three
;       bits of the NPX extended real format.  The PLM/86 definition is:
;
;       sine:   procedure (angle) real external;
;               declare angle real;
;               end sine;
;
;       cosine: procedure (angle) real external;
;               declare angle real;
;               end cosine;
;
;         Three stack registers are required.  The result of the function is
;       defined as follows for the following arguments:
;
;                       angle                                   result
;
;               valid or unnormal less than 2**62 in magnitude  correct value
;               zero                                            0 or 1
;               denormal                                        correct denormal
;               valid or unnormal greater than 2**62            indefinite
;               infinity                                        indefinite
;               NAN                                             NAN
;               empty                                           empty
;
;         This function is based on the NPX fptan instruction.  The fptan
;       instruction will only work with an angle of from 0 to PI/4.  With this
;       instruction, the sine or cosine of angles from 0 to PI/4 can be accurately
;       calculated.  The technique used by this routine can calculate a general
;       sine or cosine by using one of four possible operations:
;
;                       Let R = |angle mod PI/4|
;                           S = -1 or 1, according to the sign of the angle
;
;       1) sin(R)       2) cos(R)       3) sin(PI/4-R)  4) cos(PI/4-R)
;
;         The choice of the relation and the sign of the result follows the
;       decision table shown below based on the octant the angle falls in:
;
;               octant          sine            cosine
;
;                 0              S*1               2
;                 1              S*4               3
;                 2              S*2             -1*1
;                 3              S*3             -1*4
;                 4             -S*1             -1*2
;                 5             -S*4             -1*3
;                 6             -S*2               1
;                 7             -S*3               4
;

;
;         Angle to sine function is a zero or unnormal.
;
sine_zero_unnormal:

        fstp    st(1)                           ; Remove PI/4
        jnz     enter_sine_normalize            ; Jump if angle is unnormal
;
;         Angle is a zero.
;
        pop     bp                              ; Return the zero as the result
        ret
;
;         Angle is an unnormal.
;
enter_sine_normalize:
```

```
        call    normalize_value
        jmp     short enter_sine

cosine  proc                                    ; Entry point to cosine

        fxam                                    ; Look at the value
        push    bp                              ; Establish stack addressibility
        sub     sp,size local_area              ; Allocate stack space for status
        mov     bp,sp
        fstsw   status                          ; Store status value
        fld     pi_quarter                      ; Setup for angle reduce
        mov     cl,1                            ; Signal cosine function
        pop     ax                              ; Get status value
        lahf                                    ; ZF = C3, PF = C2, CF = C0
        jc      funny_parameter                 ; Jump if parameter is
                                                ; empty, NAN, or infinity
;
;         Angle is unnormal, normal, zero, denormal.
;
        fxch                                    ; st(0) = angle, st(1) = PI/4
        jpe     enter_sine                      ; Jump if normal or denormal
;
;         Angle is an unnormal or zero.
;
        fstp    st(1)                           ; Remove PI/4
        jnz     enter_sine_normalize
;
;         Angle is a zero.  cos(0) = 1.0
;
        fstp    st(0)                           ; Remove 0
        pop     bp                              ; Restore stack
        fld1                                    ; Return 1
        ret
;
;         All work is done as a sine function.  By adding PI/2 to the angle
;       a cosine is converted to a sine.  Of course the angle addition is not
;       done to the argument but rather to the program logic control values.
;
sine:                                           ; Entry point for sine function

        fxam                                    ; Look at the parameter
        push    bp                              ; Establish stack addressibility
        sub     sp,size local_area              ; Allocate local space
        mov     bp,sp
        fstsw   status                          ; Look at fxam status
        fld     pi_quarter                      ; Get PI/4 value
        pop     ax                              ; Get fxam status
        lahf                                    ; CF = C0, PF = C2, ZF = C3
        jc      funny_parameter                 ; Jump if empty, NAN, or infinity
;
;          Angle is unnormal, normal, zero, or denormal.
;
        fxch                                    ; ST(1) = PI/4, st(0) angle
        mov     cl,0                            ; Signal sine
        jpo     sine_zero_unnormal              ; Jump if zero or unnormal
;
;         ST(0) is either a normal or denormal value.  Both will work.
;       Use the fprem instruction to accurately reduce the range of the given
;       angle to within 0 and PI/4 in magnitude.  If fprem cannot reduce the
;       angle in one shot, the angle is too big to be meaningful, > 2**62
;       radians.  Any roundoff error in the calculation of the angle given
;       could completely change the result of this function.  It is safest to
;       call this very rare case an error.
;
enter_sine:

        fprem                                   ; Reduce angle
                                                ; Note that fprem will force a
                                                ; denormal to a very small unnormal
                                                ; Fptan of a very small unnormal
                                                ; will be the same very small
                                                ; unnormal, which is correct.
        mov     sp,bp                           ; Allocate stack space for status
        fstsw   status                          ; Check if reduction was complete
```

```
                                                ; Quotient in C0,C3,C1
        pop     bx                              ; Get fprem status
        test    bh,high(mask cond2)             ; sin(2*N*PI+x) = sin(x)
        jnz     angle_too_big
;
;         Set sign flags and test for which eighth of the revolution the
;       angle fell into.
;
;       Assert: -PI/4 < st(0) < PI/4
;
        fabs                                    ; Force the argument positive
                                                ; cond1 bit in bx holds the sign
        or      cl,cl                           ; Test for sine or cosine function
        jz      sine_select                     ; Jump if sine function
;
;         This is a cosine function.  Ignore the original sign of the angle
;       and add a quarter revolution to the octant id from the fprem instruction.
;       cos(A) = sin(A+PI/2) and cos(|A|) = cos(A)
;
        and     ah,not high(mask cond1)         ; Turn off sign of argument
        or      bh,high(mask busy)              ; Prepare to add 010 to C0,C3,C1
                                                ; status value in ax
                                                ; Set busy bit so carry out from
        add     bh,high(mask cond3)             ; C3 will go into the carry flag
        mov     al,0                            ; Extract carry flag
        rcl     al,1                            ; Put carry flag in low bit
        xor     bh,al                           ; Add carry to C0 not changing
                                                ; C1 flag
;
;         See if the argument should be reversed, depending on the octant in
;       which the argument fell during fprem.
;
sine_select:

        test    bh,high(mask cond1)             ; Reverse angle if C1 = 1
        jz      no_sine_reverse
;
;         Angle was in octants 1,3,5,7.
;
        fsub                                    ; Invert sense of rotation
        jmp     short do_sine_fptan             ; 0 < arg <= PI/4
;
;         Angle was in octants 0,2,4,6.
;       Test for a zero argument since fptan will not work if st(0) = 0
;
no_sine_reverse:

        ftst                                    ; Test for zero angle
        mov     sp,bp                           ; Allocate stack space
        fstsw   status                          ; cond3 = 1 if st(0) = 0
        fstp    st(1)                           ; Remove PI/4
        pop     cx                              ; Get ftst status
        test    ch,high(mask cond3)             ; If C3=1, argument is zero
        jnz     sine_argument_zero
;
;       Assert: 0 < st(0) <= PI/4
;
do_sine_fptan:

        fptan                                   ; TAN ST(0) = ST(1)/ST(0) = Y/X

after_sine_fptan:

        pop     bp                              ; Restore stack
        test    bh,high(mask cond3 + mask cond1); Look at octant angle fell into
        jpo     X_numerator                     ; Calculate cosine for octants
                                                ; 1,2,5,6
;
;         Calculate the sine of the argument.
;       sin(A) = tan(A)/sqrt(1+tan(A)**2)       if tan(A) = Y/X then
;       sin(A) = Y/sqrt(X*X + Y*Y)
;
        fld     st(1)                           ; Copy Y value
        jmp     short finish_sine               ; Put Y value in numerator
```

```
;
;          The top of the stack is either NAN, infinity, or empty.
;
funny_parameter:
        fstp    st(0)                         ; Remove PI/4
        jz      return_empty                  ; Return empty if no parm
        jpo     return_NAN                    ; Jump if st(0) is NAN
;
;         st(0) is infinity.  Return an indefinite value.
;
        fprem                                 ; ST(1) can be anything
return_NAN:
return_empty:
        pop     bp                            ; Restore stack
        ret                                   ; Ok to leave fprem running
;
;         Simulate fptan with st(0) = 0
;
sine_argument_zero:
        fld1                                  ; Simulate tan(0)
        jmp     after_sine_fptan              ; Return the zero value
;
;         The angle was too large.  Remove the modulus and dividend from the
;       stack and return an indefinite result.
;
angle_too_big:
        fcompp                                ; Pop two values from the stack
        fld     indefinite                    ; Return indefinite
        pop     bp                            ; Restore stack
        fwait                                 ; Wait for load to finish
        ret
;
;         Calculate the cosine of the argument.
;       cos(A) = 1/sqrt(1+tan(A)**2)    if tan(A) = Y/X then
;       cos(A) = X/sqrt(X*X + Y*Y)
;
X_numerator:
        fld     st(0)                         ; Copy X value
        fxch    st(2)                         ; Put X in numerator
finish_sine:
        fmul    st,st(0)                      ; Form X*X + Y*Y
        fxch
        fmul    st,st(0)
        fadd                                  ; st(0) = X*X + Y*Y
        fsqrt                                 ; st(0) = sqrt(X*X + Y*Y)
;
;         Form the sign of the result.  The two conditions are the C1 flag from
;       FXAM in bh and the C0 flag from fprem in ah.
;
        and     bh,high(mask cond0)           ; Look at the fprem C0 flag
        and     ah,high(mask cond1)           ; Look at the fxam C1 flag
        or      bh,ah                         ; Even number of flags cancel
        jpe     positive_sine                 ; Two negatives make a positive
        fchs                                  ; Force result negative
positive_sine:
        fdiv                                  ; Form final result
        ret                                   ; Ok to leave fdiv running
cosine  endp
```

```
;
;         This function will calculate the tangent of an angle.
;       The angle, in radians is passed in ST(0), the tangent is returned
;       in ST(0).  The tangent is calculated to an accuracy of 4 units in the
;       least three significant bits of an extended real format number.  The
;       PLM/86 calling format is:
;
;     tangent:  procedure (angle) real external;
;               declare angle real;
;               end tangent;
;
;         Two stack registers are used.  The result of the tangent function is
;       defined for the following cases:
;
;                             angle                             result
;
;               valid or unnormal < 2**62 in magnitude          correct value
;               0                                               0
;               denormal                                        correct denormal
;               valid or unnormal > 2**62 in magnitude          indefinite
;               NAN                                             NAN
;               infinity                                        indefinite
;               empty                                           empty
;
;         The tangent instruction uses the fptan instruction.  Four possible
;       relations are used:
;
;         Let R = |angle MOD PI/4|
;             S = -1 or 1 depending on the sign of the angle
;
;       1) tan(R)       2) tan(PI/4-R)  3) 1/tan(R)     4) 1/tan(PI/4-R)
;
;         The following table is used to decide which relation to use depending
;       on in which octant the angle fell.
;
;       octant          relation
;
;         0              S*1
;         1              S*4
;         2             -S*3
;         3             -S*2
;         4              S*1
;         5              S*4
;         6             -S*3
;         7             -S*2
;
tangent proc

        fxam                                    ; Look at the parameter
        push    bp                              ; Establish stack addressibility
        sub     sp,size local_area              ; Allocate local variable space
        mov     bp,sp
        fstsw   status                          ; Get fxam status
        fld     pi_quarter                      ; Get PI/4
        pop     ax
        lahf                                    ; CF = C0, PF = C2, ZF = C3
        jc      funny_parameter
;
;         Angle is unnormal, normal, zero, or denormal.
;
        fxch                                    ; st(0) = angle, st(1) = PI/4
        jpe     tan_zero_unnormal
;
;         Angle is either an normal or denormal.
;       Reduce the angle to the range -PI/4 < result < PI/4.
;       If fprem cannot perform this operation in one try, the magnitude of the
;       angle must be > 2**62.  Such an angle is so large that any rounding
;       errors could make a very large difference in the reduced angle.
;       It is safest to call this very rare case an error.
;
tan_normal:

        fprem                                   ; Quotient in C0,C3,C1
                                                ; Convert denormals into unnormals
```

```
        mov     sp,bp                          ; Allocate stack spce
        fstsw   status                         ; Quotient identifies octant
                                               ; original angle fell into
        pop     bx                             ; tan(PI*N+x) = tan(x)
        test    bh,high(mask cond2)            ; Test for complete reduction
        jnz     angle_too_big                  ; Exit if angle was too big
;
;         See if the angle must be reversed.
;
;       Assert: -PI/4 < st(0) < PI/4
;
        fabs                                   ; 0 <= st(0) < PI/4
                                               ; C1 in bx has the sign flag
        test    bh,high(mask cond1)            ; must be reversed
        jz      no_tan_reverse
;
;         Angle fell in octants 1,3,5,7.  Reverse it, subtract it from PI/4.
;
        fsub                                   ; Reverse angle
        jmp     short do_tangent
;
;         Angle is either zero or an unnormal.
;
tan_zero_unnormal:

        fstp    st(1)                          ; Remove PI/4
        jz      tan_angle_zero
;
;         Angle is an unnormal.
;
        call    normalize_value
        jmp     tan_normal

tan_angle_zero:

        pop     bp                             ; Restore stack
        ret
;
;         Angle fell in octants 0,2,4,6.  Test for st(0) = 0, fptan won't work.
;
no_tan_reverse:

        ftst                                   ; Test for zero angle
        mov     sp,bp                          ; Allocate stack space
        fstsw   status                         ; C3 = 1 if st(0) = 0
        fstp    st(1)                          ; Remove PI/4
        pop     cx                             ; Get ftst status
        test    ch,high(mask cond3)
        jnz     tan_zero

do_tangent:

        fptan                                  ; tan ST(0) = ST(1)/ST(0)

after_tangent:
;
;         Decide on the order of the operands and their sign for the divide
;       operation while the fptan instruction is working.
;
        pop     bp                             ; Restore stack
        mov     al,bh                          ; Get a copy of fprem C3 flag
        and     ax,mask cond1 + high(mask cond3); Examine fprem C3 flag and
                                               ; fxtract C1 flag
        test    bh,high(mask cond1 + mask cond3); Use reverse divide if in
                                               ; octants 1,2,5,6
        jpo     reverse_divide                 ; Note! parity works on low
                                               ;  8 bits only!
;
;         Angle was in octants 0,3,4,7.
;       Test for the sign of the result.  Two negatives cancel.
;
        or      al,ah
        jpe     positive_divide
```

```
        fchs                            ; Force result negative

positive_divide:

        fdiv                            ; Form result
        ret                             ; Ok to leave fdiv running

tan_zero:

        fldl                            ; Force 1/0 = tan(PI/2)
        jmp     after_tangent
;
;         Angle was in octants 1,2,5,6.
;       Set the correct sign of the result.
;
reverse_divide:

        or      al,ah
        jpe     positive_r_divide

        fchs                            ; Force result negative

positive_r_divide:

        fdivr                           ; Form reciprocal of result
        ret                             ; Ok to leave fdiv running

tangent endp
;
;         This function will normalize the value in st(0).
;       Then PI/4 is placed into st(1).
;
normalize_value:

        fabs                            ; Force value positive
        fxtract                         ; 0 <= st(0) < 1
        fldl                            ; Get normalize bit
        fadd    st(1),st                ; Normalize fraction
        fsub                            ; Restore original value
        fscale                          ; Form original normalized value
        fstp    st(1)                   ; Remove scale factor
        fld     pi_quarter              ; Get PI/4
        fxch
        ret

code    ends
        end

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

intel®

APPLICATION NOTE

AP-186

March 1983

Introduction to the 80186 Microprocessor

Ken Shoemaker
Applications Engineer

ORDER NUMBER: 210973-003

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Distribution
Mail Stop SC6-59
3065 Bowers Avenue
Santa Clara, CA 95051

INTRODUCTION TO THE 80186 MICROPROCESSOR

CONTENTS

1. INTRODUCTION

As state of the art technology has increased the number of transistors possible on a single integrated circuit, these devices have attained new, higher levels of both performance and functionality. Riding this crest are the Intel 80186 and 80286 microprocessors. While the 80286 has added memory protection and management to the basic 8086 architecture, the 80186 has integrated six separate functional blocks into a single device.

The purpose of this note is to explain, through example, the use of the 80186 with various peripheral and memory devices. Because the 80186 integrates a DMA unit, timer unit, interrupt controller unit, bus controller unit and chip select and ready generation unit with the CPU on a single chip (see Figure 1), system construction is simplified since many of the peripheral interfaces are integrated onto the device.

The 80186 family actually consists of two processors: the 80186 and 80188. The only difference between the two processors is that the 80186 maintains a 16-bit external data bus while the 80188 has an 8-bit external data bus. Internally, they both implement the same processor with the same integrated peripheral components. Thus, except where noted, all 80186 information in this note also applies to the 80188. The implications of having an 8-bit external data bus on the 80188 are explicitly noted in appendix I. Any parametric values included in this note are taken from the iAPX 186 Advance Information data sheet, and pertain to 8Mhz devices.

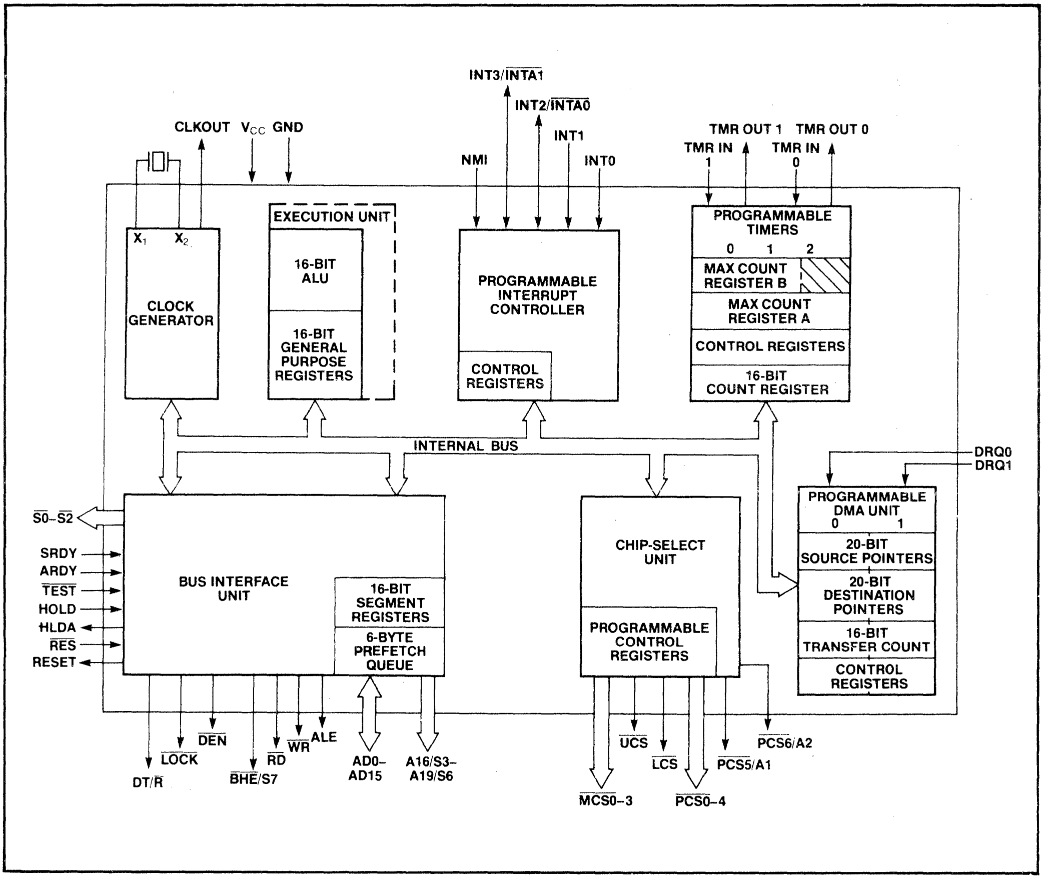


Figure 1. 80186 Block Diagram

2. OVERVIEW OF THE 80186

2.1 The CPU

The 80186 CPU shares a common base architecture with the 8086, 8088 and 80286. It is completely object code compatible with the 8086/88. This architecture features four 16-bit general purpose registers (AX,BX, CX,DX) which may be used as operands in most arithmetic operations in either 8 or 16 bit units. It also features four 16-bit "pointer" registers (SI,DI,BP,SP) which may be used both in arithmetic operations and in accessing memory based variables. Four 16-bit segment registers (CS,DS,SS,ES) are provided allowing simple memory partitioning to aid construction of modular programs. Finally, it has a 16-bit instruction pointer and a 16-bit status register.

Physical memory addresses are generated by the 80186 identically to the 8086. The 16-bit segment value is left shifted 4 bits and then is added to an offset value which is derived from combinations of the pointer registers, the instruction pointer, and immediate values (see Figure 2). Any carry out of this addition is ignored. The result of this addition is a 20-bit physical address which is presented to the system memory.

The 80186 has a 16-bit ALU which performs 8 or 16-bit arithmetic and logical operations. It provides for data movement among registers, memory and I/O space. In addition, the CPU allows for high speed data transfer from one area of memory to another using string move instructions, and to or from an I/O port and memory using block I/O instructions. Finally, the CPU provides a wealth of conditional branch and other control instructions.

In the 80186, as in the 8086, instruction fetching and instruction execution are performed by separate units: the bus interface unit and the execution unit, respectively. The 80186 also has a 6-byte prefetch queue as does the 8086. The 80188 has a 4-byte prefetch queue as does the 8088. As a program is executing, opcodes are fetched from memory by the bus interface unit and placed in this queue. Whenever the execution unit requires another instruction, it takes it out of the queue. Effective processor throughput is increased by adding this queue, since the bus interface unit may continue to fetch instructions while the execution unit executes a long instruction. Then, when the CPU completes this instruction, it does not have to wait for another instruction to be fetched from memory.

2.2 80186 CPU Enhancements

Although the 80186 is completely object code compatible with the 8086, most of the 8086 instructions require fewer clock cycles to execute on the 80186 than on the 8086 because of hardware enhancements in the bus interface unit and the execution unit. In addition, the 80186 provides many new instructions which simplify assembly language programming, enhance the performance of high level language implementations, and reduce object code sizes for the 80186. These new instructions are also included in the 80286. A complete description of the architecture and instruction execution of the 80186 can be found in volume I of the iAPX86/186 users manual. The algorithms for the new instructions are also given in appendix H of this note.

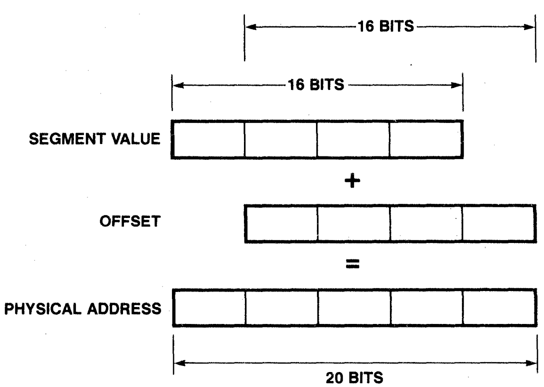


Figure 2. Physical Address Generation in the 80186

2.3 DMA Unit

The 80186 includes a DMA unit which provides two high speed DMA channels. This DMA unit will perform transfers to or from any combination of I/O space and memory space in either byte or word units. Every DMA cycle requires two to four bus cycles, one or two to fetch the data to an internal register, and one or two to deposit the data. This allows word data to be located on odd boundaries, or byte data to be moved from odd locations to even locations. This is normally difficult, since odd data bytes are transferred on the upper 8 data bits of the 16-bit data bus, while even data bytes are transferred on the lower 8 data bits of the data bus.

Each DMA channel maintains independent 20-bit source and destination pointers which are used to access the source and destination of the data transferred. Each of these pointers may independently address either I/O or memory space. After each DMA cycle, the pointers may be independently incremented, decremented, or maintained constant. Each DMA channel also maintains a transfer count which may be used to terminate a series of DMA transfers after a pre-programmed number of transfers.

2.4 Timers

The 80186 includes a timer unit which contains 3 independent 16-bit timer/counters. Two of these timers can be used to count external events, to provide waveforms derived from either the CPU clock or an external clock of any duty cycle, or to interrupt the CPU after a specified number of timer "events." The third timer counts only CPU clocks and can be used to interrupt the CPU after a programmable number of CPU clocks, to give a count pulse to either or both of the other two timers after a programmable number of CPU clocks, or to give a DMA request pulse to the integrated DMA unit after a programmable number of CPU clocks.

2.5 Interrupt Controller

The 80186 includes an interrupt controller. This controller arbitrates interrupt requests between all internal and external sources. It can be directly cascaded as the master to two external 8259A interrupt controllers. In addition, it can be configured as a slave controller to an external interrupt controller to allow complete compatibility with an 80130, 80150, and the iRMX® 86 operating system.

2.6 Clock Generator

The 80186 includes a clock generator and crystal oscillator. The crystal oscillator can be used with a parallel resonant, fundamental mode crystal at 2X the desired CPU clock speed (i.e., 16 MHz for an 8 MHz 80186), or with an external oscillator also at 2X the CPU clock. The output of the oscillator is internally divided by two to provide the 50% duty cycle CPU clock from which all 80186 system timing derives. The CPU clock is externally available, and all timing parameters are referenced to this externally available signal. The clock generator also provides ready synchronization for the processor.

2.7 Chip Select and Ready Generation Unit

The 80186 includes integrated chip select logic which can be used to enable memory or peripheral devices. Six output lines are used for memory addressing and seven output lines are used for peripheral addressing.

The memory chip select lines are split into 3 groups for separately addressing the major memory areas in a typical 8086 system: upper memory for reset ROM, lower memory for interrupt vectors, and mid-range memory for program memory. The size of each of these regions is user programmable. The starting location and ending location of lower memory and upper memory are fixed at 00000H and FFFFFH respectively; the starting location of the mid-range memory is user programmable.

Each of the seven peripheral select lines address one of seven contiguous 128 byte blocks above a programmable base address. This base address can be located in either memory or I/O space in order that peripheral devices may be I/O or memory mapped.

Each of the programmed chip select areas has associated with it a set of programmable ready bits. These ready bits control an integrated wait state generator. This allows a programmable number of wait states (0 to 3) to be automatically inserted whenever an access is made to the area of memory associated with the chip select area. In addition, each set of ready bits includes a bit which determines whether the external ready signals (ARDY and SRDY) will be used, or whether they will be ignored (i.e., the bus cycle will terminate even though a ready has not been returned on the external pins). There are 5 total sets of ready bits which allow independent ready generation for each of upper memory, lower memory, mid-range memory, peripheral devices 0-3 and peripheral devices 4-6.

2.8 Integrated Peripheral Accessing

The integrated peripheral and chip select circuitry is controlled by sets of 16-bit registers accessed using standard input, output, or memory access instructions. These peripheral control registers are all located within a 256 byte block which can be placed in either memory or I/O space. Because they are accessed exactly as if they were external devices, no new instruction types are required to access and control the integrated peripherals. For more information concerning the interfacing and accessing of the integrated 80186 peripherals not included in this note, please consult the 80186 data sheet, or the iAPX 86/186 User's Manual Hardware Reference.

3. USING THE 80186

3.1 Bus Interfacing to the 80186

3.1.1 OVERVIEW

The 80186 bus structure is very similar to the 8086 bus structure. It includes a multiplexed address/data bus, along with various control and status lines (see Table 1). Each bus cycle requires a minimum of 4 CPU clock cycles along with any number of wait states required to accommodate the speed access limitations of external memory or peripheral devices. The bus cycles initiated by the 80186 CPU are identical to the bus cycles initiated by the 80186 integrated DMA unit.

In the following discussion, all timing values given are for an 8 MHz 80186. Future speed selections of the part may have different values for the various parameters.

Each clock cycle of the 80186 bus cycle is called a "T" state, and are numbered sequentially T_1, T_2, T_3, T_w and T_4. Additional idle T states (T_i) can occur between T_4 and T_1 when the processor requires no bus activity (instruction fetches, memory writes, I/O reads, etc.). The ready signals control the number or wait states (T_w) inserted in each bus cycle. This number can vary from 0 to positive infinity.

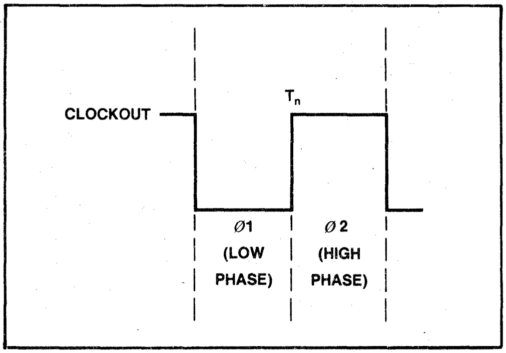


Figure 3. T-state in the 80186

The beginning of a T state is signaled by a high to low transition of the CPU clock. Each T state is divided into two phases, phase 1 (or the low phase) and phase 2 (or the high phase) which occur during the low and high levels of the CPU clock respectively (see Figure 3).

Different types of bus activity occur for all of the T-states (see Figure 4). Address generation information occurs during T_1, data generation during T_2, T_3, T_w and

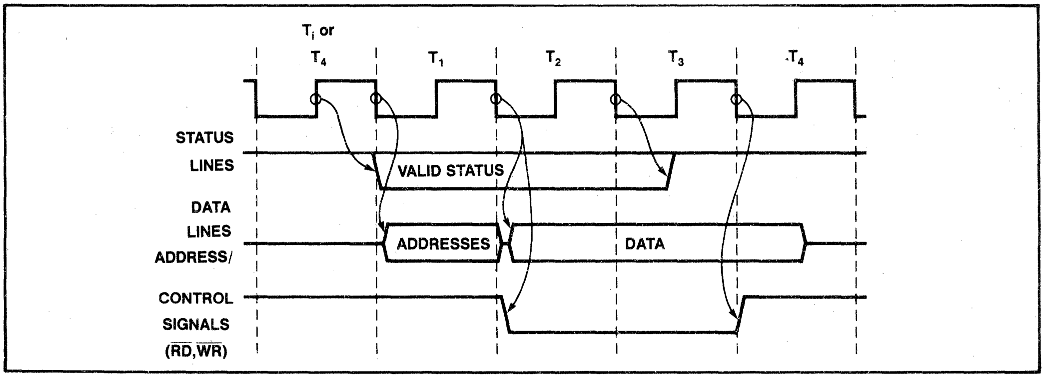


Figure 4. Example Bus Cycle of the 80186

Table 1. 80186 Bus Signals

Function	Signal Name
address/data	AD0-AD15
address/status	A16/S3-A19-S6,$\overline{BHE}$/S7
co-processor control	$\overline{TEST}$
local bus arbitration	HOLD,HLDA
local bus control	ALE,$\overline{RD}$,$\overline{WR}$,DT/$\overline{R}$,$\overline{DEN}$
multi-master bus	$\overline{LOCK}$
ready (wait) interface	SRDY,ARDY
status information	$\overline{S0}$-$\overline{S2}$

T_4. The beginning of a bus cycle is signaled by the status lines of the processor going from a passive state (all high) to an active state in the middle of the T-state immediately before T_1 (either a T_4 or a T_i). Because information concerning an impending bus cycle occurs during the T-state immediately before the first T-state of the cycle itself, two different types of T_4 and T_i can be generated: one where the T state is immediately followed by a bus cycle, and one where the T state is immediatly followed by an idle T state.

During the first type of T_4 or T_i, status information concerning the impending bus cycle is generated for the bus cycle immediately to follow. This information will be available no later than t_{CHSV} (55ns) after the low-to-high transition of the 80186 clock in the middle of the T state. During the second type of T_4 or T_i the status outputs remain inactive (high), since no bus cycle is to be started. This means that the decision per the nature of a T_4 or T_i state (i.e., whether it is immediately followed by a T_i or a T_1) is decided at the beginning of the T-state immediately preceding the T_4 or T_i (see Figure 5). This has consequences for the bus latency time (see section 3.3.2 on bus latency).

3.1.2 PHYSICAL ADDRESS GENERATION

Physical addresses are generated by the 80186 during T_1 of a bus cycle. Since the address and data lines are multiplexed on the same set of pins, addresses must be latched during T_1 if they are required to remain stable for the duration of the bus cycle. To facilitate latching of the physical address, the 80186 generates an active high ALE (Address Latch Enable) signal which can be directly connected to a transparent latch's strobe input.

Figure 6 illustrates the physical address generation parameters of the 80186. Addresses are guaranteed valid no greater then t_{CLAV} (44ns) after the beginning of T_1, and remain valid at least t_{CLAX} (10ns) after the end of T_1. The ALE signal is driven high in the middle of the T state (either T_4 or T_i) immediately preceding T_1 and is driven low in the middle of T_1, no sooner than t_{AVAL} (30 ns) after addresses become valid. This parameter (t_{AVAL}) is required to satisfy the address latch set-up times of address valid until strobe inactive. Addresses remain stable on the address/data bus at least t_{LLAX} (30 ns) after ALE goes inactive to satisfy address latch hold times of strobe inactive to address invalid.

Because ALE goes high long before addresses become valid, the delay through the address latches will be chiefly the propagation delay through the latch rather than the delay from the latch strobe, which is typically longer than the propagation delay. For the Intel 8282 latch, this parameter is t_{IVOV}, the input valid to output valid delay when strobe is held active (high). Note that the 80186 drives ALE high one full clock phase earlier than the 8086 or the 8288 bus controller, and keeps it high throughout the 8086 or 8288 ALE high time (i.e., the 80186 ALE pulse is wider).

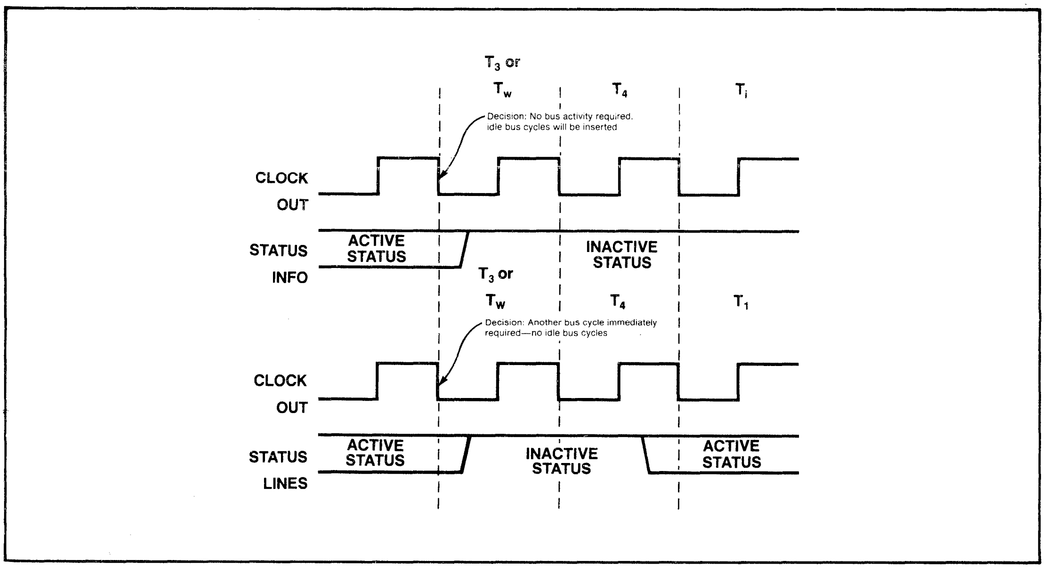
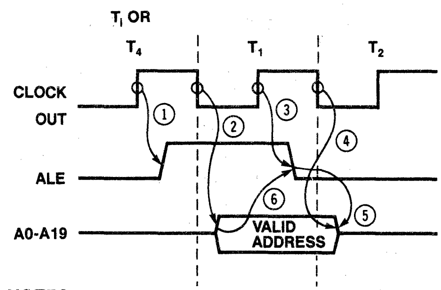


Figure 5. Active-Inactive Status Transitions in the 80186

210973-003



NOTES:
1. t_{CHLH}: Clock high to ALE high=35 ns max
2. t_{CLAV}: Clock low to address valid=44 ns max
3. t_{CHLL}: Clock high to ALE low=35 ns max
4. t_{CLAX}: Clock low to address invalid (address hold from clock low)=10 ns min
5. t_{LLAX}: ALE low to address invalid (address hold from ALE)=30 ns min
6. t_{AVAL}: Address valid to ALE low (address setup to ALE)=30 ns min

Figure 6. Address Generation Timing of the 80186

A typical circuit for latching physical addresses is shown in Figure 7. This circuit uses 3 8282 transparent octal non-inverting latches to demultiplex all 20 address bits provided by the 80186. Typically, the upper 4 address bits are used only to select among various memory components or subsystems, so when the integrated chip selects (see section 8) are used, these upper bits need not be latched. The worst case address generation time from the beginning of T_1 (including address latch propagation time (t_{IVOV}) of the Intel 8282) for the circuit is:

$$t_{CLAV}\ (44ns) + t_{IVOV}\ (30ns) = 74ns$$

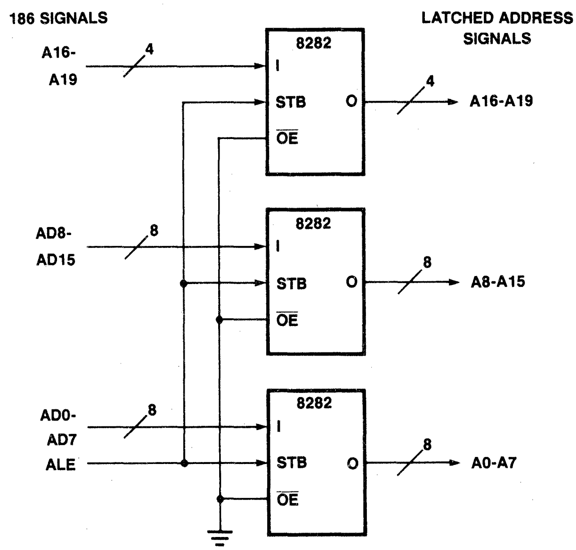


Figure 7. Demultiplexing the Address Bus of the 80186

Many memory or peripheral devices may not require addresses to remain stable throughout a data transfer. Examples of these are the 80130 and 80150 operating system firmware chips, and the 2186 8K x 8 iRAM. If a system is constructed wholly with these types of devices, addresses need not be latched. In addition, two of the peripheral chip select outputs of the 80186 may be configured to provide latched A1 and A2 outputs for peripheral register selects in a system which does not demultiplex the address/data bus.

One more signal is generated by the 80186 to address memory: $\overline{BHE}$ (Bus High Enable). This signal, along with A0, is used to enable byte devices connected to either or both halves (bytes) of the 16-bit data bus (see section 3.1.3 on data bus operation section). Because A0 is used only to enable devices onto the lower half of the data bus, memory chip address inputs are usually driven by address bits A1-A19, NOT A0-A19. This provides 512K unique *word* addresses, or 1M unique BYTE addresses.

Of course, $\overline{BHE}$ is not present on the 8 bit 80188. All data transfers occur on the 8 bits of the data bus.

3.1.3 80186 DATA BUS OPERATION

Throughout T_2, T_3, T_w, and T_4 of a bus cycle the multiplexed address/data bus becomes a 16-bit data bus. Data transfers on this bus may be either in bytes or in words. All memory is byte addressable, that is, the upper and lower byte of a 16-bit word each have a unique byte address by which they may be individually accessed, even though they share a common word address (see Figure 3-6).

All bytes with even addresses (A0 = 0) reside on the lower 8 bits of the data bus, while all bytes with odd addresses (A0 = 1) reside on the upper 8 bits of the data bus. Whenever an access is made to only the even byte, A0 is driven low, $\overline{BHE}$ is driven high, and the data transfer occurs on D0-D7 of the data bus. Whenever an access is made to only the odd byte, $\overline{BHE}$ is driven low, A0 is driven high, and the data transfer occurs on D8-D15 of the data bus. Finally, if a word access is performed to an even address, both A0 and $\overline{BHE}$ are driven low and the data transfer occurs on D0-D15.

Word accesses are made to the addressed byte and to the next higher numbered byte. If a word access is performed to an odd address, two byte accesses must be performed, the first to access the odd byte at the first word address on D8-D15, the second to access the even byte at the next sequential word address on D0-D7. For example, in Figure 8, byte 0 and byte 1 can be individually accessed (read or written) in two separate bus cycles (byte accesses) to byte addresses 0 and 1 at word address 0. They may also be accessed together in a single bus cycle (word access) to word address 0. However, if a word access is made to address 1, two bus cycles will be required, the first to access byte 1 at word address 0 (note byte 0 will not be accessed), and the second to access byte 2 at word address 2 (note byte 3 will not be accessed). This is why all word data should be located at even addresses to maximize processor performance.

When byte reads are made, the data returned on the half of the data bus not being accessed is ignored. When byte writes are made, the data driven on the half of the data bus not being written is indeterminate.

3.1.4 80188 DATA BUS OPERATION

Because the 80188 externally has only an 8 bit data bus, the above discussion about upper and lower bytes of the data bus does not apply to the 80188. No performance improvement will occur if word data is placed on even boundaries in memory space. All word accesses require two bus cycles, the first to access the lower byte of the word; the second to access the upper byte of the word.

Any 80188 access to the integrated peripherals must be done 16 bits at a time: thus in this special case, a word access will occur in a single bus cycle in the 80188. The

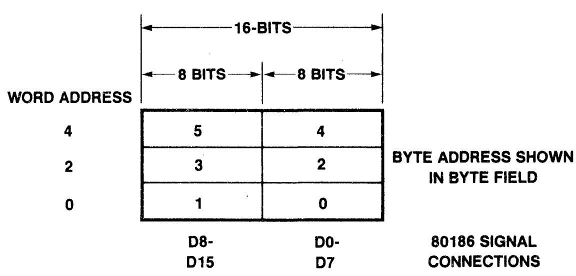


Figure 8. Physical Memory Byte/Word Addressing in the 80186

external data bus will record only a single byte being transferred, however.

3.1.5 GENERAL DATA BUS OPERATION

Because of the bus drive capabilities of the 80186 (200pF, sinking 2mA, sourcing 400uA, roughly twice that of the 8086), this bus may not require additional buffering in many small systems. If data buffers are not used in the system, care should be taken not to allow bus contention between the 80186 and the devices directly connected to the 80186 data bus. Since the 80186 floats the address/data bus before activating any command lines, the only requirement on a directly connected device is that it floats its output drivers after a read *BEFORE* the 80186 begins to drive address information for the next bus cycle. The parameter of interest here is the minimum time from $\overline{RD}$ inactive until addresses active for the next bus cycle (t_{RHAV}) which has a minimum value of 85ns. If the memory or peripheral device cannot disable its output drivers in this time, data buffers will be required to prevent both the 80186 and the peripheral or memory device from driving these lines concurrently. Note, this parameter is unaffected by the addition of wait states. Data buffers solve this problem because their output float times are typically much faster than the 80186 required minimum.

If buffers are required, the 80186 provides a $\overline{DEN}$ (Data ENable) and DT/$\overline{R}$ (Data Transmit/Receive) signals to simplify buffer interfacing. The $\overline{DEN}$ and DT/$\overline{R}$ signals are activated during all bus cycles, whether or not the cycle addresses buffered devices. The $\overline{DEN}$ signal is driven low whenever the processor is either ready to receive data (during a read) or when the processor is ready to send data (during a write) (that is, any time during an active bus cycle when address information is not being generated on the address/data pins). In most systems, the $\overline{DEN}$ signal should NOT be directly connected to the $\overline{OE}$ input of buffers, since unbuffered devices (or other buffers) may be directly connected to the processor's address/data pins. If $\overline{DEN}$ were directly connected to several buffers, contention would occur during read cycles, as many devices attempt to drive the processor bus. Rather, it should be a factor (along with the chip selects for buffered devices) in generating the output enable input of a bi-directional buffer.

The DT/$\overline{R}$ signal determines the direction of data propagation through the bi- directional bus buffers. It is high whenever data is being driven out from the processor, and is low whenever data is being read into the processor. Unlike the $\overline{DEN}$ signal, it may be directly connected to bus buffers, since this signal does not usually directly enable the output drivers of the buffer. An example data bus subsystem supporting both buffered and unbuffered devices is shown in Figure 9. Note that the A side of the 8286 buffer is connected to the 80186, the B side to the external device. The B side of the buffer has greater drive capacity than the A side (since it is meant to drive much greater loads). The DT/$\overline{R}$ signal can directly drive the T (transmit) signal of the buffer, since it has the correct polarity for this configuration.

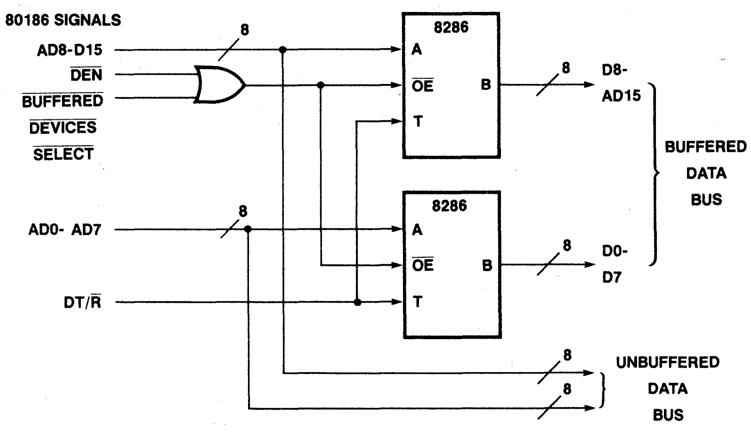
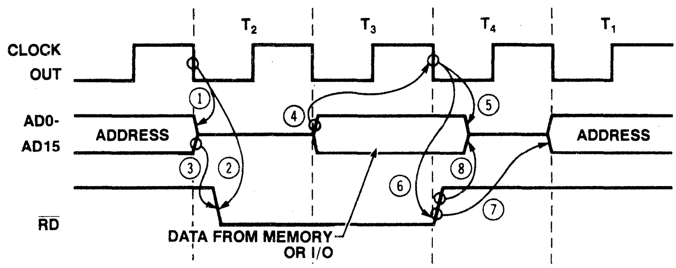


Figure 9. Example 80186 Buffered/Unbuffered Data Bus

210973-003



1. t_{CLAZ}: Clock low until address float=35 ns max
2. t_{CLRL}: Clock low until $\overline{\text{RD}}$ active = 70 ns max
3. t_{AZRL}: Address float until $\overline{\text{RD}}$ active = 0 ns min
4. t_{DVCL}: Data valid until clock low (data input set-up time) = 20 ns min*
5. t_{CLDX}: Clock low until data invalid (data input hold time from clock) = 10 ns min*
6. t_{CLRH}: Clock low until $\overline{\text{RD}}$ high = 10 ns min
7. t_{RHAV}: $\overline{\text{RD}}$ high until addresses valid = 85 ns min
8. t_{RHDX}: Read high until data invalid (data input hold from $\overline{\text{RD}}$) = 0 ns min*

* Input requirements of 80186, all others are output characteristics

Figure 10. Read Cycle Timing of the 80186

3.1.6 CONTROL SIGNALS

The 80186 directly provides the control signals $\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{LOCK}}$ and $\overline{\text{TEST}}$. In addition, the 80186 provides the status signals $\overline{\text{S0}}$-$\overline{\text{S2}}$ and S6 from which all other required bus control signals can be generated.

3.1.6.1 $\overline{\text{RD}}$ and $\overline{\text{WR}}$

The $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals strobe data to or from memory or I/O space. The $\overline{\text{RD}}$ signal is driven low off the beginning of T_2, and is driven high off the beginning of T_4 during all memory and I/O reads (see Figure 10). $\overline{\text{RD}}$ will not become active until the 80186 has ceased driving address information on the address/data bus. Data is sampled into the processor at the beginning of T_4. $\overline{\text{RD}}$ will not go inactive until the processor's data hold time (10ns) has been satisfied.

Note that the 80186 does not provide separate I/O and memory $\overline{\text{RD}}$ signals. If separate I/O read and memory read signals are required, they can be synthesized using the $\overline{\text{S2}}$ signal (which is low for all I/O operations and high for all memory operations) and the $\overline{\text{RD}}$ signal (see Figure 11). It should be noted that if this approach is used, the $\overline{\text{S2}}$ signal will require latching, since the $\overline{\text{S2}}$ signal (like $\overline{\text{S0}}$ and $\overline{\text{S1}}$) goes to a passive state well before the beginning of T_4 (where $\overline{\text{RD}}$ goes inactive). If $\overline{\text{S2}}$ was directly used for this purpose, the type of read command (I/O or memory) could change just before T_4 as $\overline{\text{S2}}$ goes to the passive state (high). The status signals may be latched using ALE in an identical fashion as is used to latch the address signals (often using the spare bits in the address latches).

Often the lack of a seperate I/O and memory $\overline{\text{RD}}$ signal

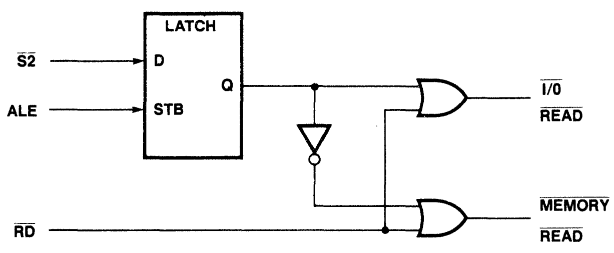
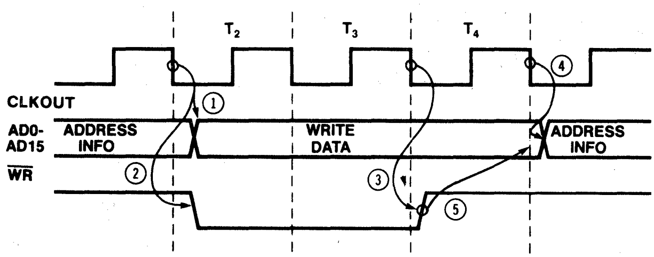


Figure 11. Generating I/O and Memory Read Signals from the 80186

is not important in an 80186 system. Each of the 80186 chip select signals will respond on only one of memory or I/O accesses (the memory chip selects respond only to accesses memory space; the peripheral chip selects can respond to accesses in either I/O or memory space, at programmer option). Thus, the chip select signal enables the external device only during accesses to the proper address in the proper space.

The $\overline{WR}$ signal is also driven low off the beginning of T_2 and driven high off the beginning of T_4. Like the $\overline{RD}$ signal, the $\overline{WR}$ signal is active for all memory and I/O writes, and also like the $\overline{RD}$ signal, separate I/O and memory writes may be generated using the latched $\overline{S2}$ signal along with the $\overline{WR}$ signal (see Figure 12). More importantly, however, is the active going edge of write. At the time $\overline{WR}$ makes its active (high to low) transition, valid write data is NOT present on the data bus. This has consequences when using this signal as a write enable signal for DRAMs and iRAMs since both of these devices require that the write data be stable on the data bus at the time of the inactive to active transition of the $\overline{WE}$ signal. In DRAM applications, this problem is solved by a DRAM controller (such as the Intel 8207 or 8203), while with iRAMs this problem may be solved by placing cross-coupled NAND gates between the CPU and the iRAMs on the $\overline{WR}$ line (see Figure 13). This will delay the active going edge of the $\overline{WR}$ signal to the iRAMs by a clock phase, allowing valid data to be driven onto the data bus.



1. t_{CLDV}: Clock low until data valid = 44 ns max
2. t_{CVCTV}: Clock low until $\overline{WR}$ active = 70 ns max
3. t_{CVCTX}: Clock low until $\overline{WR}$ inactive = 55 ns max
4. t_{CLDOX}: Clock high until data invalid = 10 ns max
5. $\overline{WR}$ inactive until data invalid = t_{CLCL} min — t_{CVCTX} + t_{CLDOX}
= 125 — 55 + 10
= 80 ns

Figure 12. Write Cycle Timing of the 80186

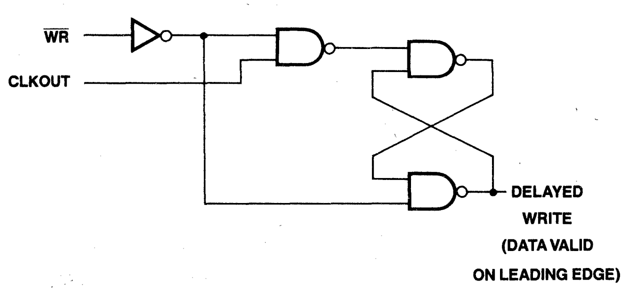


Figure 13. Synthesizing Delayed Write from the 80186

3.1.6.2 Queue Status Signals

If the $\overline{RD}$ line is externally grounded during reset and remains grounded during processor operation, the 80186 will enter "queue status" mode. When in this mode, the $\overline{WR}$ and ALE signals become queue status outputs, reflecting the status of the internal prefetch queue during each clock cycle. These signals are provided to allow a processor extension (such as the Intel 8087 floating point processor) to track execution of instructions within the 80186. The interpretation of QS0 (ALE) and QS1 ($\overline{WR}$) are given in Table 2. These signals change on the high-to-low clock transition, one clock phase earlier than on the 8086. Note that since execution unit operation is independent of bus interface unit operation, queue status lines may change in any T state.

Table 2. 80186 Queue Status

QS1	QS0	Interpretation
0	0	no operation
0	1	first byte of instruction taken from queue
1	0	queue was reinitialized
1	1	subsequent byte of instruction taken from queue

Since the ALE, $\overline{RD}$, and $\overline{WR}$ signals are not directly available from the 80186 when it is configured in queue status mode, these signals must be derived from the status lines $\overline{S0}$-$\overline{S2}$ using an external 8288 bus controller (see below). To prevent the 80186 from accidentally entering queue status mode during reset, the $\overline{RD}$ line is internally provided with a weak pullup device. $\overline{RD}$ is the ONLY three-state or input pin on the 80186 which is supplied with a pullup or pulldown device.

3.1.6.3 Status Lines

The 80186 provides 3 status outputs which are used to indicate the type of bus cycle currently being executed. These signals go from an inactive state (all high) to one of seven possible active states during the T state immediately preceding T_1 of a bus cycle (see Figure 5). The possible status line encodings and their interpretations are given in Table 3. The status lines are driven to their inactive state in the T state (T_3 or T_w) immediately preceding T_4 of the current bus cycle.

The status lines may be directly connected to an 8288 bus controller, which can be used to provide local bus control signals or multi-bus control signals (see Figure 14). Use of the 8288 bus controller does not preclude the use of the 80186 generated $\overline{RD}$, $\overline{WR}$ and ALE signals, however. The 80186 directly generated signals may be used to provide local bus control signals, while an 8288 is used to provide multi-bus control signals, for example.

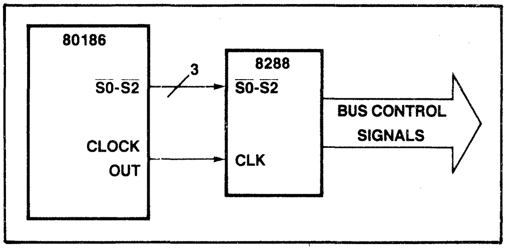


Figure 14. 80186/8288 Bus Controller Interconnection

Table 3. 80186 Status Line Interpretation

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Operation
0	0	0	interrupt acknowledge
0	0	1	read I/O
0	1	0	write I/O
0	1	1	halt
1	0	0	instruction fetch
1	0	1	read memory
1	1	0	write memory
1	1	1	passive

The 80186 provides two additional status signals: S6 and S7. S7 is equivalent to $\overline{BHE}$ (see section 3.1.2) and appears on the same pin as $\overline{BHE}$. $\overline{BHE}$/S7 changes state, reflecting the bus cycle about to be run, in the middle of the T state (T_4 or T_i) immediately preceding T_1 of the bus cycle. This means that $\overline{BHE}$/S7 does not need to be latched, i.e., it may be used directly as the $\overline{BHE}$ signal. S6 provides information concerning the unit generating the bus cycle. It is time multiplexed with A19, and is available during T_2, T_3, T_4 and T_w. In the 8086 family, all central processors (e.g., the 8086, 8088 and 8087) drive this line low, while all I/O processors (e.g., 8089) drive this line high during their respective bus cycles. Following this scheme, the 80186 drives this line low whenever the bus cycle is generated by the 80186 CPU, but drives it high when the bus cycle is generated by the integrated 80186 DMA unit. This allows external devices to distinguish between bus cycles fetching data for the CPU from those transfering data for the DMA unit.

Three other status signals are available on the 8086 but not on the 80186. They are S3, S4, and S5. Taken together, S3 and S4 indicate the segment register from which the current physical address derives. S5 indicates the state of the interrupt flip-flop. On the 80186, these signals will *ALWAYS* be low.

3.1.6.4 $\overline{TEST}$ and $\overline{LOCK}$

Finally, the 80186 provides a $\overline{TEST}$ input and a $\overline{LOCK}$ output. The $\overline{TEST}$ input is used in conjunction with the processor WAIT instruction. It is typically driven by a processor extension (like the 8087) to indicate whether it is busy. Then, by executing the WAIT (or FWAIT) instruction, the central processor may be forced to temporarily suspend program execution until the processor extension indicates that it is idle by driving the $\overline{\text{TEST}}$ line low.

The $\overline{\text{LOCK}}$ output is driven low whenever the data cycles of a LOCKED instruction are executed. A LOCKED instruction is generated whenever the LOCK prefix occurs immediately before an instruction. The LOCK prefix is active for the single instruction immediately following the LOCK prefix. This signal is used to indicate to a bus arbiter (e.g., the 8289) that a series of locked data transfers is occurring. The bus arbiter should under no circumstances release the bus while locked transfers are occurring. The 80186 will not recognize a bus HOLD, nor will it allow DMA cycles to be run by the integrated DMA controller during locked data transfers. LOCKED transfers are used in multiprocessor systems to access memory based semaphore variables which control access to shared system resources (see AP-106, "Multiprogramming with the iAPX88 and iAPX86 Microsystems," by George Alexy (Sept. 1980)).

On the 80186, the $\overline{\text{LOCK}}$ signal will go active during T_1 of the first DATA cycle of the locked transfer. It is driven inactive 3 T-states after the beginning of the last DATA cycle of the locked transfer. On the 8086, the $\overline{\text{LOCK}}$ signal is activated immediately after the LOCK prefix is executed. The LOCK prefix may be executed well before the processor is prepared to perform the locked data transfer. This has the unfortunate consequence of activating the $\overline{\text{LOCK}}$ signal before the first LOCKED data cycle is performed. Since $\overline{\text{LOCK}}$ is active before the processor requires the bus for the data transfer, opcode pre-fetching can be LOCKED. However, since the 80186 does not activate the $\overline{\text{LOCK}}$ signal until the processor is ready to actually perform the locked transfer, locked pre-fetching will not occur with the 80186.

Note that the $\overline{\text{LOCK}}$ signal does not remain active until the end of the last data cycle of the locked transfer. This may cause problems in some systems if, for example, the processor requests memory access from a dual ported RAM array and is denied immediate access (because of a DRAM refresh cycle, for example). When the processor finally is able to gain access to the RAM array, it may have already dropped its $\overline{\text{LOCK}}$ signal, thus allowing the dual port controller to give the other port access to the RAM array instead. An example circuit which can be used to hold $\overline{\text{LOCK}}$ active until a RDY has been received by the 80186 is shown in Figure 15.

3.1.7 HALT TIMING

A HALT bus cycle is used to signal the world that the 80186 CPU has executed a HLT instruction. It differs from a normal bus cycle in two important ways.

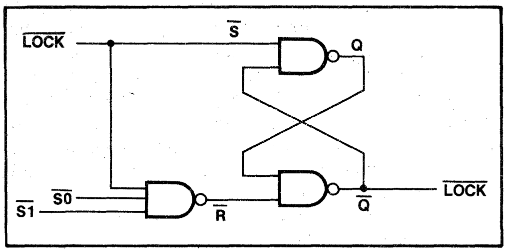


Figure 15. Circuit Holding $\overline{\text{Lock}}$ Active Until Ready is Returned

The first way in which a HALT bus cycle differs from a normal bus cycle is that since the processor is entering a halted state, none of the control lines ($\overline{\text{RD}}$ or $\overline{\text{WR}}$) will be driven active. Address and data information will not be driven by the processor, and no data will be returned. The second way a HALT bus cycle differs from a normal bus cycle is that the $\overline{\text{S0}}$-$\overline{\text{S2}}$ status lines go to their passive state (all high) during T_2 of the bus cycle, well before they go to their passive state during a normal bus cycle.

Like a normal bus cycle, however, ALE is driven active. Since no valid address information is present, the information strobed into the address latches should be ignored. This ALE pulse can be used, however, to latch the HALT status from the $\overline{\text{S0}}$-$\overline{\text{S2}}$ status lines.

The processor being halted does not interfere with the operation of any of the 80186 integrated peripheral units. This means that if a DMA transfer is pending while the processor is halted, the bus cycles associated with the DMA transfer will run. In fact, DMA latency time will improve while the processor is halted because the DMA unit will not be contending with the processor for access to the 80186 bus (see section 4.4.1).

3.1.8 8288 AND 8289 INTERFACING

The 8288 and 8289 are the bus controller and multimaster bus arbitration devices used with the 8086 and 8088. Because the 80186 bus is similar to the 8086 bus, they can be directly used with the 80186. Figure 16 shows an 80186 interconnection to these two devices.

The 8288 bus controller generates control signals ($\overline{\text{RD}}$, $\overline{\text{WR}}$, ALE, DT/$\overline{\text{R}}$, $\overline{\text{DEN}}$, etc.) for an 8086 maximum mode system. It derives its information by decoding status lines $\overline{\text{S0}}$-$\overline{\text{S2}}$ of the processor. Because the 80186 and the 8086 drive the same status information on these lines, the 80186 can be directly connected to the 8288 just as in an 8086 system. Using the 8288 with the 80186 does not prevent using the 80186 control signals directly. Many systems require both local bus control signals and system bus control signals. In this type of system, the 80186 lines could be used as the local signals, with the

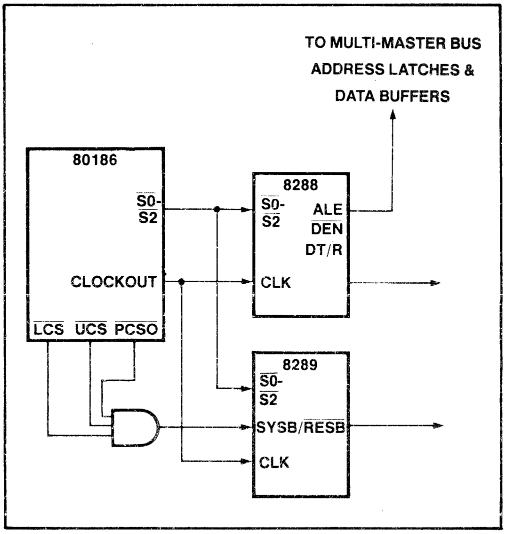


Figure 16. 80186/8288/8289 Interconnection

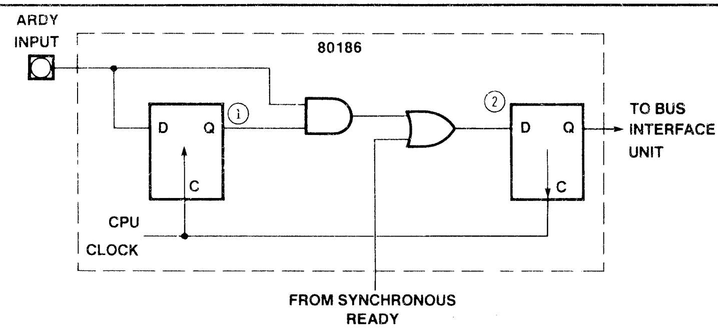
8288 lines used as the system signals. Note that in an 80186 system, the 8288 generated ALE pulse occurs later than that of the 80186 itself. In many multimaster bus systems, the 8288 ALE pulse should be used to strobe the addresses into the system bus address latches to insure that the address hold times are met.

The 8289 bus arbiter arbitrates the use of a multi-master system bus among various devices each of which can become the bus master. This component also decodes status lines S0-S2 of the processor directly to determine when the system bus is required. When the system bus is required, the 8289 forces the processor to wait until it has acquired control of the bus, then it allows the processor to drive address, data and control information onto the system bus. The system determines when it requires system bus resources by an address decode. Whenever the address being driven coincides with the address of an on-board resource, the system bus is not required and thus will not be requested. The circuit shown factors the 80186 chip select lines to determine when the system bus should be requested, or when the 80186 request can be satisfied using a local resource.

3.1.9 READY INTERFACING

The 80186 provides two ready lines, a synchronous ready (SRDY) line and an asynchronous ready (ARDY) line. These lines signal the processor to insert wait states (T_w) into a CPU bus cycle. This allows slower devices to respond to CPU service requests (reads or writes). Wait states will only be inserted when both ARDY and SRDY are low, i.e., only one of ARDY or SRDY need be active to terminate a bus cycle. Any number of wait states may be inserted into a bus cycle. The 80186 will ignore the RDY inputs during any accesses to the integrated peripheral registers, and to any area where the chip select ready bits indicate that the external ready should be ignored.

The timing required by the two RDY lines is different. The ARDY line is meant to be used with asynchronous ready inputs. Thus, inputs to this line will be internally synchronized to the CPU clock before being presented to the processor. The synchronization circuitry used with the ARDY line is shown in Figure 17. Figure 18A and 18B show valid and invalid transitions of the ARDY line (and subsequent wait state insertion). The first flip-flop is used to "resolve" the asynchronous transition of the ARDY line. It will achieve a definite level (either high or low) before its output is latched into the second flip-



1. Asynchronous Resolution Flip Flop
2. Ready Latch Flip Flop

NOTE: The second flip-flop is not actually in the circuit. It is drawn here only to show the functional equivalent of the interface to the BIU.

Figure 17. Asynchronous Ready Circuitry of the 80186

flop for presentation to the CPU. When latched high, it allows the level present on the ARDY line to pass directly to the CPU; when latched low, it forces not ready to be presented to the CPU (see Appendix B for 80186 synchronizer information).

With this scheme, notice that only the active going edge of the ARDY signal is synchronized. Once the synchronization flip-flop has sampled high, the ARDY input directly drives the RDY flip-flop. Since inputs to this RDY flip-flop must satisfy certain setup and hold times, it is important that these setup and hold times (t_{ARYLCL} = 35ns and t_{CHARYX} = 15 ns respectively) be satisfied by any inactive going transition of the ARDY line. The reason ARDY is implemented in this manner is to allow a slow device the greatest amount of time to respond with a not ready after it has been selected. In a normally ready system, a slow device must respond with a not ready quickly after it has been selected to prevent the processor from continuing and accessing invalid data from the slow device. By implementing ARDY in the above fashion, the slow device has an additional clock phase to respond with a not ready.

If RDY is sampled active into the RDY flip-flop at the beginning of T_3 or T_w (meaning that ARDY was sam-

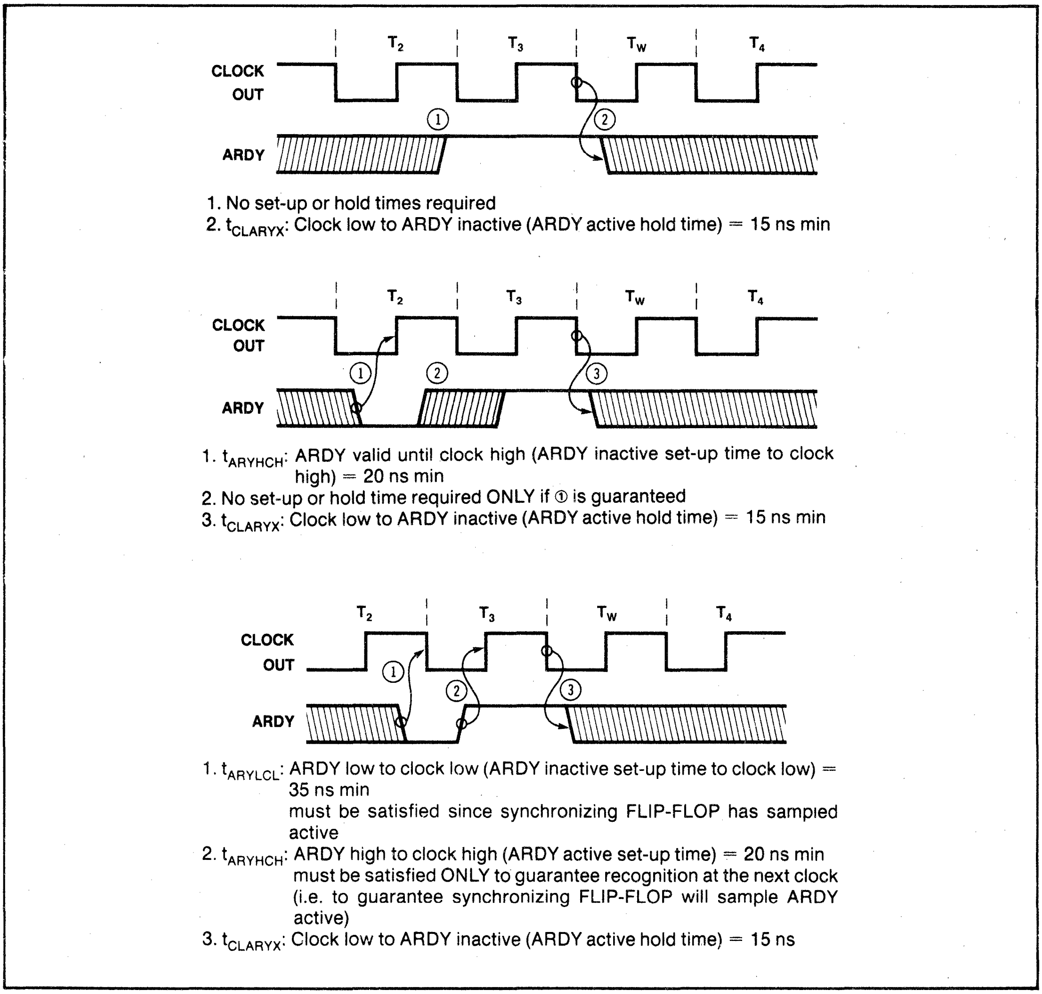
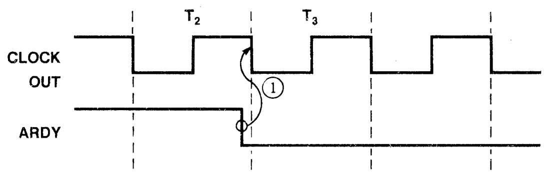
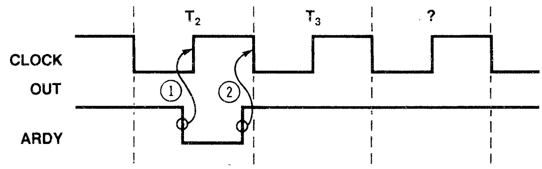


Figure 18A. Valid ARDY Transitions

210973-003



(1) LESS THAN 35 ns

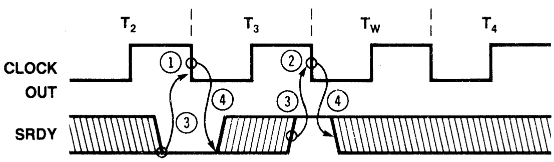


1. Less than 20 ns
2. Less than 35 ns

Figure 18B. Invalid ARDY Transitions

pled high into the synchronization flip-flop in the middle of a T state, and has remained high until the beginning of the next T state), that T state will be immediately followed by T_4. If RDY is sampled low into the RDY flip-flop at the beginning of T_3 or T_w (meaning that either ARDY was sampled low into the synchronization flip-flop OR that ARDY was sampled high into the synchronization flip-flop, but has subsequently changed to low before the ARDY setup time) that T state will be immediately followed by a wait state (T_w). Any asynchronous transition on the ARDY line not occurring during the above times, that is, when the processor is not "looking at" the ready lines, will not cause CPU malfunction. Again, for ARDY to force wait states to be inserted, SRDY must be driven low, since they are internally ORed together to form the processor RDY signal.

The synchronous ready (SRDY) line requires that ALL transitions on this line during T_2, T_3 or T_w satisfy a certain setup and hold time (t_{SRYCL} = 35 ns and t_{CLSRY} = 15 ns respectively). If these requirements are not met, the CPU will not function properly. Valid transitions on this line, and subsequent wait state insertion is shown in Figure 19. The processor looks at this line at the beginning of each T_3 and T_w. If the line is sampled active at the beginning of either of these two cycles, that cycle will



1. Decision: Not ready, T-state will be followed by a wait state
2. Decision: Ready, T-state will not be followed by a wait state
3. t_{SRYCL}: Synchronous ready stable until clock low (SRDY set-up time) = 35 ns min
4. t_{CLSRY}:

Clock low until synchronous ready transition (SRDY hold time) = 15 ns min

Figure 19. Valid SRDY transitions on the 80186

be immediately followed by T_4. On the other hand, if the line is sampled inactive at the beginning of either of these two cycles, that cycle will be followed by a T_w. Any asynchronous transition on the SRDY line not occurring at the beginning of T_3 or T_w, that is, when the processor is not "looking at" the ready lines will not cause CPU malfunction.

3.1.10 BUS PERFORMANCE ISSUES

Bus cycles occur sequentially, but do not necessarily come immediately one after another, that is the bus may remain idle for several T states (T_i) between each bus access initiated by the 80186. This occurs whenever the 80186 internal queue is full and no read/write cycles are being requested by the execution unit or integrated DMA unit. The reader should recall that a separate unit, the bus interface unit, fetches opcodes (including immediate data) from memory, while the execution unit actually executes the pre-fetched instructions. The number of clock cycles required to execute an 80186 instruction vary from 2 clock cycles for a register to register move to 67 clock cycles for an integer divide.

If a program contains many long instructions, program execution will be CPU limited, that is, the instruction queue will be constantly filled. Thus, the execution unit does not need to wait for an instruction to be fetched. If a program contains mainly short instructions or data move instructions, the execution will be bus limited. Here, the execution unit will be required to wait often for an instruction to be fetched before it continues its operation. Programs illustrating this effect and performance degradation of each with the addition of wait states are given in appendix G.

All instruction fetches are word (16-bit) fetches from even addresses unless the fetch occurs as a result of a jump to an odd location. This maximizes the utilization of each bus cycle used for instruction fetching, since each fetch will access two bytes of information. It is also good programming practice to locate all word data at even locations, so that both bytes of the word may be accessed in a single bus cycle (see discussion on data bus interfacing for further information, section 3.1.3 of this note).

Although the amount of bus utilization, i.e., the percentage of bus time used by the 80186 for instruction fetching and execution required for top performance will vary considerably from one program to another, a typical instruction mix on the 80186 will require greater bus utilization than the 8086. This is caused by the higher performance execution unit requiring instructions from the prefetch queue at a greater rate. This also means that the effect of wait states is more pronounced in an 80186 system than in an 8086 system. In all but a few cases, however, the performance degradation incurred by adding a wait state is less than might be expected because instruction fetching and execution are performed by separate units.

3.2 Example Memory Systems

3.2.1 2764 INTERFACE

With the above knowledge of the 80186 bus, various memory interfaces may be generated. One of the simplest of these is the example EPROM interface shown in Figure 20.

The addresses are latched using the address generation circuit shown earlier. Note that the A0 line of each EPROM is connected to the A1 address line from the 80186, NOT the A0 line. Remember, A0 only signals a data transfer on the lower 8 bits of the 16-bit data bus! The EPROM outputs are connected directly to the address/data inputs of the 80186, and the 80186 $\overline{RD}$ signal is used as the $\overline{OE}$ for the EPROMs.

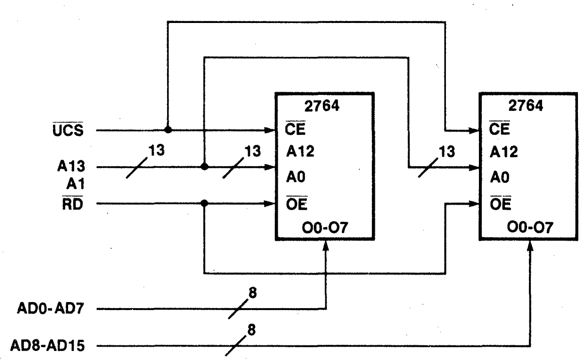


Figure 20. Example 2764/80186 Interface

The chip enable of the EPROM is driven directly by the chip select output of the 80186 (see section 8). In this configuration, the access time calculation for the EPROMs are:

time from
address: $(3 + N)*t_{CLCL} - t_{CLAV} - t_{IVOV}(8282) - t_{DVCL}$
$= 375 + (N * 125) - 44 - 30 - 20$
$= 281 + (N * 125)$ ns

time from
chip select: $(3 + N)*t_{CLCL} - t_{CLCSV} - t_{DVCL}$
$= 375 + (N * 125) - 66 - 20$
$= 289 + (N * 125)$ ns

time from
$\overline{RD}$ ($\overline{OE}$): $(2 + N)t_{CLCL} - t_{CLRL} - t_{DVCL}$
$= 250 + (N * 125) - 70 - 20$
$= 160 + (N * 125)$ ns

where:

t_{CLAV} = time from clock low in T_1 until addresses are valid

t_{CLCL} = clock period of processor

t_{IVOV} = time from input valid of 8282 until output valid of 8282

t_{DVCL} = 186 data valid input setup time until clock low time of T_4

t_{CLCSV} = time from clock low in T_1 until chip selects are valid

t_{CLRL} = time from clock low in T_2 until $\overline{RD}$ goes low

N = number of wait states inserted

Thus, for 0 wait state operation, 250ns EPROMs must be used. The only significant parameter not included above is t_{RHAV}, the time from $\overline{RD}$ inactive (high) until the 80186 begins driving address information. This parameter is 85ns, which meets the 2764-25 (250ns speed selection) output float time of 85ns. If slower EPROMs are used, a discrete data buffer *MUST* be inserted between the EPROM data lines and the address/data bus, since these devices may continue to drive data information on the multiplexed address/data bus when the 80186 begins to drive address information for the next bus cycle.

3.2.2 2186 INTERFACE

An example interface between the 80186 and 2186 iRAMs is shown in Figure 21. This memory component is almost an ideal match with the 80186, because of its large integration, and its not requiring address latching.

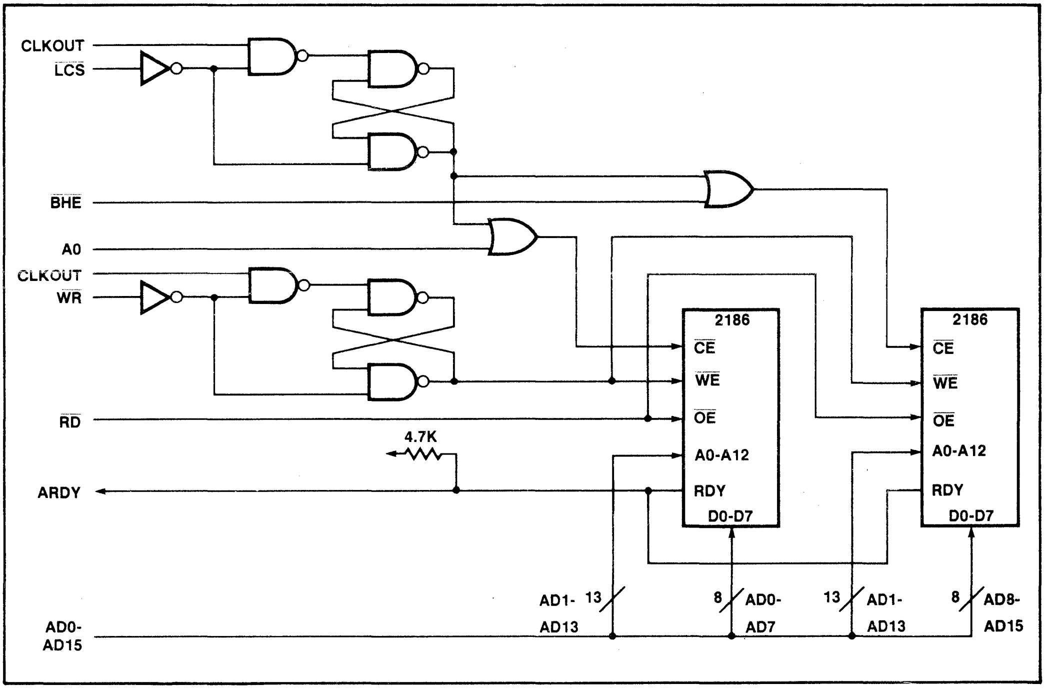


Figure 21. Example 2186/80186 Interface

The 2186 internally is a dynamic RAM integrated with refresh and control circuitry. It operates in two modes, pulse mode and late cycle mode. Pulse mode is entered if the $\overline{CE}$ signal is low to the device a maximum of 130ns, and requires the command input ($\overline{RD}$ or $\overline{WE}$) to go active within 90ns after $\overline{CE}$. Because of these requirements, interfacing the 80186 to the 2186 in pulse mode would be difficult. Instead, the late cycle mode is used. This affords a much simpler interface with no loss of performance. The iRAM automatically selects between these modes by the nature of the control signals.

The 2186 is a leading edge triggered device. This means that address and data information are strobed into the device on the active going (high to low) transition of the command signal. This requires both $\overline{CE}$ and $\overline{WR}$ be delayed until the address and data driven by the 80186 are guaranteed stable. Figure 21 shows a simple circuit which can be used to perform this function. Note that ALE *CANNOT* be used to delay $\overline{CE}$ if addresses are not latched externally, because this would violate the address hold time required by the 2186 (30ns).

Because the 2186s are RAMs, data bus enables ($\overline{BHE}$ and A0, see previous section) MUST be used to factor either the chip enables or write enables of the lower and upper bytes of the 16-bit RAM memory system. If this is not done, all memory writes, including single byte writes, will write to both the upper and lower bytes of the memory system. The example system shown uses $\overline{BHE}$ and A0 as factors to the 2186 $\overline{CE}$. This may be done, because both of these signals (A0 and $\overline{BHE}$) are valid when the address information is valid from the 80186.

The 2186 requires a certain amount of recovery time between its chip enable going inactive and its chip enable going active insure proper operation. For a "normal" cycle (a read or write), this time is t_{EHEL} = 40ns. This means that the 80186 chip select lines will go inactive soon enough at the end of a bus cycle to provide the required recovery time even if two consecutive accesses are made to the iRAMs. If the 2186 $\overline{CE}$ is asserted without a command signal ($\overline{WE}$ or $\overline{OE}$), a "False Memory Cycle" (FMC) will be generated. Whenever a FMC is generated, the recovery time is much longer; another memory cycle must not be initiated for 200ns. As a result, if the memory system will generate FMCs, $\overline{CE}$ must be taken away in the middle of the T state (T_3 or T_w) immediately preceding T_4 to insure two consecutive cycles to the iRAM will not violate this parameter. Status going passive (all high) can be used for this purpose. These lines will all go high during the first phase of the next to last T state (either T_3 or T_w) of a bus cycle (see section 3.1.5).

Finally, since it is a dynamic device, the 2186 requires refresh cycles to maintain data integrity. The circuitry to generate these refresh cycles is integrated within the 2186. Because of this, the 2186 has a ready line which is used to suspend processor operation if a processor RAM access coincides with an internally generated refresh cycle. This is an open collector output, allowing many of them to be wire-OR'ed together, since more than one device may be accessed at at time. These lines are also normally ready, which means that they will be high whenever the 2186 is not being accessed, i.e., they will only be driven low if a processor request coincides with an internal refresh cycle. Thus, the ready lines from the iRAM must be factored into the 80186 RDY circuit only during accesses to the iRAM itself. Since the 2186 refresh logic operates asynchronously to the 80186, this RDY line must be synchronized for proper operation with the 80186, either by the integrated ready synchronizer or by an external circuit. The example circuit uses the integrated synchronizer associated with the ARDY processor input.

The ready lines of the 2186 are active unless a processor access coincides with an internal refresh cycle. These lines must go inactive soon enough after a cycle is requested to insert wait states into the data cycle. The 2186 will drive this line low within 50ns after $\overline{CE}$ is received, which is early enough to force the 80186 to insert wait states if they are required. The primary concern here is that the ARDY line be driven not active before its setup time in the middle of T_2. This is required by the nature of the asynchronous ready synchronization circuitry of the 80186. Since the RDY pulse of the 2186 may be as narrow as 50ns, if ready was returned after the first stage of the synchronizer, and subsequently changed state within the ready setup and hold time of the high to low going edge of the CPU clock at the end of T_2, improper operation may occur (see section 3.1.6).

The example interface shown has a zero wait state RAM read access time from $\overline{CE}$ of:

$$3 * t_{CLCL} - t_{CLCSV} - (\text{TTL delay}) - t_{DVCL}$$
$$= 375 - 66 - 30 - 20 \text{ ns}$$
$$= 259 \text{ ns}$$

where:

t_{CLCL} = CPU clock cycle time

t_{CLCSV} = time from clock low in T_1 until chip selects are valid

t_{DVCL} = 80186 data in setup time before clock low in T_4

The data valid delay from $\overline{OE}$ active is less than 100ns, and is therefore not an access time limiter in this interface. Additionally, the 2186 data float time from $\overline{RD}$ inactive is less than the 85ns 80186 imposed maximum. The $\overline{CE}$ generation circuit shown in Figure 21 provides an address setup time of at least 11ns, and an address hold time of at least 35ns (assuming a maximum two level TTL delay of less than 30ns).

Write cycle address setup and hold times are identical to the read cycle times. The circuit shown provides at least 11ns write data setup and 100ns data hold time from WE, easily meeting the 0ns setup and 40ns hold times required by the 2186.

For more information concerning 2186 timing and interfacing, please consult the 2186 data sheet, or the application note AP-132, "Designing Memory Systems with the 8Kx8 iRAM" by John Fallin and William Righter (June 1982).

3.2.3 8203 DRAM INTERFACE

An example 8203/DRAM interface is shown in Figure 22. The 8203 provides all required DRAM control signals, address multiplexing, and refresh generation. In this circuit, the 8203 is configured to interface with 64K DRAMs.

All 8203 cycles are generated off control signals (RD and WR) provided by the 80186. These signals will not go active until T_2 of the bus cycle. In addition, since the 8203 clock (generated by the internal crystal oscillator of the 8203) is asynchronous to the 80186 clock, all memory requests by the 80186 must be synchronized to the 8203 before the cycle will be run. To minimize this synchronization time, the 8203 should be used with the highest speed crystal that will maintain DRAM compatibility. Even if a 25 MHz crystal is used (the maximum allowed by the 8203) two wait states will be required by the example circuit when using 150ns DRAMs with an 8 MHz 80186, three wait states if 200ns DRAMs are used (see timing analysis, Figure 23).

The entire RAM array controlled by the 8203 can be selected by one or a group of the 80186 provided chip selects. These chip selects can also be used to insert the wait states required by the interface.

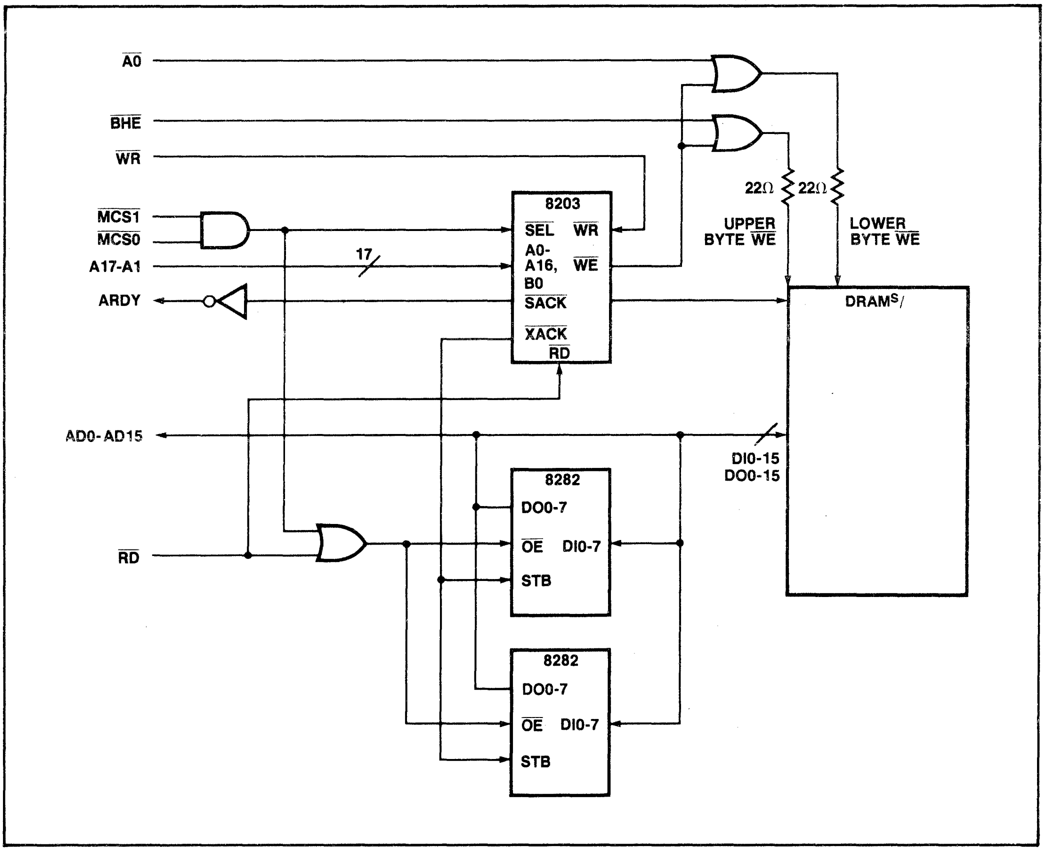
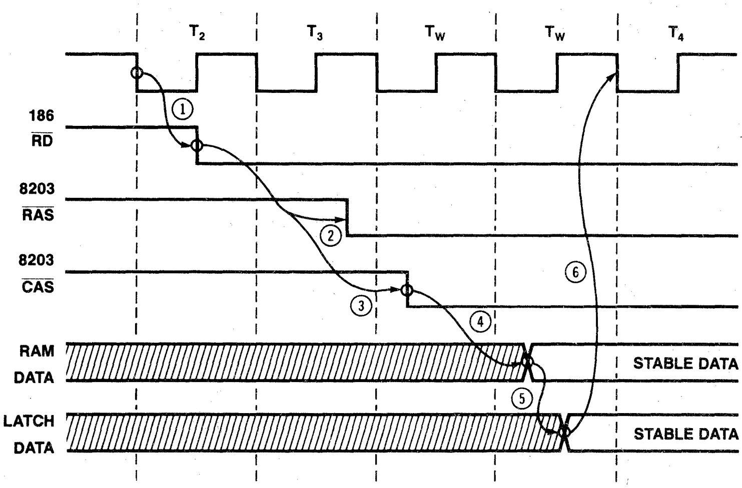


Figure 22. Example 8203/DRAM/80186 Interface



1. t_{CLEL}: Clock low until read low = 70 ns max
2. t_{CR}: Command active until RAS = 150 ns max*
3. t_{CC}: Command active until CAS = 245 ns max*
4. t_{CAC}: Access time from CAS = 85 ns max
5. t_{ISOU}: Input to output delay = 30 ns max
6. t_{DVCL}: Data valid to clock low (data in set up) = 20 ns min

Total Access Time = 70 + 245 +85 +30 +20 = 450 ns (3.6 T-states)

① & ⑥ are 186 specs
② & ③ are 8203 specs
④ is a 2164A-15 spec
⑤ is on 8282 spec

*Assumes 25MHz 8203 operation

Figure 23. 8203/2164A-15 Access Time Calculation

Since the 8203 is operating asynchronously to the 80186, the RDY output of the 8203 (used to suspend processor operation when a processor DRAM request coincides with a DRAM refresh cycle) must be synchronized to the 80186. The 80186 ARDY line is used to provide the necessary ready synchronization. The 8203 ready outputs operate in a normally not ready mode, that is, they are only driven active when an 8203 cycle is being executed, and a refresh cycle is not being run. This is fundamentally different than the normally ready mode used by the 2186 iRAMs (see previous section). The 8203 SACK signal is presented to the 80186 only when the DRAM is being accessed. Notice that the SACK output of the 8203 is used, rather than the XACK output. Since the 80186 will insert at least one full CPU clock cycle between the time RDY is sampled active, and the time data must be present on the data bus, using the XACK signal would insert unnecessary additional wait states, since it does not indicate ready until valid data is available from the memory.

For more information about 8203/DRAM interfacing and timing, please consult the 8203 data sheet, or AP97A, "Interfacing Dynamic RAM to iAPX86/88 Systems Using the Intel 8202A and 8203" by Brad May (April 1982).

3.2.4 8207 DRAM INTERFACE

The 8207 advanced dual-port DRAM controller provides a high performance DRAM memory interface specifically for 80186 or 80286 microcomputer systems. This controller provides all address multiplexing and DRAM refresh circuitry. In addition, it synchronizes and arbitrates memory requests from two different ports (e.g., an 80186 and a Multibus), allowing the two ports to share memory. Finally, the 8207 provides a simple interface to the 8206 error detection and correction chip.

The simplest 8207 (and also the highest performance) interface is shown in Figure 24. This shows the 80186 connected to an 8207 using the 8207 slow cycle, synchronous status interface. In this mode, the 8207 decodes the type of cycle to be run directly from the status lines of the 80186. In addition, since the 8207 CLOCKIN is driven by the CLOCKOUT of the 80186, any performance degradation caused by required memory request synchronization between the 80186 and the 8207 is not present. Finally, the entire memory array driven by the

8207 may be selected using one or a group of the 80186 memory chip selects, as in the 8203 interface above.

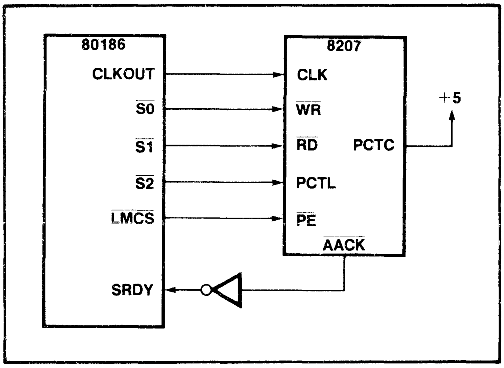


Figure 24. 80186/8207/DRAM Interface

The 8207 AACK signal may be used to generate a synchronous ready signal to the 80186 in the above interface. Since dynamic memory periodically requires refreshing, 80186 access cycles may occur simultaneously with an 8207 generated refresh cycle. When this occurs, the 8207 will hold the AACK line high until the processor initiated access is run (note, the sense of this line is reversed with respect to the 80186 SRDY input). This signal should be factored with the DRAM (8207) select input and used to drive the SRDY line of the 80186. Remember that only one of SRDY and ARDY needs to be active for a bus cycle to be terminated. If asynchronous devices (e.g., a Multibus interface) are connected to the ARDY line with the 8207 connected to the SRDY line, care must be taken in design of the ready circuit such that only one of the RDY lines is driven active at a time to prevent premature termination of the bus cycle.

3.3 HOLD/HLDA Interface

The 80186 employs a HOLD/HLDA bus exchange protocol. This protocol allows other asynchronous bus master devices (i.e., ones which drive address, data, and control information on the bus) to gain control of the bus to perform bus cycles (memory or I/O reads or writes).

3.3.1 HOLD RESPONSE

In the HOLD/HLDA protocol, a device requiring bus control (e.g., an external DMA device) raises the HOLD line. In response to this HOLD request, the 80186 will raise its HLDA line after it has finished its current bus activity. When the external device is finished with the bus, it drops its bus HOLD request. The 80186 responds by dropping its HLDA line and resuming bus operation.

When the 80186 recognizes a bus hold by driving HLDA high, it will float many of its signals (see Figure 25). AD0 - AD15 (address/data 0 - 15) and DEN (data enable) are floated within t_{CLAZ} (35ns) after the same clock edge that HLDA is driven active. A16–A19 (address 16 - 19), RD, WR, BHE (Bus High Enable), DT/R (Data Transmit/Receive) and S0 - S2 (status 0 - 2) are floated within t_{CHCZ} (45ns) after the clock edge immediately before the clock edge on which HLDA comes active.

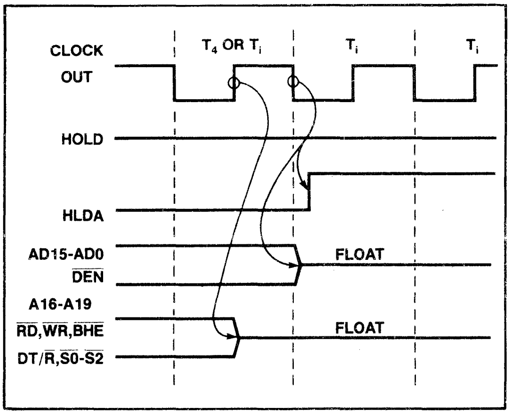


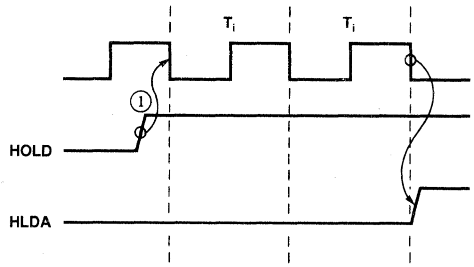
Figure 25. Signal Float/HLDA Timing of the 80186

Only the above mentioned signals are floated during bus HOLD. Of the signals not floated by the 80186, some have to do with peripheral functionality (e.g., TmrOut). Many others either directly or indirectly control bus devices. These signals are ALE (Address Latch Enable, see section 3.1.2) and all the chip select lines (UCS, LCS, MCS0-3, and PCS0-6). The designer must be aware that the chip select circuitry does not look at externally generated addresses (see section 10 for a discussion of the chip select logic). Thus, for memory or peripheral devices which are addressed by external bus master devices, discrete chip select and ready generation logic must be used.

3.3.2 HOLD/HLDA TIMING AND BUS LATENCY

The time required between HOLD going active and the 80186 driving HLDA active is known as bus latency. Many factors affect this latency, including synchronization delays, bus cycle times, locked transfer times and interrupt acknowledge cycles.

The HOLD request line is internally synchronized by the 80186, and may therefore be an asynchronous signal. To guarantee recognition on a certain clock edge, it must satisfy a certain setup and hold time to the falling edge of the CPU clock. A full CPU clock cycle is required for this synchronization, that is, the internal HOLD signal is not presented to the internal bus arbitration circuitry until one full clock cycle after it is latched from the HOLD input (see Appendix B for a discussion of 80186 synchronizers). If the bus is idle, HLDA will follow HOLD by two CPU clock cycles plus a small amount of setup and propagation delay time. The first clock cycle synchronizes the input; the second signals the internal circuitry to initiate a bus hold. (see Figure 26).

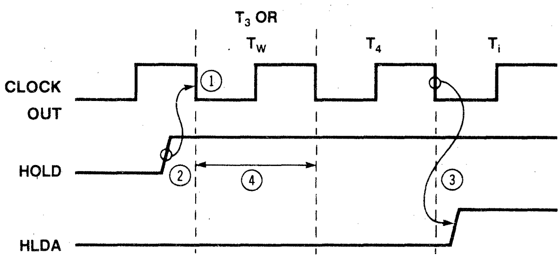


1. t_{HVCL}: Hold valid until clock low = 25 ns min
2. t_{CLHAV}: Clock low until HLDA active = 50 ns max

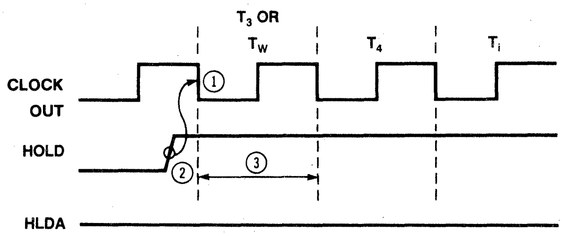
Figure 26. 80186 Idle Bus Hold/HLDA Timing

Many factors influence the number of clock cycles between a HOLD request and a HLDA. These may make bus latency longer than the best case shown above. Perhaps the most important factor is that the 80186 will not relinquish the local bus until the bus is idle. An idle bus occurs whenever the 80186 is not performing any bus transfers. As stated in section 3.1.1, when the bus is idle, the 80186 generates idle T-states. The bus can become idle only at the end of a bus cycle. Thus, the 80186 can recognize HOLD only after the end of its current bus cycle. The 80186 will normally insert no T_i states between T_4 and T_1 of the next bus cycle if it requires any bus activity (e.g., instruction fetches or I/O reads). However, the 80186 may not have an immediate need for the bus after a bus cycle, and will insert T_i states independent of the HOLD input (see section 3.1.7).

When the HOLD request is active, the 80186 will be



1. Decision: No additional internal bus cycles required, idle T-states will be inserted after T_4
2. Greater than 25 ns (t_{HVCL})
3. Less than 50 ns (t_{CLHAV})
4. HOLD request internally synchronized



1. Decision: Additional internal bus cycles required, no idle T-states will be inserted, Hold not active soon enough to force idle T-states
2. Greater than 25 ns (t_{HVCL}): not required since it will not get recognized anyway
3. HOLD request internally synchronized

Figure 27. HOLD/HLDA Timing in the 80186

210973-003

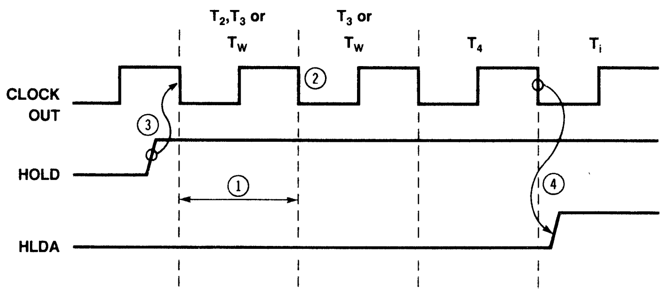


Figure 27A. HOLD/HLDA Timing in the 80186

forced to proceed from T_4 to T_i in order that the bus may be relinquished. HOLD must go active 3 T-states before the end of a bus cycle to force the 80186 to insert idle T-states after T_4 (one to synchronize the request, and one to signal the 80186 that T_4 of the bus cycle will be followed by idle T-states, see section 3.1.1). After the bus cycle has ended, the bus hold will be immediately acknowledged. If, however, the 80186 has already determined that an idle T-state will follow T_4 of the current bus cycle, HOLD need go active only 2 T-states before the end of a bus cycle to force the 80186 to relinquish the bus at the end of the current bus cycle. This is because the external HOLD request is not required to force the generation of idle T-states. Figure 27 graphically portrays the scenarios depicted above.

An external HOLD has higher priority than both the 80186 CPU or integrated DMA unit. However, an external HOLD will not separate the two cycles needed to perform a word access when the word accessed is located at an odd location (see section 3.1.3). In addition, an external HOLD will not separate the two-to-four bus cycles required to perform a DMA transfer using the integrated controller. Each of these factors will add additional bus cycle times to the bus latency of the 80186.

Another factor influencing bus latency time is locked transfers. Whenever a locked transfer is occurring, the 80186 will not recognize external HOLDs (nor will it recognize internal DMA bus requests). Locked transfers are programmed by preceding an instruction with the LOCK prefix. Any transfers generated by such a prefixed instruction will be locked, and will not be separated by any external bus requesting device. String instructions may be locked. Since string transfers may require thousands of bus cycles, bus latency time will suffer if they are locked.

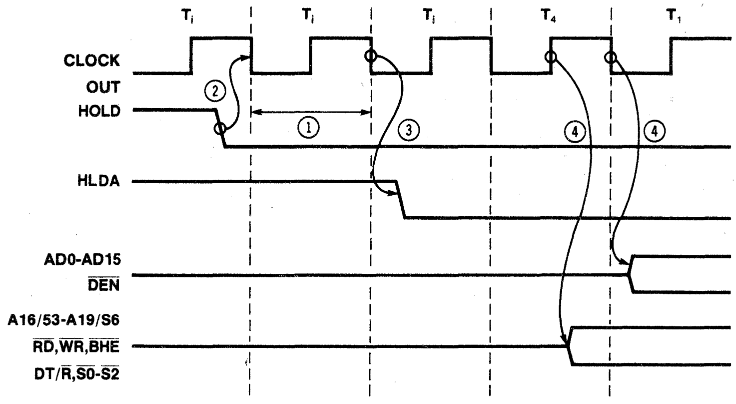
The final factor affecting bus latency time is interrupt acknowledge cycles. When an external interrupt controller is used, or if the integrated interrupt controller is used in iRMX 86 mode (see section 4.4.1) the 80186 will run two interrupt acknowledge cycles back to back. These cycles are automatically "locked" and will never be separated by any bus HOLD, either internal or external. See section 6.5 on interrupt acknowledge timing for more information concerning interrupt acknowledge timing.

3.3.3 COMING OUT OF HOLD

After the 80186 recognizes that the HOLD input has gone inactive, it will drop its HLDA line in a single clock. Figure 28 shows this timing. The 80186 will insert only two T_i after HLDA has gone inactive, assuming that the 80186 has internal bus cycles to run. During the last T_i, status information will go active concerning the bus cycle about to be run (see section 3.1.1). If the 80186 has no pending bus activity, it will maintain all lines floating (high impedance) until the last T_i before it begins its first bus cycle after the HOLD.

3.4 Differences Between the 8086 bus and the 80186 Bus

The 80186 bus was defined to be upward compatible with the 8086 bus. As a result, the 8086 bus interface components (the 8288 bus controller and the 8289 bus arbiter) may be used directly with the 80186. There are a few significant differences between the two processors which should be considered.



1. HOLD internally synchronized
2. Greater than 25 ns
3. Less than 50 ns
4. Lines come out of float only if a bus cycle is pending

Figure 28. 80186 Coming out of Hold

- **CPU Duty Cycle and Clock Generator**

The 80186 employs an integrated clock generator which provides a 50% duty cycle CPU clock (1/2 of the time it is high, the other 1/2 of the time it is low). This is different that the 8086, which employs an external clock generator (the 8284A) with a 33% duty cycle CPU clock (1/3 of the time it is high, the other 2/3 of the time, it is low). These differences manifest themselves as follows:

1) No oscillator output is available from the 80186, as it is available from the 8284A clock generator.
2) The 80186 does not provide a PCLK (50% duty cycle, 1/2 CPU clock frequency) output as does the 8284A.
3) The clock low phase of the 80186 is narrower, and the clock high phase is wider than on the same speed 8086.
4) The 80186 does not internally factor AEN with RDY. This means that if both RDY inputs (ARDY and SRDY) are used, external logic must be used to prevent the RDY not connected to a certain device from being driven active during an access to this device (remember, only one RDY input needs to be active to terminate a bus cycle, see section 3.1.6).
5) The 80186 concurrently provides both a single asynchronous ready input and a single synchronous ready input, while the 8284A provides either two synchronous ready inputs or two asynchronous ready inputs as a user strapable option.
6) The CLOCKOUT (CPU clock output signal) drive capacity of the 80186 is less than the CPU clock drive capacity of the 8284A. This means that not as many high speed devices (e.g., Schottky TTL flip-flops) may be connected to this signal as can be used with the 8284A clock output.
7) The crystal or external oscillator used by the 80186 is twice the CPU clock frequency, while the crystal or external oscillator used with the 8284A is three times the CPU clock frequency.

- **Local Bus Controller and Control Signals**

The 80186 simultaneously provides both local bus controller outputs ($\overline{RD}$, $\overline{WR}$, ALE, $\overline{DEN}$ and DT/$\overline{R}$) and status outputs ($\overline{S0}$, $\overline{S1}$, $\overline{S2}$) for use with the 8288 bus controller. This is different from the 8086 where the local bus controller outputs (generated only in min mode) are sacrificed if status outputs (generated only in max mode) are desired. These differences will manifest themselves in 8086 systems and 80186 systems as follows:

1) Because the 80186 can simultaneously provide local bus control signals and status outputs, many systems supporting both a system bus (e.g.,

a Multibus®) and a local bus will not require two separate external bus controllers, that is, the 80186 bus control signals may be used to control the local bus while the 80186 status signals are concurrently connected to the 8288 bus controller to drive the control signals of the system bus.

2) The ALE signal of the 80186 goes active a clock phase earlier on the 80186 then on the 8086 or 8288. This minimizes address propagation time through the address latches, since typically the delay time through these latches from inputs valid is less than the propagation delay from the strobe input active.

3) The 80186 $\overline{RD}$ input must be tied low to provide queue status outputs from the 80186 (see Figure 29). When so strapped into "queue status mode," the ALE and $\overline{WR}$ outputs provide queue status information. Notice that this queue status information is available one clock phase earlier from the 80186 than from the 8086 (see Figure 30).

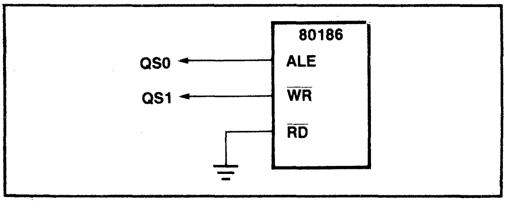


Figure 29. Generating Queue Status Information from the 80186

- **HOLD/HLDA vs. RQ/GT**

As discussed earlier, the 80186 uses a HOLD/HLDA type of protocol for exchanging bus mastership (like the 8086 in min mode) rather than the RQ/GT protocol used by the 8086 in max mode. This allows compatiblity with Intel's the new generation of high performance/high integration bus master peripheral devices (for example the 82586 Ethernet* controller or 82730 high performance CRT controller/text coprocessor).

- **Status Information**

The 80186 does not provide S3-S5 status information. On the 8086, S3 and S4 provide information regarding the segment register used to generate the physical address of the currently executing bus cycle. S5 provides information concerning the state of the interrupt enable flip-flop. These status bits are always low on the 80186.

Status signal S6 is used to indicate whether the current bus cycle is initiated by either the CPU or a DMA device. Subsequently, it is always low on the 8086. On the 80186, it is low whenever the current bus cycle is initiated by the 80186 CPU, and is high when the current bus cycle is initiated by the 80186 integrated DMA unit.

- **Bus Drive**

The 80186 output drivers will drive 200pF loads. This is double that of the 8086 (100pF). This allows larger systems to be constructed without the need for bus buffers. It also means that it is very important to provide good grounds to the 80186, since its large drivers can discharge its outputs very quickly causing large current transients on the 80186 ground pins.

- **Misc.**

The 80186 does not provide early and late write signals, as does the 8288 bus controller. The $\overline{WR}$ signal generated by the 80186 corresponds to the early write signal of the 8288. This means that data is not stable on the address/data bus when this signal is driven active.

The 80186 also does not provide differentiated I/O and memory read and write command signals. If these signals are desired, an external 8288 bus controller may be used, or the $\overline{S2}$ signal may be used to synthesize differentiated commands (see section 3.1.4).

*Ethernet is a registered trademark of Xerox Corp.

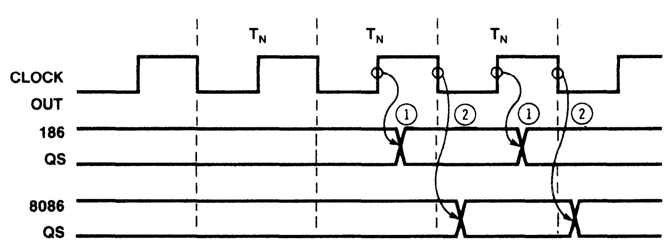


Figure 30. 80186 and 8086 Queue Status Generation

4. DMA UNIT INTERFACING

The 80186 includes a DMA unit which provides two independent high speed DMA channels. These channels operate independently of the CPU, and drive all integrated bus interface components (bus controller, chip selects, etc.) exactly as the CPU (see Figure 31). This means that bus cycles initiated by the DMA unit are exactly the same as bus cycles initiated by the CPU (except that S6 = 1 during all DMA initiated cycles, see section 3.1). Thus interfacing with the DMA unit itself is very simple, since except for the addition of the DMA request connection, it is exactly the same as interfacing to the CPU.

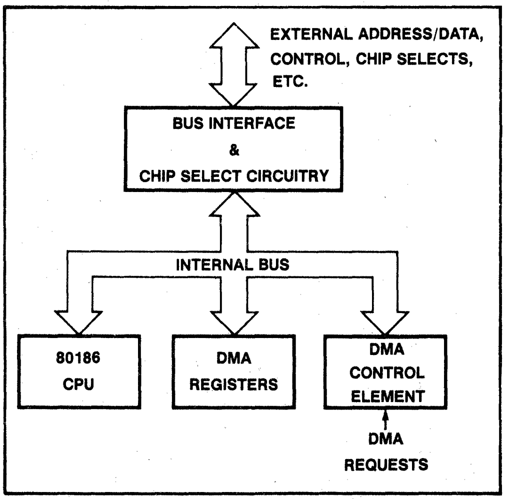


Figure 31. 80186 CPU/DMA Channel Internal Model

4.1 DMA Features

Each of the two DMA channels provides the following features:

- Independent 20-bit source and destination pointers which are used to access the I/O or memory location from which data will be fetched or to which data will be deposited
- Programmable auto-increment, auto-decrement or neither of the source and destination pointers after each DMA transfer
- Programmable termination of DMA activity after a certain number of DMA transfers
- Programmable CPU interruption at DMA termination
- Byte or word DMA transfers to or from even or odd memory or I/O addresses
- Programmable generation of DMA requests by:
  1) the source of the data
  2) the destination of the data
  3) timer 2 (see section 5)
  4) the DMA unit itself (continuous DMA requests)

4.2 DMA Unit Programming

Each of the two DMA channels contains a number of registers which are used to control channel operation. These registers are included in the 80186 integrated peripheral control block (see appendix A). These registers include the source and destination pointer registers, the transfer count register and the control register. The layout and interpretation of the bits in these registers is given in Figure 32.

The 20-bit source and destination pointers allow access to the complete 1 Mbyte address space of the 80186, and that all 20 bits are affected by the auto-increment or auto-decrement unit of the DMA (i.e., the DMA channels address the full 1 Mbyte address space of the 80186 as a flat, linear array without segments). When addressing I/O space, the upper 4 bits of the DMA pointer registers should be programmed to be 0. If they are not programmed 0, then the programmed value (greater than 64K in *I/O space*) will be driven onto the address bus (an area of I/O space not accessable to the CPU). The data transfer will occur correctly, however.

After every DMA transfer the 16-bit DMA transfer count register it is decremented by 1, whether a byte transfer or a word transfer has occurred. If the TC bit in the DMA control register is set, the DMA ST/$\overline{\text{STOP}}$ bit (see below) will be cleared when this register goes to 0, causing all DMA activity to cease. A transfer count of zero allows 65536 (2^{16}) transfers.

The DMA control register (see Figure 33) contains bits which control various channel characteristics, including for each of the data source and destination whether the pointer points to memory or I/O space, or whether the pointer will be incremented, decremented or left alone after each DMA transfer. It also contains a bit which selects between byte or word transfers. Two synchronization bits are used to determine the source of the DMA requests (see section 4.7). The TC bit determines whether DMA activity will cease after a programmed number of DMA transfers, and the INT bit is used to enable interrupts to the processor when this has occurred (note that an interrupt will not be generated to the CPU when the transfer count register reaches zero unless both the INT bit and the TC bit are set).

The control register also contains a start/stop (ST/$\overline{\text{STOP}}$) bit. This bit is used to enable DMA transfers. Whenever this bit is set, the channel is

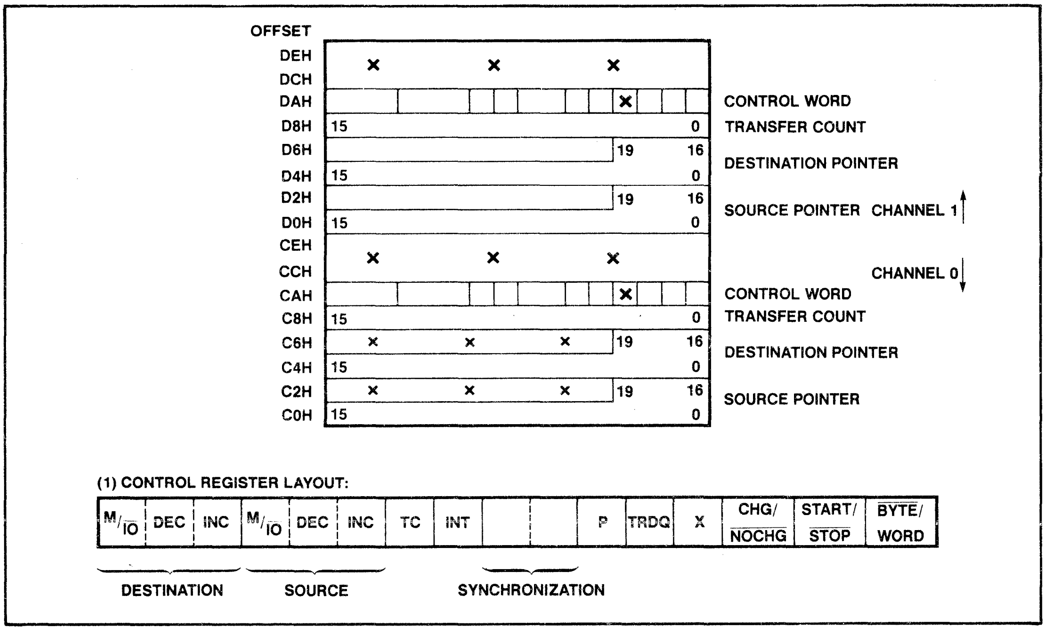


Figure 32. 80186 DMA Register Layout

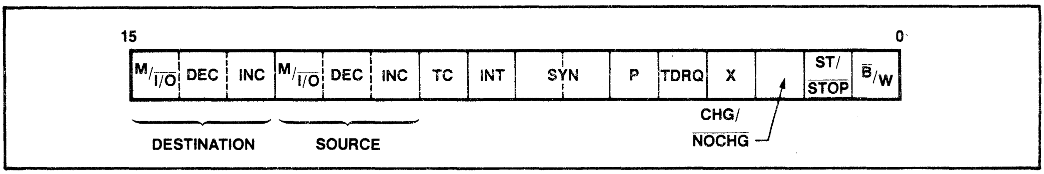


Figure 33. DMA Control Register

"armed," that is, a DMA transfer will occur whenever a DMA request is made to the channel. If this bit is cleared, no DMA transfers will be performed by the channel. A companion bit, the CHG/$\overline{\text{NOCHG}}$ bit, allows the contents of the DMA control register to be changed without modifying the state of the start/stop bit. The ST/$\overline{\text{STOP}}$ bit will only be modified if the CHG/$\overline{\text{NOCHG}}$ bit is also set during the write to the DMA control register. The CHG/$\overline{\text{NOCHG}}$ bit is write only. It will always be read back as a 0. Because DMA transfers could occur immediately after the ST/$\overline{\text{STOP}}$ bit is set, it should only be set only after all other DMA controller registers have been programmed. This bit is automatically cleared when the transfer count register reaches zero and the TC bit in the DMA control register is set, or when the transfer count register reaches zero and unsynchronized DMA transfers are programmed.

All DMA unit programming registers are directly accessable by the CPU. This means the CPU can, for example, modify the DMA source pointer register after 137 DMA transfers have occurred, and have the new pointer value used for the 138th DMA transfer. If more than one register in the DMA channel is being modified at any time that a DMA request may be generated and the DMA channel is enabled (the ST/$\overline{\text{STOP}}$ bit in the control register is set), the register programming values should be placed in memory locations and moved into the DMA registers using a locked string move instruction. This will prevent a DMA transfer from occurring after only half of the register values have changed. The above also holds true if a read/modify/write type of operation is being performed (e.g., ANDing off bits in a pointer register in a single AND instruction to a pointer register mapped into memory space).

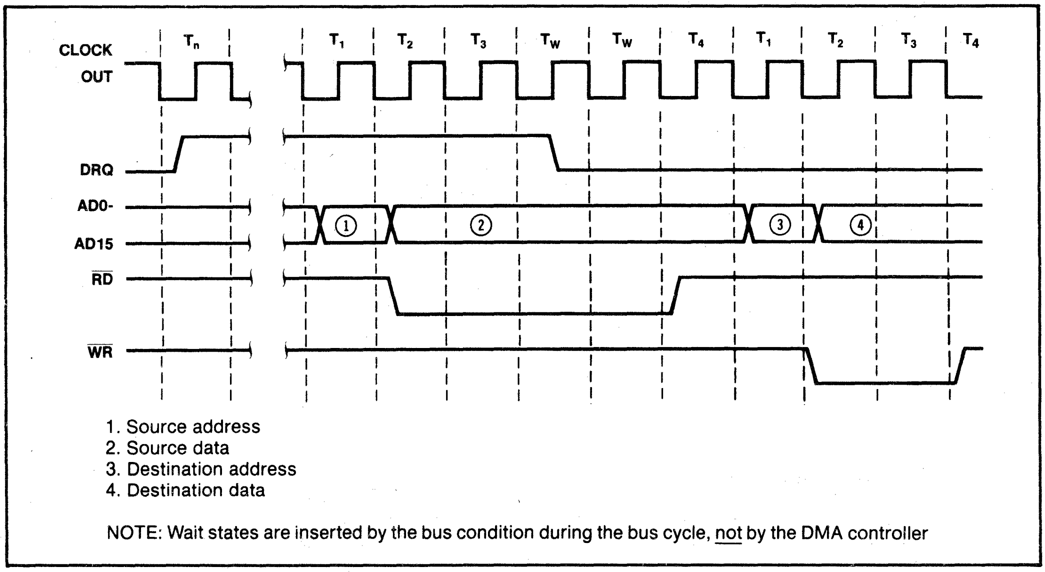


Figure 34. Example DMA Transfer Cycle on the 80186

4.3 DMA Transfers

Every DMA transfer in the 80186 consists of two independent bus cycles, the fetch cycle and the deposit cycle (see Figure 34). During the fetch cycle, the byte or word data is accessed from memory or I/O space using the address in the source pointer register. The data accessed is placed in an internal temporary register, which is not accessible to the CPU. During the deposit cycle, the byte or word data in this internal register is placed in memory or I/O space using the address in the destination pointer register. These two bus cycles will not be separated by bus HOLD or by the other DMA channel, and one will never be run without the other except when the CPU is RESET. Notice that the bus cycles run by the DMA unit are exactly the same as memory or I/O bus cycles run by the CPU. The only difference between the two is the state of the S6 status line (which is multiplexed on the A19 line): on all CPU initiated bus cycles, this status line will be driven low; on all DMA initiated bus cycles, this status line will be driven high.

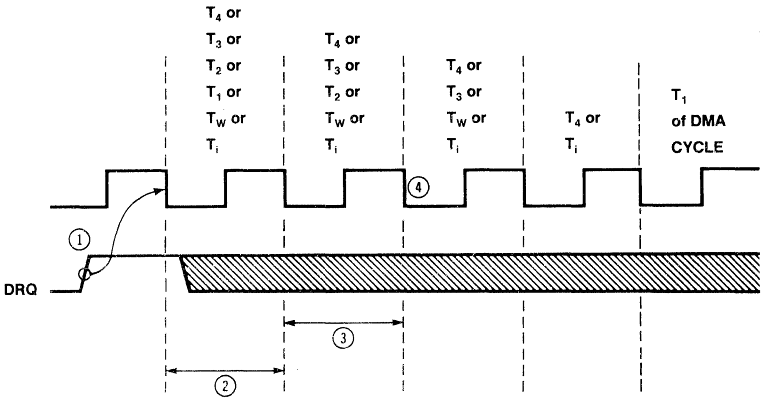
4.4 DMA Requests

Each DMA channel has a single DMA request line by which an external device may request a DMA transfer. The synchronization bits in the DMA control register determine whether this line is interpreted to be connected to the source of the DMA data or the destination of the DMA data. All transfer requests on this line are synchronized to the CPU clock before being presented to internal DMA logic. This means that any asynchronous transitions of the DMA request line will not cause the DMA channel to malfunction. In addition to external requests, DMA requests may be generated whenever the internal timer 2 times out, or continuously by programming the synchronization bits in the DMA control register to call for unsynchronized DMA transfers.

4.4.1 DMA REQUEST TIMING AND LATENCY

Before any DMA request can be generated, the 80186 internal bus must be granted to the DMA unit. A certain amount of time is required for the CPU to grant this internal bus to the DMA unit. The time between a DMA request being issued and the DMA transfer being run is known as DMA latency. Many of the issues concerning DMA latency are the same as those concerning bus latency (see section 3.3.2). The only important difference is that external HOLD always has bus priority over an internal DMA transfer. Thus, the latency time of an internal DMA cycle will suffer during an external bus HOLD.

Each DMA channel has a programmed priority relative to the other DMA channel. Both channels may be programmed to be the same priority, or one may be programmed to be of higher priority than the other channel. If both channels are active, DMA latency will suffer on the lower priority channel. If both channels are active and both channels are of the same programmed priority, DMA transfer cycles will alternate between the two channels (i.e., the first channel will perform a fetch and



1. t_{DRQCL} = DMA request to clock low = 25 ns min to guarantee recognition
2. Synchronizer resolution time
3. DMA unit priority arbitration, etc. time
4. Bus Interface Unit latches DMA request and decides to run DMA cycle

Figure 35. DMA Request Timing on the 80186 (showing minimum response time to request)

deposit, followed by a fetch and deposit by the second channel, etc).

The minimum timing required to generate a DMA cycle is shown in Figure 35. Note that the minimum time from DRQ becoming active until the beginning of the first DMA cycle is 4 CPU clock cycles, that is, a DMA request is sampled 4 clock cycles before the beginning of a bus cycle to determine if any DMA activity will be required. This time is independent of the number of wait states inserted in the bus cycle. The maximum DMA latency is a function of other processor activity (see above).

Also notice that if DRQ is sampled active at 1 in Figure 35, the DMA cycle will be executed, even if the DMA request goes inactive before the beginning of the first DMA cycle. This does not mean that the DMA request is latched into the processor such that any transition on the DMA request line will cause a DMA cycle eventually. Quite the contrary, DMA request must be active at a certain time before the end of a bus cycle for the DMA request to be recognized by the processor. If the DMA request line goes inactive before that window, then no DMA cycles will be run.

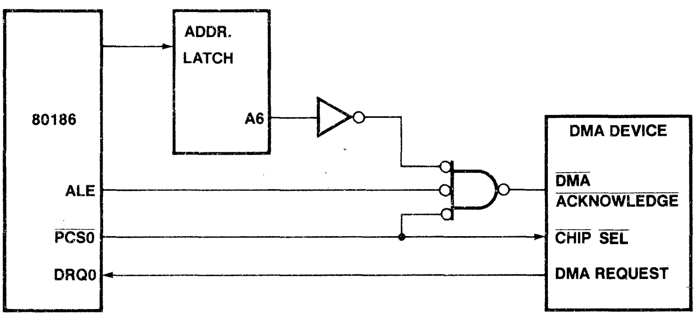


Figure 36. DMA Acknowledge Synthesis from the 80186

4.5 DMA Acknowledge

The 80186 generates no explicit DMA acknowledge signal. Instead, the 80186 performs a read or write directly to the DMA requesting device. If required, a DMA acknowledge signal can be generated by a decode of an address, or by merely using one of the $\overline{PCS}$ lines (see Figure 36). Note ALE must be used to factor the DACK because addresses are not guaranteed stable when chip selects go active. This is required because if the address is not stable when the $\overline{PCS}$ goes active, glitches can occur at the output of the DACK generation circuitry as the address lines change state. Once ALE has gone low, the addresses are guaranteed to have been stable for at least t_{AVAL} (30ns).

4.6 Internally Generated DMA Requests

There are two types in internally synchronized DMA transfers, that is, transfer initiated by a unit integrated in the 80186. These two types are transfers in which the DMA request is generated by timer 2, or where DMA request is generated by the DMA channel itself.

The DMA channel can be programmed such that whenever timer 2 reaches its maximum count, a DMA request will be generated. This feature is selected by setting the TDRQ bit in the DMA channel control register. A DMA request generated in this manner will be latched in the DMA controller, so that once the timer request has been generated, it cannot be cleared except by running the DMA cycle or by clearing the TDRQ bits in both DMA control registers. Before any DMA requests are generated in this mode, timer 2 must be initiated and enabled.

A timer requested DMA cycle being run by either DMA channel will reset the timer request. Thus, if both channels are using it to request a DMA cycle, only one DMA channel will execute a transfer for every timeout of timer 2. Another implication of having a single bit timer DMA request latch in the DMA controller is that if another timer 2 timeout occurs before a DMA channel has a chance to run a DMA transfer, the first request will be lost, i.e., only a single DMA transfer will occur, even though the timer has timed out twice.

The DMA channel can also be programmed to provide its own DMA requests. In this mode, DMA transfer cycles will be run continuously at the maximum bus bandwidth, one after the other until the preprogrammed number of DMA transfers (in the DMA transfer count register) have occurred. This mode is selected by programming the synchronization bits in the DMA control register for unsynchronized transfers. Note that in this mode, the DMA controller will monopolize the CPU bus, i.e., the CPU will not be able to perform opcode fetching, memory operations, etc., while the DMA transfers are occurring. Also notice that the DMA will only perform the number of transfers indicated in the maximum count register regardless of the state of the TC bit in the DMA control register.

4.7 Externally Synchronized DMA Transfers

There are two types of externally synchronized DMA transfers, that is, DMA transfers which are requested by an external device rather than by integrated timer 2 or by the DMA channel itself (in unsynchronized transfers). These are source synchronized and destination synchronized transfers. These modes are selected by programming the synchronization bits in the DMA channel control register. The only difference between the two is the time at which the DMA request pin is sampled to determine if another DMA transfer is immediately required after the currently executing DMA transfer. On source synchronized transfers, this is done such that two source synchronized DMA transfers may occur one immediately after the other, while on destination synchronized transfers a certain amount of idle time is automatically inserted between two DMA transfers to allow time for the DMA requesting device to drive its DMA request inactive.

4.7.1 SOURCE SYNCHRONIZED DMA TRANSFERS

In a source synchronized DMA transfer, the source of the DMA data requests the DMA cycle. An example of this would be a floppy disk read from the disk to main memory. In this type of transfer, the device requesting the transfer is read during the fetch cycle of the DMA transfer. Since it takes 4 CPU clock cycles from the time DMA request is sampled to the time the DMA transfer is actually begun, and a bus cycle takes a minimum of 4 clock cycles, the earliest time the DMA request pin will be sampled for another DMA transfer will be at the beginning of the deposit cycle of a DMA transfer. This allows over 3 CPU clock cycles between the time the DMA requesting device receives an acknowledge to its DMA request (around the beginning of T_2 of the DMA fetch cycle), and the time it must drive this request inactive (assuming no wait states) to insure that another DMA transfer is not performed if it is not desired (see Figure 37).

4.7.2 DESTINATION SYNCHRONIZED DMA TRANSFERS

In destination synchronized DMA transfers, the destination of the DMA data requests the DMA transfer. An example of this would be a floppy disk write from main memory to the disk. In this type of transfer, the device requesting the transfer is written during the deposit cycle of the DMA transfer. This causes a problem, since the DMA requesting device will not receive notification of the DMA cycle being run until 3 clock cycles before the end of the DMA transfer (if no wait states are being

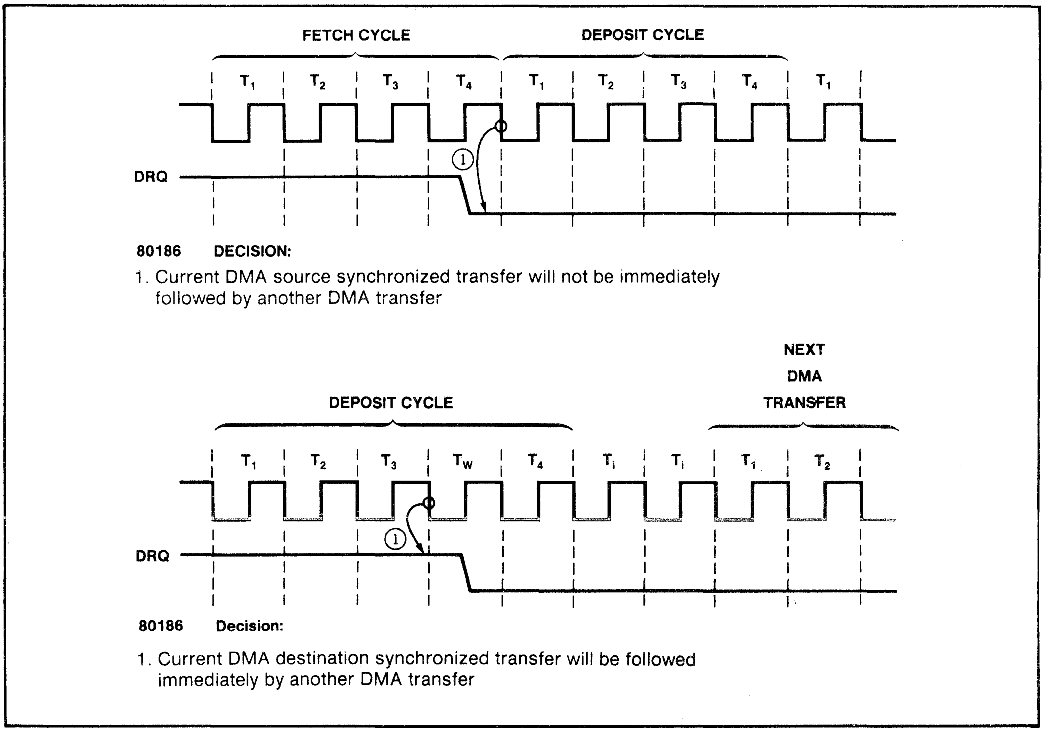
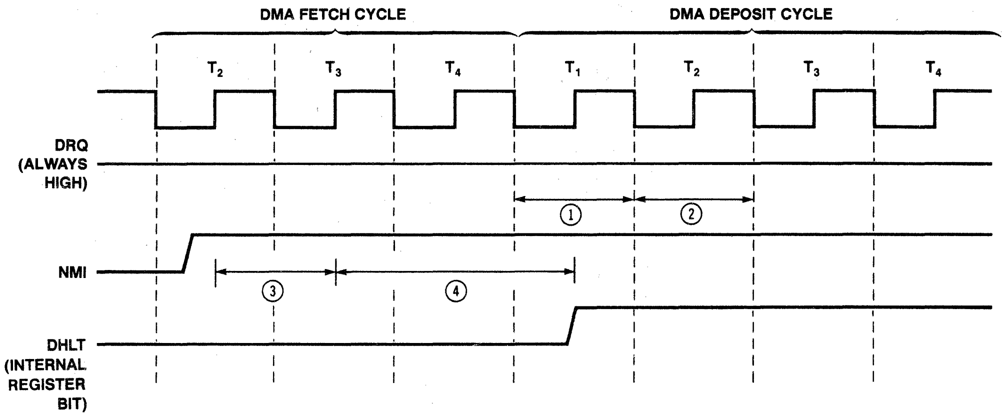


Figure 37. Source & Destination Synchronized DMA Request Timing

inserted into the deposit cycle of the DMA transfer) and it takes 4 clock cycles to determine whether another DMA cycle should be run immediately following the current DMA transfer. To get around this problem, the DMA unit will relinquish the CPU bus after each destination synchronized DMA transfer for at least 2 CPU clock cycles to allow the DMA requesting device time to drop its DMA request if it does not immediately desire another immediate DMA transfer. When the bus is relinquished by the DMA unit, the CPU may resume bus operation (e.g., instruction fetching, memory or I/O reads or writes, etc.) Thus, typically, a CPU initiated bus cycle will be inserted between each destination synchronized DMA transfer. If no CPU bus activity is required, however (and none can be guaranteed), the DMA unit will insert only 2 CPU clock cycles between the deposit cycle of one DMA transfer and the fetch cycle of the next DMA transfer. This means that the DMA destination requesting device must drop its DMA request at least two clock cycles before the end of the deposit cycle regardless of the number of wait states inserted into the bus cycle. Figure 37 shows the DMA request going away too late to prevent the immediate generation of another DMA transfer. Any wait states inserted in the deposit cycle of the DMA transfer will lengthen the amount of time from the beginning of the deposit cycle to the time DMA will be sampled for another DMA transfer. Thus, if the amount of time a device requires to drop its DMA request after receiving a DMA acknowledge from the 80186 is longer than the 0 wait state 80186 maximum (100 ns), wait states can be inserted into the DMA cycle to lengthen the amount of time the device has to drop its DMA request after receiving the DMA acknowledge. Table 4 shows the amount of time between the beginning of T_2 and the time DMA request is sampled as wait states are inserted in the DMA deposit cycle.

Table 4. DMA Request Inactive Timing

Number of Wait States	Max Time(ns) For DRQ Inactive From Start of T_2
0	100
1	225
2	350
3	475



1. DMA request synchronization
2. Decision: Will DMA cycle be run?
   Answer: No DMA request is active but DHLT is set (from NMI request)
3. NMI synchronization time
4. Logic delay time from synchronized NMI until DHLT set (note: DHLT is in the interrupt control status register)

Figure 38. NMI and DMA Interaction

4.8 DMA Halt and NMI

Whenever a Non-Maskable Interrupt is received by the 80186, all DMA activity will be suspended after the end of the current DMA transfer. This is performed by the NMI automatically setting the DMA Halt (DHLT) bit in the interrupt controller status register (see section 6.3.7). The timing of NMI required to prevent a DMA cycle from occurring is shown in Figure 38. After the NMI has been serviced, the DHLT bit should be cleared by the programmer, and DMA activity will resume exactly where it left off, i.e., none of the DMA registers will have been modified. The DMA Halt bit is not automatically reset after the NMI has been serviced. It is automatically reset by the IRET instruction. This DMA halt bit may also be set by the programmer to prevent DMA activity during any critical section of code.

4.9 Example DMA Interfaces

4.9.1 8272 FLOPPY DISK INTERFACE

An example DMA Interface to the 8272 Floppy Disk Controller is shown in Figure 39. This shows how a typical DMA device can be interfaced to the 80186. An example floppy disk software driver for this interface is given in Appendix C.

The data lines of the 8272 are connected, through buffers, to the 80186 AD0-AD7 lines. The buffers are required because the 8272 will not float its output drivers quickly enough to prevent contention with the 80186 driven address information after a read from the 8272 (see section 3.1.3).

DMA acknowledge for the 8272 is driven by an address decode within the region assigned to PCS2. If PCS2 is assigned to be active between I/O locations 0500H and 057FH, then an access to I/O location 0500H will enable only the chip select, while an access to I/O location 0510H will enable both the chip select and the DMA acknowledge. Remember, ALE must be factored into the DACK generation logic because addresses are not guaranteed stable when the chip selects become active. If ALE were not used, the DACK generation circuitry could glitch as address output changed state while the chip select was active.

Notice that the TC line of the 8272 is driven by a very similar circuit as the one generating DACK (except for the reversed sense of the output!). This line is used to terminate an 8272 command before the command has completed execution. Thus, the TC input to the 8272 is software driven in this case. Another method of driving the TC input would be to connect the DACK signal to one of the 80186 timers, and program the timer to out-

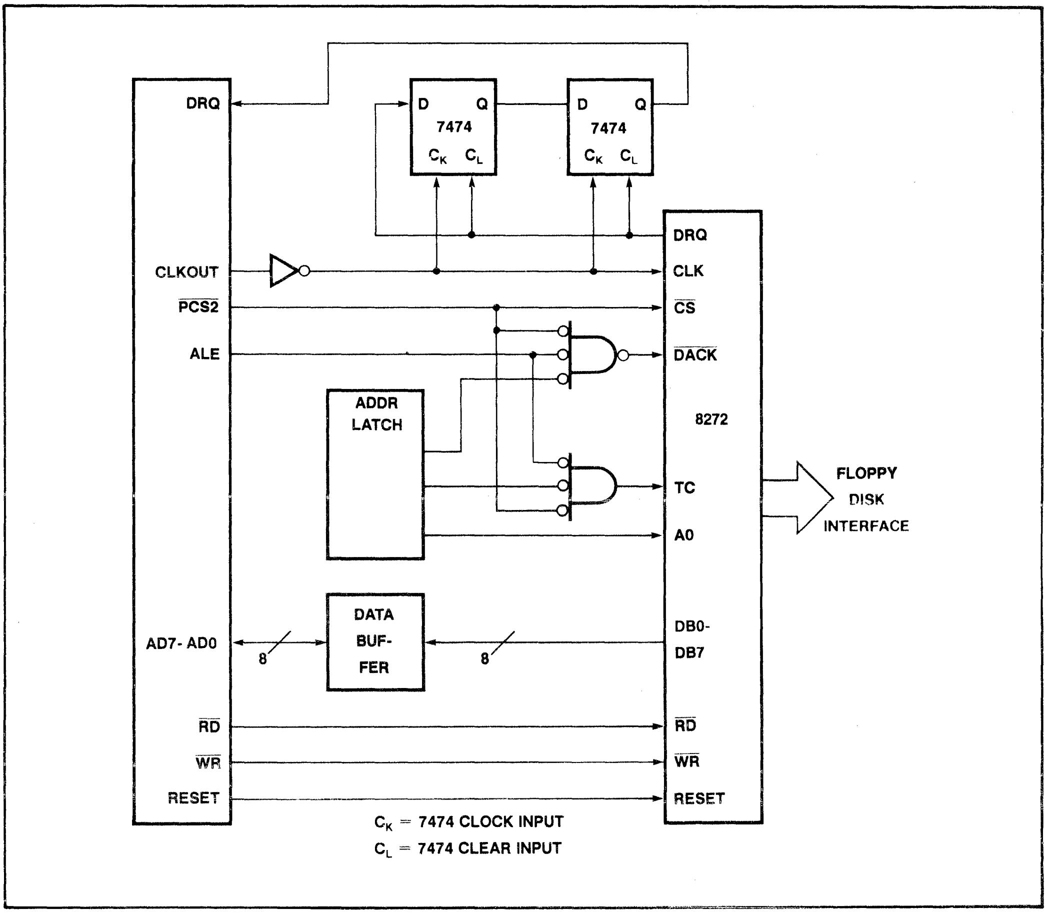


Figure 39. Example 8272/80186 DMA Interface

put a pulse to the 8272 after a certain number of DMA cycles have been run (see next section for 80186 timer information).

The above discussion assumed that a single 80186 $\overline{PCS}$ line is free to generate all 8272 select signals. If more than one chip select is free, however, different 80186 generated $\overline{PCS}$ lines could be used for each function. For example, $\overline{PCS2}$ could be used to select the 8272, $\overline{PCS3}$ could be used to drive the DACK line of the 8272, etc.

DMA requests are delayed by two clock periods in going from the 8272 to the 80186. This is required by the 8272 t_{RQR} (time from DMA request to DMA $\overline{RD}$ going active) spec of 800ns min. This requires 6.4 80186 CPU clock cycles (at 8 MHz), well beyond the 5 minimum provided by the 80186 (4 clock cycles to the beginning of the DMA bus cycle, 5 to the beginning of T_2 of the DMA bus cycle where $\overline{RD}$ will go active). The two flip-flops add two complete CPU clock cycles to this response time.

DMA request will go away 200ns after DACK is presented to the 8272. During a DMA write cycle (i.e., a destination synchronized transfer), this is not soon enough to prevent the immediate generation of another DMA transfer if no wait states are inserted in the deposit cycle to the 8272. Therefore, at least 1 wait state is required by this interface, regardless of the data access parameters of the 8272.

4.9.2 8274 SERIAL COMMUNICATION INTERFACE

An example 8274 synchronous/asynchronous serial chip/80186 DMA interface is shown in Figure 40. The 8274 interface is even simpler than the 8272 interface, since it does not require the generation of a DMA acknowledge signal, and the 8274 does not require the length of time between a DMA request and the DMA read or write cycle that the 8272 does. An example serial driver using the 8274 in DMA mode with the 80186 is given in Appendix C.

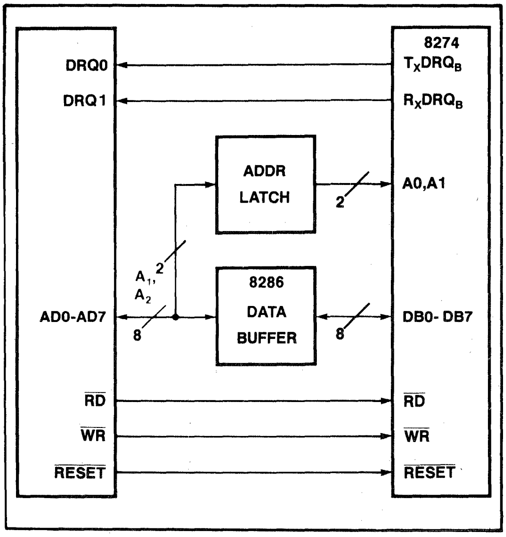


Figure 40. Example 8274/80186 DMA Interface

The data lines of the 8274 are connected through buffers to the 80186 AD0-AD7 lines. Again, these are required not because of bus drive problems, but because the 8274 will not float its drivers before the 80186 will begin driving address information on its address/data bus. If both the 8274 and the 8272 are included in the same 80186 system, they could share the same data bus buffer (as could any other peripheral devices in the system).

The 8274 does not require a DMA acknowledge signal. The first read from or write to the data register of the 8274 after the 8274 generates the DMA request signal will clear the DMA request. The time between when the control signal ($\overline{RD}$ or $\overline{WR}$) becomes active and when the 8274 will drop its DMA request during a DMA write is 150ns, which will require at least one wait state be inserted into the DMA write cycle for proper operation of the interface.

5. TIMER UNIT INTERFACING

The 80186 includes a timer unit which provides three independent 16-bit timers. These timers operate independently of the CPU. Two of these have input and output pins allowing counting of external events and generation of arbitrary waveforms. The third timer can be used as a timer, as a prescaler for the other two timers, or as a DMA request source.

5.1 Timer Operation

The internal timer unit on the 80186 could be modeled by a single counter element, time multiplexed to three register banks, each of which contains different control and count values. These register banks are, in turn, dual ported between the counter element and the 80186 CPU (see Figure 41). Figure 42 shows the timer element sequencing, and the subsequent constraints on input and output signals. If the CPU modifies one of the timer registers, this change will affect the counter element the next time that register is presented to the counter element. There is no connection between the sequencing of the counter element through the timer register banks and the Bus Interface Unit's sequencing through T-states. Timer operation and bus interface operation are completely asynchronous.

5.2 Timer Registers

Each timer is controlled by a block of registers (see Figure 43). Each of these registers can be read or written whether or not the timer is operating. All processor accesses to these registers are synchronized to all counter element accesses to these registers, meaning that one will never read a count register in which only half of the bits have been modified. Because of this synchronization, one wait state is automatically inserted into any access to the timer registers. Unlike the DMA unit, locking accesses to timer registers will not prevent the timer's counter element from accessing the timer registers.

Each timer has a 16-bit count register. This register is incremented for each timer event. A timer event can be a low-to-high transition on the external pin (for timers 0 and 1), a CPU clock transition (divided by 4 because of the counter element multiplexing), or a time out of timer 2 (for timers 0 and 1). Because the count register is 16 bits wide, up to 65536 (2^{16}) timer events can be counted by a single timer/counter. This register can be both read or written whether the timer is or is not operating.

Each timer includes a maximum count register. Whenever the timer count register is equal to the maximum count register, the count register will be reset to zero, that is, the maximum count value will never be stored in the count register. This maximum count value may be written while the timer is operating. A maximum count

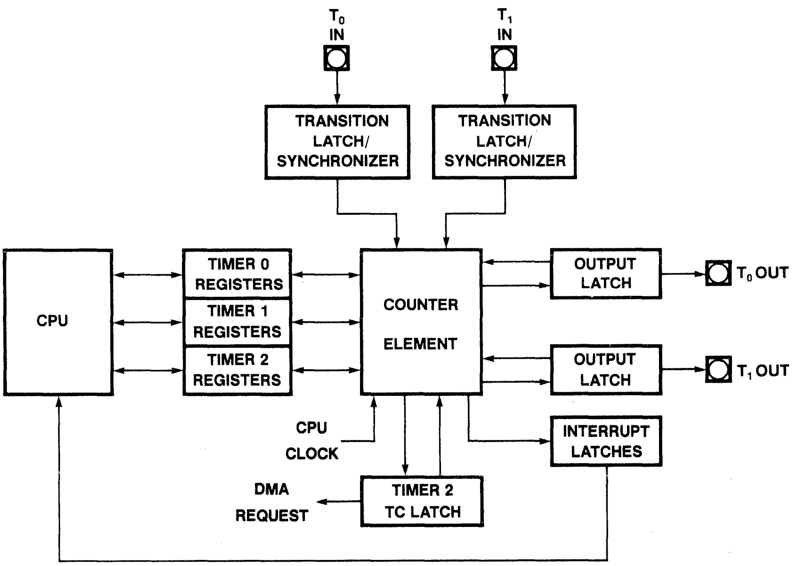
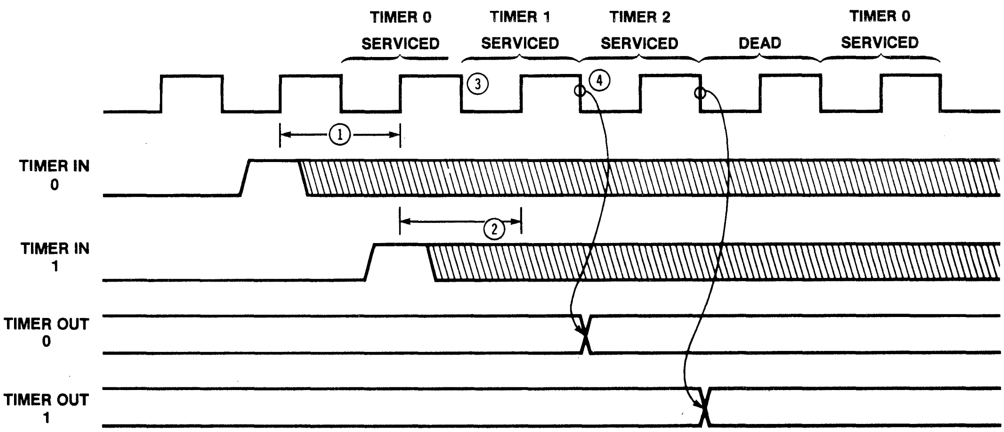


Figure 41. 80186 Timer Model



1. Timer in 0 resolution time
2. Timer in 1 resolution time
3. Modified count value written into 80186 timer 0 count register
4. Modified count value written into 80186 timer 1 count register

Figure 42. 80186 Counter Element Multiplexing and Timer Input Synchronization

OFFSET		
50H	COUNT REGISTER	TIMER 0
52H	MAX COUNT REGISTER A	
54H	MAX COUNT REGISTER B	
56H	CONTROL REGISTER (1)	
58H	COUNT REGISTER	TIMER 1
5AH	MAX COUNT REGISTER A	
5CH	MAX COUNT REGISTER B	
5EH	CONTROL REGISTER (1)	
60H	COUNT REGISTER	TIMER 2
62H	MAX COUNT REGISTER	
64H	X X X	
66H	CONTROL REGISTER (1)	

(1) CONTROL REGISTER LAYOUT

EN	$\overline{INH}$	INT	RIU		MC	RTG	P	EXT	ALT	CONT
15										0

Figure 43. 80186 Timer Register Layout

value of 0 implies a maximum count of 65536, a maximum count value of 1 implies a maximum count of 1, etc. The user should be aware that only equivalence between the count value and the maximum count register value is checked, that is, the count value will not be cleared if the value in the count register is greater than the value in the maximum count register. This could only occur by programmer intervention, either by setting the value in the count register greater than the value in the maximum count register, or by setting the value in the maximum count register to be less than the value in the count register. If this is programmed, the timer will count to the maximum possible count (FFFFH), increment to 0, then count up to the value in the maximum count register. The TC bit in the timer control register will not be set when the counter overflows to 0, nor will an interrupt be generated from the timer unit.

Timers 0 and 1 each contain an additional maximum count register. When both maximum count registers are used, the timer will first count up to the value in maximum count register A, reset to zero, count up to the value in maximum count register B, and reset to zero again. The ALTernate bit in the timer control register determines whether one or both maximum count registers are used. If this bit is low, only maximum count register A is used; maximum count register B is ignored. If it is high, both maximum count register A and maximum count register B are used. The RIU (register in use) bit in the timer control register indicates which maximum count register is currently being used. This bit is 0 when maximum count register A is being used, 1 when maximum count register B is being used. This RIU bit is read only. It is unaffected by any write to the timer control register. It will always be read 0 in single maximum count register mode (since only maximum count register A will be used).

Each timer can generate an interrupt whenever the timer count value reaches a maximum count value. That is, an interrupt can be generated whenever the value in maximum count register A is reached, and whenever the value in maximum count register B is reached. In addition, the MC (maximum count) bit in the timer control register is set whenever the timer count reaches a maximum count value. This bit is never automatically cleared, i.e., programmer intervention is required to clear this bit. If a timer generates a second interrupt request before the first interrupt request has been serviced, the first interrupt request to the CPU will be lost.

Each timer has an ENable bit in the timer control register. This bit is used to enable the timer to count. The timer will count timer events only when this bit is set. Any timer events occurring when this bit is reset are ignored. Any write to the timer control register will modify the ENable bit only if the INHibit bit is also set. The timer ENable bit will not be modified by a write to the timer control register if the INHibit bit is not set. The INHibit bit in the timer control register allows selective updating of the timer ENable bit. The value of the INHibit bit is not stored in a write to the timer control register; it will always be read as a 1.

Each timer has a CONTinuous bit in the timer control register. If this bit is cleared, the timer ENable bit will be automatically cleared at the end of each timing cycle. If a single maximum count register is used, the end of a timing cycle occurs when the count value resets to zero after reaching the value in maximum count register A. If dual maximum count registers are used, the end of a timing cycle occurs when the count value resets to zero after reaching the value in maximum count register B. If the CONTinuous bit is set, the ENable bit in the timer control register will never be automatically reset. Thus, after each timing cycle, another timing cycle will automatically begin. For example, in single maximum count register mode, the timer will count up to the value in maximum count register A, reset to zero, count up to the value in maximum count register A, reset to zero, ad infinitum. In dual maximum count register mode, the timer will count up the the value in maximum count register A, reset to zero, count up the value in maximum count register B, reset to zero, count up to the value in maximum count register A, reset to zero, et cetera.

5.3 Timer Events

Each timer counts timer events. All timers can use a transition of the CPU clock as an event. Because of the counter element multiplexing, the timer count value will be incremented every fourth CPU clock. For timer 2, this is the only timer event which can be used. For timers 0 and 1, this event is selected by clearing the EXTernal and Prescaler bits in the timer control register.

Timers 0 and 1 can use timer 2 reaching its maximum count as a timer event. This is selected by clearing the EXTernal bit and setting the Prescaler bit in the timer control register. When this is done, the timer will increment whenever timer 2 resets to zero having reached its own maximum count. Note that timer 2 must be initialized and running for the other timer's value to be incremented.

Timers 0 and 1 can also be programmed to count low-to-high transitions on the external input pin. Each transition on the external pin is synchronized to the 80186 clock before it is presented to the timer circuitry, and may, therefore, be asynchronous (see Appendix B for information on 80186 synchronizers). The timer counts transitions on the input pin: the input value must go low, then go high to cause the timer to increment. Any transition on this line is latched. If a transition occurs when a timer is not being serviced by the counter element, the transition on the input line will be remembered so that when the timer does get serviced, the input transition will be counted. Because of the counter element multiplexing, the maximum rate at which the timer can count is 1/4 of the CPU clock rate (2 MHz with an 8 MHz CPU clock).

5.4 Timer Input Pin Operation

Timers 0 and 1 each have individual timer input pins. All low-to-high transitions on these input pins are synchronized, latched, and presented to the counter element when the particular timer is being serviced by the counter element.

Signals on this input can affect timer operation in three different ways. The manner in which the pin signals are used is determined by the EXTernal and RTG (retrigger) bits in the timer control register. If the EXTernal bit is set, transitions on the input pin will cause the timer count value to increment if the timer is enabled (the ENable bit in the timer control register is set). Thus, the timer counts external events. If the EXTernal bit is cleared, all timer increments are caused by either the CPU clock or by timer 2 timing out. In this mode, the RTG bit determines whether the input pin will enable timer operation, or whether it will retrigger timer operation.

If the EXTernal bit is low and the RTG bit is also low, the timer will count internal timer events only when the timer input pin is high and the ENable bit in the timer control register is set. Note that in this mode, the pin is level sensitive, not edge sensitive. A low-to-high transition on the timer input pin is not required to enable timer operation. If the input is tied high, the timer will be continually enabled. The timer enable input signal is completely independent of the ENable bit in the timer control register: both must be high for the timer to count. Example uses for the timer in this mode would be a real time clock or a baud rate generator.

If the EXTernal bit is low and the RTG bit is high, the timer will act as a digital one-shot. In this mode, every low-to-high transition on the timer input pin will cause the timer to reset to zero. If the timer is enabled (i.e., the ENable bit in the timer control register is set) timer operation will begin (the timer will count CPU clock transitions or timer 2 timeouts). Timer operation will cease at the end of a timer cycle, that is, when the value in the maximum count register A is reached and the timer count value resets to zero (in single maximum count register mode, remember that the maximum count value is never stored in the timer count register) or when the value in maximum count register B is reached and the timer count value resets to zero (in dual maximum count register mode). If another low-to-high transition occurs on the input pin before the end of the timer cycle, the timer will reset to zero and begin the timing cycle again regardless of the state of the CONTinuous bit in the timer control register the RIU bit will not be changed by the input transition. If the CONTinuous bit in the timer control register is cleared, the timer ENable bit will automatically be cleared at the end of the timer cycle. This means that any additional transitions on the input pin will be ignored by the timer. If the CONTinuous bit in the timer control register is set, the timer will reset to zero and begin another timing cycle for every low-to-high transition on the input pin, regardless of whether the timer had reached the end of a timer cycle, because the timer ENable bit would not have been cleared at the end of the timing cycle. The timer will also continue counting at the end of a timer cycle, whether or not another transition has occurred on the input pin. An example use of the timer in this mode is an alarm clock time out signal or interrupt.

5.5 Timer Output Pin Operation

Timers 0 and 1 each contain a single timer output pin. This pin can perform two functions at programmer option. The first is a single pulse indicating the end of a timing cycle. The second is a level indicating the maximum count register currently being used. The timer outputs operate as outlined below whether internal or external clocking of the timer is used. If external clocking is used, however, the user should remember that the time between an external transition on the timer input pin and the time this transition is reflected in the timer out pin will vary depending on when the input transition occurs relative to the timer's being serviced by the counter element.

When the timer is in single maximum count register mode (the ALTernate bit in the timer control register is cleared) the timer output pin will go low for a single CPU clock the clock after the timer is serviced by the counter element where maximum count is reached (see Figure 44). This mode is useful when using the timer as a baud rate generator.

When the timer is programmed in dual maximum count register mode (the ALTernate bit in the timer control register is set), the timer output pin indicates which maximum count register is being used. It is low if maximum count register B is being used for the current count, high if maximum count register A is being used. If the timer is programmed in continuous mode (the CONTinuous bit in the timer control register is set), this pin could generate a waveform of any duty cycle. For example, if maximum count register A contained 10 and maximum count register B contained 20, a 33% duty cycle waveform would be generated.

5.6 Sample 80186 Timer Applications

The 80186 timers can be used for almost any application for which a discrete timer circuit would be used. These include real time clocks, baud rate generators, or event counters.

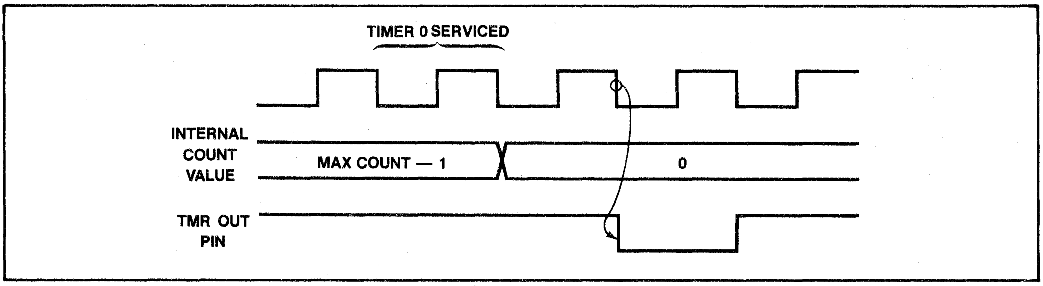


Figure 44. 80186 Timer Out Signal

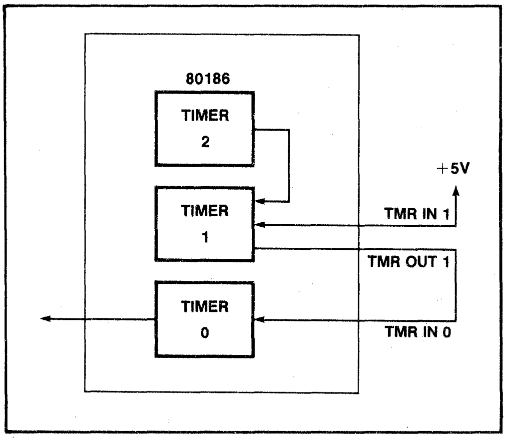


Figure 45. 80186 Real Time Clock

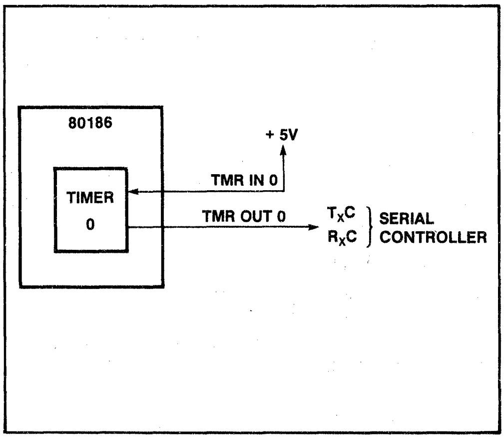


Figure 46. 80186 Baud Rate Generator

210973-003

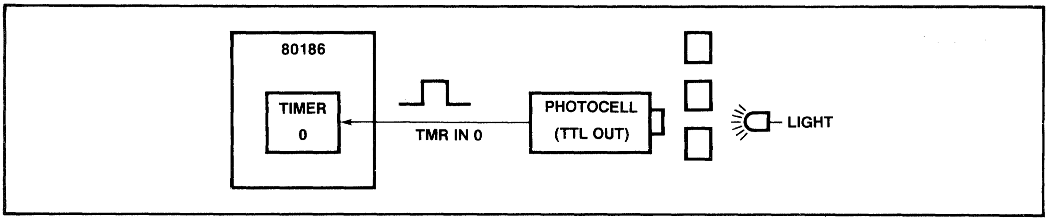


Figure 47.

5.6.1 80186 TIMER REAL TIME CLOCK

The sample program in appendix D shows the 80186 timer being used with the 80186 CPU to form a real time clock. In this implementation, timer 2 is programmed to provide an interrupt to the CPU every millisecond. The CPU then increments memory based clock variables.

5.6.2 80186 TIMER BAUD RATE GENERATOR

The 80186 timers can also be used as baud rate generators for serial communication controllers (e.g., the 8274). Figure 46 shows this simple connection, and the code to program the timer as a baud rate generator is included in appendix D.

5.6.3 80186 TIMER EVENT COUNTER

The 80186 timer can be used to count events. Figure 47 shows a hypothetical set up in which the 80186 timer will count the interruptions in a light source. The number of interruptions can be read directly from the count register of the timer, since the timer counts up, i.e., each interruption in the light source will cause the timer count value to increase. The code to set up the 80186 timer in this mode is included in appendix D.

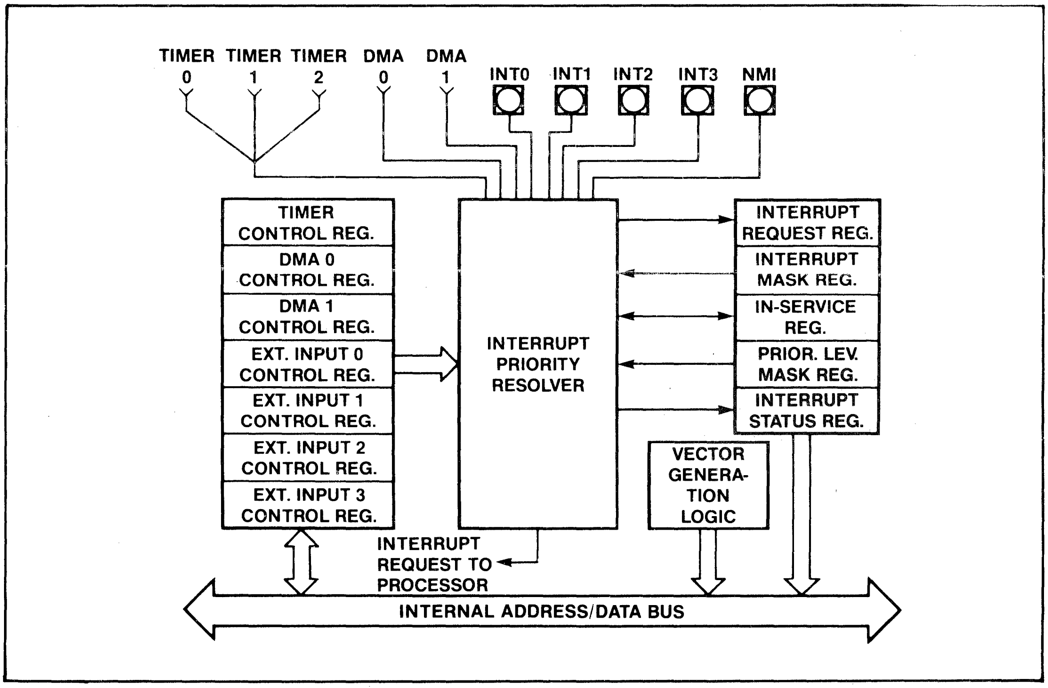


Figure 48. 80186 Interrupt Controller Block Diagram

6. 80186 INTERRUPT CONTROLLER INTERFACING

The 80186 contains an integrated interrupt controller. This unit performs tasks of the interrupt controller in a typical system. These include synchronization of interrupt requests, priortization of interrupt requests, and request type vectoring in response to a CPU interrupt acknowledge. It can be a master to two external 8259A interrupt controllers or can be a slave to an external interrupt controller to allow compatibility with the iRMX 86 operating system, and the 80130/80150 operating system firmware chips.

6.1 Interrupt Controller Model

The integrated interrupt controller block diagram is shown in Figure 48. It contains registers and a control element. Four inputs are provided for external interfacing to the interrupt controller. Their functions change according to the programmed mode of the interrupt controller. Like the other 80186 integrated peripheral registers, the interrupt controller registers are available for CPU reading or writing at any time.

6.2 Interrupt Controller Operation

The interrupt controller operates in two major modes, non-iRMX 86 mode (referred to henceforth as **master** mode), and **iRMX 86** mode. In master mode the integrated controller acts as the master interrupt controller for the system, while in iRMX 86 mode the controller operates as a slave to an external interrupt controller which operates as the master interrupt controller for the system. Some of the interrupt controller registers and interrupt controller pins change definition between these two modes, but the basic charter and function of the interrupt controller remains fundamentally the same. The difference is when in master mode, the interrupt controller presents its interrupt input directly to the 80186 CPU, while in iRMX 86 mode the interrupt controller presents its interrupt input to an external controller (which then presents its interrupt input to the 80186 CPU). Placing the interrupt controller in iRMX 86 mode is done by setting the iRMX mode bit in the peripheral control block pointer (see appendix A).

6.3 Interrupt Controller Registers

The interrupt controller has a number of registers which are used to control its operation (see Figure 49). Some of these change their function between the two major modes of the interrupt controller (master and iRMX 86 mode). The differences are indicated in the following section. If not indicated, the function and implementation of the registers is the same in the two basic modes of operation of the interrupt controller. The method of interaction among the various interrupt controller registers is shown in the flowcharts in Figures 57 and 58.

6.3.1 CONTROL REGISTERS

Each source of interrupt to the 80186 has a control register in the internal controller. These registers contain

MASTER MODE	OFFSET ADDRESS	iRMX86™ Mode
INT3 CONTROL REGISTER	3EH	(1)
INT2 CONTROL REGISTER	3CH	(1)
INT1 CONTROL REGISTER	3AH	TIMER 2 CONTROL REGISTER
INT0 CONTROL REGISTER	38H	TIMER 1 CONTROL REGISTER
DMA1 CONTROL REGISTER	36H	DMA1 CONTROL REGISTER
DMA0 CONTROL REGISTER	34H	DMA0 CONTROL REGISTER
TIMER CONTROL REGISTER	32H	TIMER 0 CONTROL REGISTER
INTERRUPT CONTROLLER STATUS REGISTER	30H	INTERRUPT CONTROLLER STATUS REGISTER
INTERRUPT REQUEST REGISTER	2EH	INTERRUPT REQUEST REGISTER
IN-SERVICE REGISTER	2CH	IN SERVICE REGISTER
PRIORITY MASK REGISTER	2AH	PRIORITY MASK REGISTER
MASK REGISTER	28H	MASK REGISTER
POLL STATUS REGISTER	26H	(1)
POLL REGISTER	24H	(1)
EOI REGISTER	22H	SPECIFIC EOI REGISTER
(1)	20H	INTERRUPT VECTOR REGISTER

1. Unsupported in this mode: values written may or may not be stored

Figure 49. 80186 Interrupt Controller Registers

15								0
0	⌇	0	SPECIAL FULLY NESTED BIT (2)	CAS-CADE MODE (2)	LEVEL TRIG. MODE (1)	MASK BIT	PRIORITY BITS	

1. This bit present only in INT0-INT3 control registers
2. These bits present only in INT0-INT1 control register

Figure 50. Interrupt Controller Control Register

MASTER MODE

15										0
X	X	X	I3	I2	I1	I0	D1	D0	X	TMR

iRMX™ 86 MODE

15								0
X	X	X	TMR 2	TMR 1	D0	D1	X	TMR 0

Figure 51. 80186 Interrupt Controller In-Service, Interrupt Request and Mask Register Format

three bits which select one of eight different interrupt priority levels for the interrupt device (0 is highest priority, 7 is lowest priority), and a mask bit to enable the interrupt (see Figure 50). When the mask bit is low, the interrupt is enabled, when it is high, the interrupt is masked.

There are seven control registers in the 80186 integrated interrupt controller. In master mode, four of these serve the external interrupt inputs, one each for the two DMA channels, and one for the collective timer interrupts. In iRMX 86 mode, the external interrupt inputs are not used, so each timer can have its own individual control register.

6.3.2 REQUEST REGISTER

The interrupt controller includes an interrupt request register (see Figure 51). This register contains seven active bits, one for each interrupt control register. Whenever an interrupt request is made by the interrupt source associated with a specific control register, the bit in interrupt request register is set, regardless if the interrupt is enabled, or if it is of sufficient priority to cause a processor interrupt. The bits in this register which are associated with integrated peripheral devices (the DMA and timer units) can be read or written, while the bits in this register which are associated with the external interrupt pins can only be read (values written to them are not stored). These interrupt request bits are automatically cleared when the interrupt is acknowledged.

6.3.3 MASK REGISTER AND PRIORITY MASK REGISTER

The interrupt controller contains a mask register (see Figure 51). This register contains a mask bit for each interrupt source associated with an interrupt control register. The bit for an interrupt source in the mask register is identically the same bit as is provided in the interrupt control register: modifying a mask bit in the control register will also modify it in the mask register, and vice versa.

The interrupt controller also contains a priority mask register (see Figure 52). This register contains three bits which indicate the lowest priority an interrupt may have that will cause an interrupt acknowledge. Interrupts received which have a lower priority will be effectively masked off. Upon reset this register is set to the lowest priority of 7 to enable all interrupts of any priority. This register may be read or written.

15							0
X	X	X	X	X	P2	P1	P0

Figure 52. 80186 Interrupt Controller Priority Mask Register Format

6.3.4 IN-SERVICE REGISTER

The interrupt controller contains an in-service register (see Figure 51). A bit in the in-service register is associated with each interrupt control register so that when an interrupt request by the device associated with the control register is acknowledged by the processor (either by the processor running the interrupt acknowledge or by the processor reading the interrupt poll register) the bit is set. The bit is reset when the CPU issues an End Of Interrupt to the interrupt controller. This register may be both read and written, i.e., the CPU may set in-service bits without an interrupt ever occurring, or may reset them without using the EOI function of the interrupt controller.

6.3.5 POLL AND POLL STATUS REGISTERS

The interrupt controller contains both a poll register and a poll status register (see Figure 53). Both of these registers contain the same information. They have a single bit to indicate an interrupt is pending. This bit is set if an interrupt of sufficient priority has been received. It is automatically cleared when the interrupt is acknowledged. If (and only if) an interrupt is pending, they also contain information as to the interrupt type of the highest priority interrupt pending.

15									0
INT REQ	X	X	X	X	S4	S3	S2	S1	S0

S0-S4 = interrupt type

Figure 53. 80186 Poll & Poll Status Register Format

Reading the poll register will acknowledge the pending interrupt to the interrupt controller just as if the processor had acknowledged the interrupt through interrupt acknowledge cycles. The processor will not actually run any interrupt acknowledge cycles, and will not vector through a location in the interrupt vector table. Only the interrupt request, in-service and priority mask registers in the interrupt controller are set appropriately. Reading the poll status register will merely transmit the status of the polling bits without modifying any of the other interrupt controller registers. These registers are read only: data written to them is not stored. These registers are not supported in iRMX 86 mode. The state of the bits in these registers in iRMX 86 mode is not defined.

6.3.6 END OF INTERRUPT REGISTER

The interrupt controller contains an End Of Interrupt register (see Figure 54). The programmer issues an End Of Interrupt to the controller by writing to this register. After receiving the End Of Interrupt, the interrupt controller automatically resets the in-service bit for the interrupt. The value of the word written to this register determines whether the End Of Interrupt is specific or non-specific. A non-specific End Of Interrupt is specified by setting the non-specific bit in the word written to the End Of Interrupt register. In a non-specific End Of Interrupt, the in-service bit of the highest priority interrupt set is automatically cleared, while a specific End Of Interrupt allows the in-service bit cleared to be explicitly specified. The in-service bit is reset whether the bit was set by an interrupt acknowledge or if it was set by the CPU writing the bit directly to the in-service register. If the

MASTER MODE

15									0
SPEC/$\overline{NSPEC}$	X	X	X	X	S4	S3	S2	S1	S0

S0-S4 = interrupt type

iRMX 86 MODE

15								0
X	X	X	X	X	X	L2	L1	L0

L0–L2 = interrupt priority level

Figure 54. 80186 End of Interrupt Register Format

15									0
DHLT	X	X	X	X	X	X	T_2	T_1	T_0

Figure 55. 80186 Interrupt Status Register Format

15											0
X	X	X	X	t_4	t_3	t_2	t_1	t_0	0	0	0

Figure 56. 80186 Interrupt Vector Register Format (iRMX 86 mode only)

highest priority interrupt is reset, the priority mask register bits will change to reflect the next lowest priority interrupt to be serviced. If a less than highest priority interrupt in-service bit is reset, the priority mask register bits will not be modified (because the highest priority interrupt being serviced has not changed). Only the specific EOI is supported in iRMX 86 mode. This register is write only: data written is not stored and cannot be read back.

6.3.7 INTERRUPT STATUS REGISTER

The interrupt controller also contains an interrupt status register (see Figure 55). This register contains four significant bits. There are three bits used to show which timer is causing an interrupt. This is required because in master mode, the timers share a single interrupt control register. A bit in this register is set to indicate which timer has generated an interrupt. The bit associated with a timer is automatically cleared after the interrupt request for the timer is acknowledged. More than one of these bits may be set at a time. The fourth bit in the interrupt status register is the DMA halt bit. When set, this bit prevents any DMA activity. It is automatically set whenever a NMI is received by the interrupt controller. It can also be set explicitly by the programmer. This bit is automatically cleared whenever the IRET instruction is executed. All significant bits in this register are read/write.

6.3.8 INTERRUPT VECTOR REGISTER

Finally, in iRMX 86 mode only, the interrupt controller contains an interrupt vector register (see Figure 56). This register is used to specify the 5 most significant bits of the interrupt type vector placed on the CPU bus in response to an interrupt acknowledgement (the lower 3 significant bits of the interrupt type are determined by the priority level of the device causing the interrupt in iRMX 86 mode).

6.4 Interrupt Sources

The 80186 interrupt controller receives and arbitrates among many different interrupt request sources, both internal and external. Each interrupt source may be programmed to be a different priority level in the interrupt controller. An interrupt request generation flow chart is shown in Figure 57. Such a flowchart would be followed independently by each interrupt source.

6.4.1 INTERNAL INTERRUPT SOURCES

The internal interrupt sources are the three timers and the two DMA channels. An interrupt from each of these interrupt sources is latched in the interrupt controller, so that if the condition causing the interrupt is cleared in the originating integrated peripheral device, the interrupt request will remain pending in the interrupt controller. The state of the pending interrupt can be obtained by reading the interrupt request register of the interrupt controller. For all internal interrupts, the latched interrupt request can be reset by the processor by writing to the interrupt request register. Note that all timers share a common bit in the interrupt request register in master mode. The interrupt controller status register may be read to determine which timer is actually causing the interrupt request in this mode. Each timer has a unique interrupt vector (see section 6.5.1). Thus polling is not required to determine which timer has caused the interrupt in the interrupt service routine. Also, because the timers share a common interrupt control register, they are placed at a common priority level as referenced to all other interrupt devices. Among themselves they have a fixed priority, with timer 0 as the highest priority timer and timer 2 as the lowest priority timer.

6.4.2 EXTERNAL INTERRUPT SOURCES

The 80186 interrupt controller will accept external interrupt requests only when it is programmed in master mode. In this mode, the external pins associated with the interrupt controller may serve either as direct interrupt inputs, or as cascaded interrupt inputs from other interrupt controllers as a programmed option. These options are selected by programming the C and SFNM bits in the INT0 and INT1 control registers (see Figure 50).

When programmed as direct interrupt inputs, the four interrupt inputs are each controlled by an individual interrupt control register. As stated earlier, these registers contain 3 bits which select the priority level for the interrupt and a single bit which enables the interrupt source to the processor. In addition each of these control registers contains a bit which selects either edge or level triggered mode for the interrupt input. When edge triggered mode is selected, a low-to-high transition must occur on the interrupt input before an interrupt is generated, while in level triggered mode, only a high level needs to be maintained to generate an interrupt. In edge triggered mode, the input must remain low at least 1 clock cycle before the input is "re-armed." In both modes, the interrupt level must remain high until the interrupt is acknowledged, i.e., the interrupt request is not latched in the interrupt controller. The status of the interrupt input can be shown by reading the interrupt request register. Each of the external pins has a bit in this register which indicates an interrupt request on the particular pin. Note that since interrupt requests on these inputs are not latched by the interrupt controller, if the external input goes inactive, the interrupt request (and also the bit in the interrupt request register) will also go inactive (low). Also, if the interrupt input is in edge triggered mode, a low-to-high transition on the input pin must occur before the interrupt request bit will be set in the interrupt request register.

If the C (Cascade) bit of the INT0 or INT1 control registers are set, the interrupt input is cascaded to an external interrupt controller. In this mode, whenever the

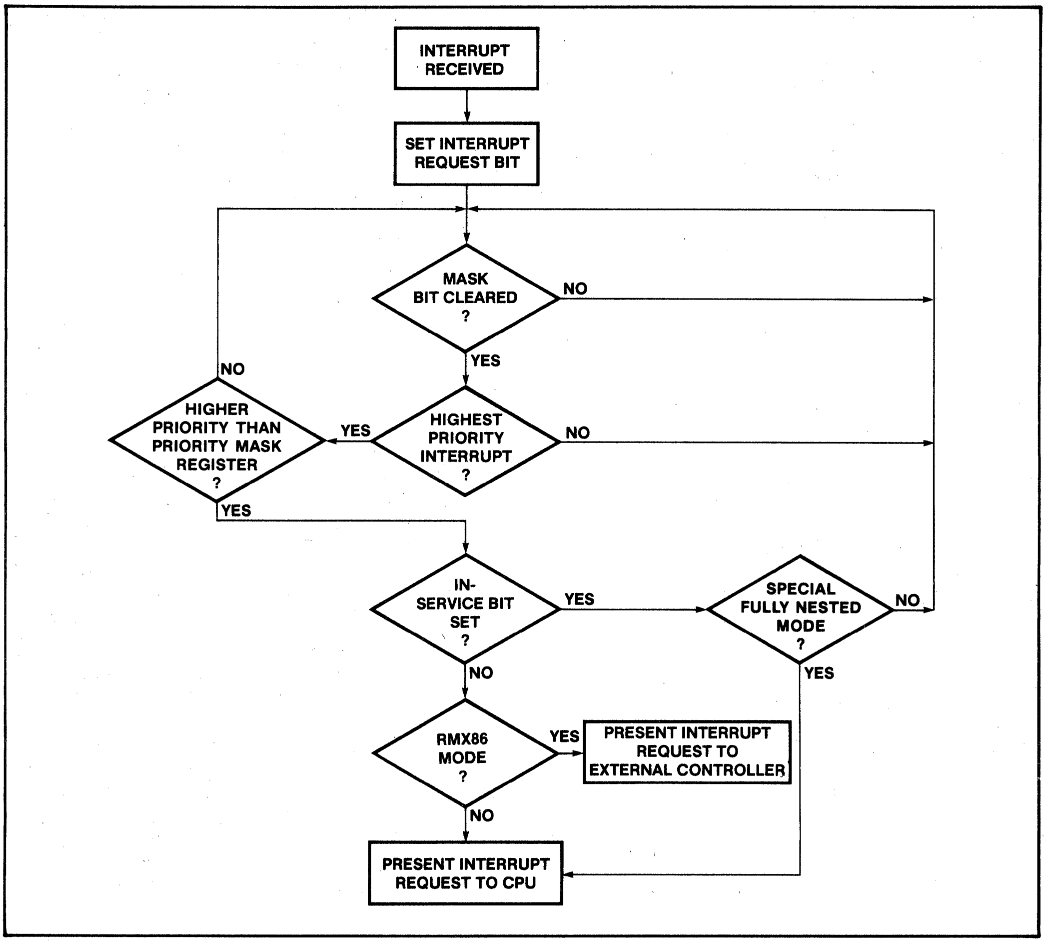


Figure 57. 80186 Interrupt Request Sequencing

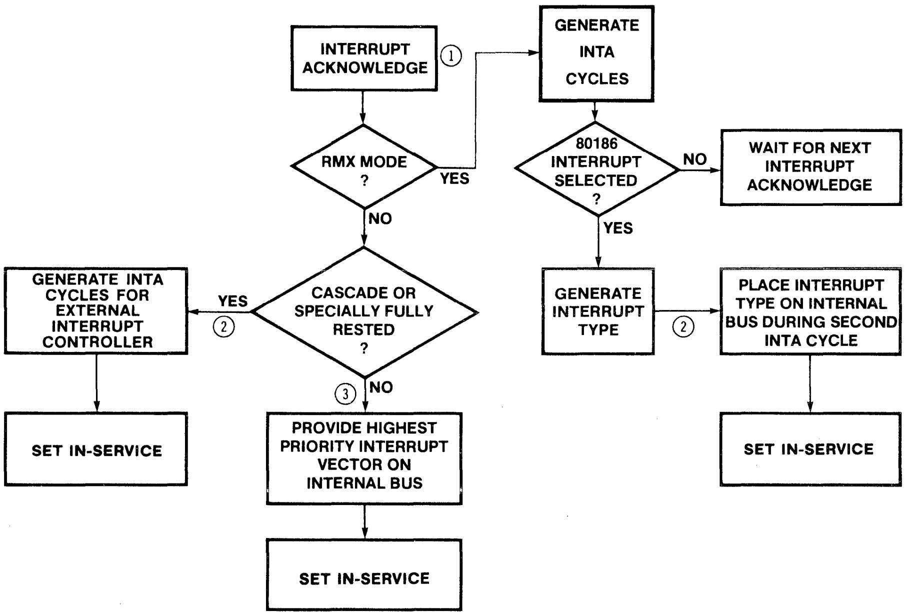
interrupt presented to the INT0 or INT1 line is acknowledged, the integrated interrupt controller will not provide the interrupt type for the interrupt. Instead, two INTA bus cycles will be run, with the INT2 and INT3 lines providing the interrupt acknowledge pulses for the INT0 and the INT1 interrupt requests respectively. INT0/INT2 and INT1/INT3 may be individually programmed into cascade mode. This allows 128 individually vectored interrupt sources if two banks of 9 external interrupt controllers each are used.

6.4.3 iRMX™ 86 MODE INTERRUPT SOURCES

When the interrupt controller is configured in iRMX 86 mode, the integrated interrupt controller accepts interrupt requests only from the integrated peripherals. Any external interrupt requests must go through an external interrupt controller. This external interrupt controller requests interrupt service directly from the 80186 CPU through the INT0 line on the 80186, In this mode, the function of this line is not affected by the integrated interrupt controller. In addition, in iRMX 86 mode the integrated interrupt controller must request interrupt service through this external interrupt controller. This interrupt request is made on the INT3 line (see section 6.7.4 on external interrupt connections).

6.5 Interrupt Response

The 80186 can respond to an interrupt in two different ways. The first will occur if the internal controller is pro-



1. Before actual interrupt acknowledge is run by CPU
2. Two interrupt acknowledge cycles will be run, the interrupt type is read by the CPU on the second cycle
3. Interrupt acknowledge cycles will not be run, the interrupt vector address is placed on an internal bus and is not available outside the processor
4. Interrupt type is not driven on external bus in iRMX86 mode

Figure 58. 80186 Interrupt Acknowledge Sequencing

viding the interrupt vector information with the controller in master mode. The second will occur if the CPU reads interrupt type information from an external interrupt controller or if the interrupt controller is in iRMX 86 mode. In both of these instances the interrupt vector information driven by the 80186 integrated interrupt controller is not available outside the 80186 microprocessor.

In each interrupt mode, when the integrated interrupt controller receives an interrupt response, the interrupt controller will automatically set the in-service bit and reset the interrupt request bit in the integrated controller. In addition, unless the interrupt control register for the interrupt is set in Special Fully Nested Mode, the interrupt controller will prevent any interrupts from occurring from the same interrupt line until the in-service bit for that line has been cleared.

6.5.1 INTERNAL VECTORING, MASTER MODE

In master mode, the interrupt types associated with all the interrupt sources are fixed and unalterable. These interrupt types are given in Table 5. In response to an internal CPU interrupt acknowledge the interrupt controller will generate the vector address rather than the interrupt type. On the 80186 (like the 8086) the interrupt vector address is the interrupt type multiplied by 4. This speeds interrupt response.

In master mode, the integrated interrupt controller is the master interrupt controller of the system. As a result, no external interrupt controller need know when the integrated controller is providing an interrupt vector, nor when the interrupt acknowledge is taking place. As a result, no interrupt acknowledge bus cycles will be generated. The first external indication that an interrupt has been acknowledged will be the processor reading the interrupt vector from the interrupt vector table in low memory.

Table 5. 80186 Interrupt Vector Types

Interrupt Name	Vector Type	Default Priority
timer 0	8	0a
timer 1	18	0b
timer 2	19	0c
DMA 0	10	2
DMA 1	11	3
INT 0	12	4
INT 1	13	5
INT 2	14	6
INT 3	15	7

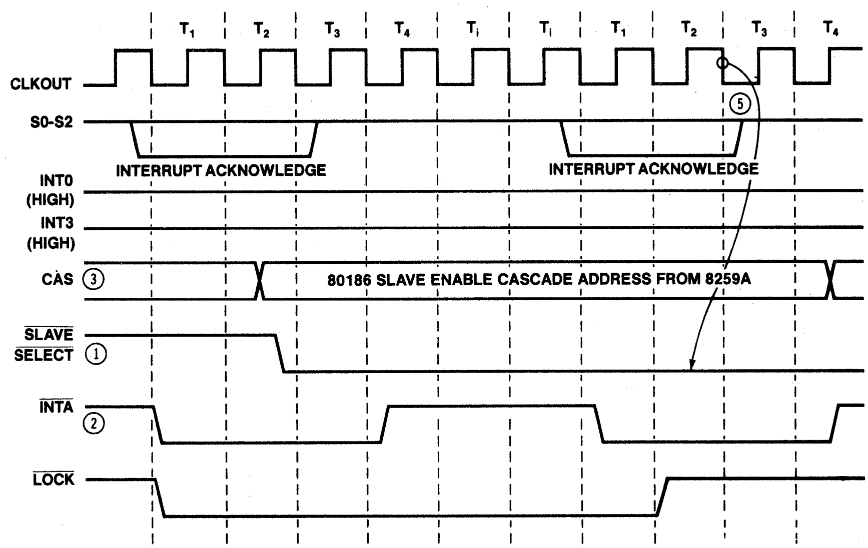
Because the two interrupt acknowledge cycles are not run, and the interrupt vector address does not need be be calculated, interrupt response to an internally vectored interrupt is 42 clock cycles, which is faster then the interrupt response when external vectoring is required, or if the interrupt controller is run in iRMX 86 mode.

If two interrupts of the same programmed priority occur, the default priority scheme (as shown in table 5) is used.

6.5.2 INTERNAL VECTORING, iRMX™ 86 MODE

In iRMX 86 mode, the interrupt types associated with the various interrupt sources are alterable. The upper 5 most significant bits are taken from the interrupt vector register, and the lower 3 significant bits are taken from the priority level of the device causing the interrupt. Because the interrupt type, rather than the interrupt vector address, is given by the interrupt controller in this mode the interrupt vector address must be calculated by the CPU before servicing the interrupt.

In iRMX 86 mode, the integrated interrupt controller will present the interrupt type to the CPU in response to the two interrupt acknowledge bus cycles run by the processor. During the first interrupt acknowledge cycle, the external master interrupt controller determines which slave interrupt controller will be allowed to place its interrupt vector on the microprocessor bus. During the second interrupt acknowledge cycle, the processor reads the interrupt vector from its bus. Thus, these two interrupt acknowledge cycles must be run, since the integrated controller will present the interrupt type information only when the external interrupt controller signals the integrated controller that it has the highest pending interrupt request (see Figure 59). The 80186 samples the



1. SLAVE SELECT = INT1
2. INTA = INT2
3. Driven by external interrupt controller
4. SLAVE SELECT must be driven before Phase 2 of T_2 of the second INTA cycle
5. SLAVE SELECT read by 80186

Figure 59. 80186 iRMX-86 Mode Interrupt Acknowledge Timing

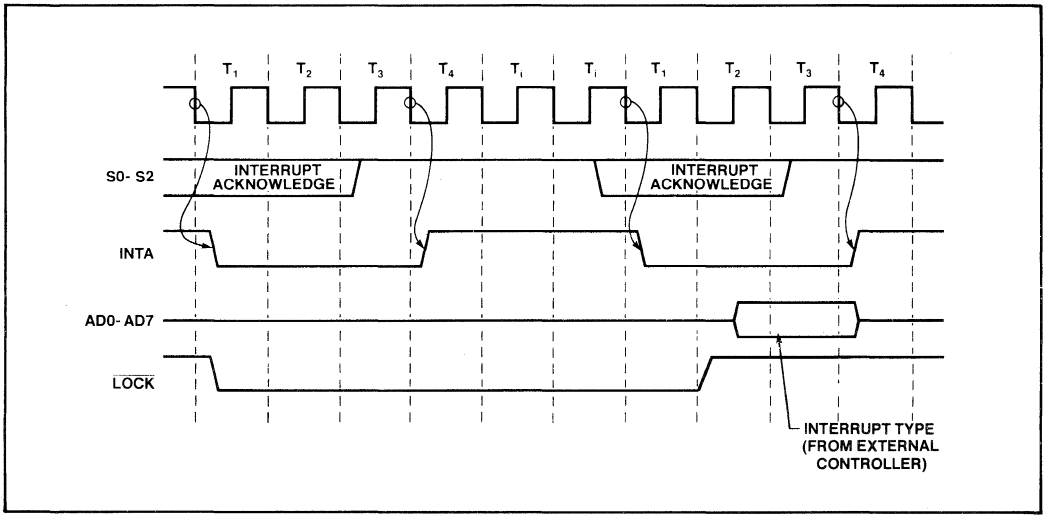


Figure 60. 80186 Cascaded Interrupt Acknowledge Timing

SLAVE SELECT line during the falling edge of the clock at the beginning of T_3 of the second interrupt acknowledge cycle. This input must be stable 20ns before and 10ns after this edge.

These two interrupt acknowledge cycles will be run back to back, and will be LOCKED with the LOCK output active (meaning that DMA requests and HOLD requests will not be honored until both cycles have been run). Note that the two interrupt acknowledge cycles will always be separated by two idle T states, and that wait states will be inserted into the interrupt acknowledge cycle if a ready is not returned by the processor bus interface. The two idle T states are inserted to allow compatibility with the timing requirements of an external 8259A interrupt controller.

Because the interrupt acknowledge cycles must be run in iRMX 86 mode, even for internally generated vectors, and the integrated controller presents an interrupt type rather than a vector address, the interrupt response time here is the same as if an externally vectored interrupt was required, namely 55 CPU clocks.

6.5.3 EXTERNAL VECTORING

External interrupt vectoring occurs whenever the 80186 interrupt controller is placed in cascade mode, special fully nested mode, or iRMX 86 mode (and the integrated controller is not enabled by the external master interrupt controller). In this mode, the 80186 generates two interrupt acknowledge cycles, reading the interrupt type off the lower 8 bits of the address/data bus on the second interrupt acknowledge cycle (see Figure 60). This interrupt response is exactly the same as the 8086, so that the 8259A interrupt controller can be used exactly as it would in an 8086 system. Notice that the two interrupt acknowledge cycles are LOCKED, and that two idle T-states are always inserted between the two interrupt acknowledge bus cycles, and that wait states will be inserted in the interrupt acknowledge cycle if a ready is not returned to the processor. Also notice that the 80186 provides two interrupt acknowledge signals, one for interrupts signaled by the INT0 line, and one for interrupts signaled by the INT1 line (on the INT2/INTA0 and INT3/INTA1 lines, respectively). These two interrupt acknowledge signals are mutually exclusive. Interrupt acknowledge status will be driven on the status lines (S0-S2) when either INT2/INTA0 or INT3/INTA1 signal an interrupt acknowledge.

6.6 Interrupt Controller External Connections

The four interrupt signals can be programmably configured into 3 major options. These are direct interrupt inputs (with the integrated controller providing the interrupt vector), cascaded (with an external interrupt controller providing the interrupt vector), or iRMX 86 mode. In all these modes, any interrupt presented to the external lines must remain set until the interrupt is acknowledged.

6.6.1 DIRECT INPUT MODE

When the Cascade mode bits are cleared, the interrupt input lines are configured as direct interrupt input lines (see Figure 61). In this mode an interrupt source (e.g., an 8272 floppy disk controller) may be directly connected to the interrupt input line. Whenever an interrupt is received on the input line, the integrated controller will do nothing unless the interrupt is enabled, and it is the highest priority pending interrupt. At this time, the interrupt controller will present the interrupt to the CPU and wait for an interrupt acknowledge. When the acknowledge occurs, it will present the interrupt vector address to the CPU. In this mode, the CPU will not run any interrupt acknowledge cycles.

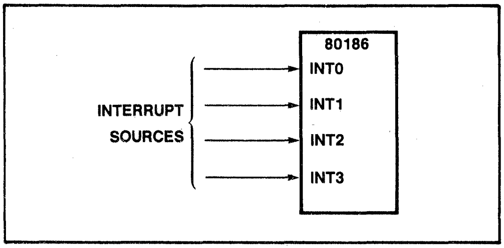


Figure 61. 80186 Non-Cascaded Interrupt Connection

These lines can be individually programmed in either edge or level triggered mode using their respective control registers. In edge triggered mode, a low-to-high transition must occur before the interrupt will be generated to the CPU, while in level triggered mode, only a high level must be present on the input for an interrupt to be generated. In edge trigger mode, the interrupt input must also be low for at least 1 CPU clock cycle to insure recognition. In both modes, the interrupt input must remain active until acknowledged.

6.6.2 CASCADE MODE

When the Cascade mode bit is set and the SFNM bit is cleared, the interrupt input lines are configured in cascade mode. In this mode, the interrupt input line is paired with an interrupt acknowledge line. The INT2/$\overline{\text{INTA0}}$ and INT3/$\overline{\text{INTA1}}$ lines are dual purpose; they can function as direct input lines, or they can function as interrupt acknowledge outputs. INT2/$\overline{\text{INTA0}}$ provides the interrupt acknowledge for an INT0 input, and INT3/$\overline{\text{INTA1}}$ provides the interrupt acknowledge for an INT1 input. Figure 62 shows this connection.

When programmed in this mode, in response to an interrupt request on the INT0 line, the 80186 will provide two interrupt acknowledge pulses. These pulses will be provided on the INT2/$\overline{\text{INTA0}}$ line, and will also be reflected by interrupt acknowledge status being generated on the $\overline{\text{S0}}$-$\overline{\text{S2}}$ status lines. On the second pulse, the interrupt type will be read in. The 80186 externally vectored interrupt response is covered in more detail in section 6.5.

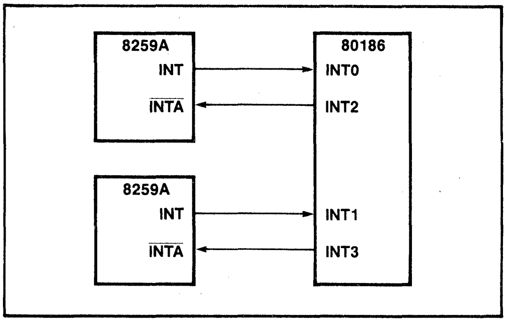


Figure 62. 80186 Cascade and Special Fully Nested Mode Interface

INT0/INT2/$\overline{\text{INTA0}}$ and INT1/INT3/$\overline{\text{INTA1}}$ may be individually programmed into interrupt request/acknowledge pairs, or programmed as direct inputs. This means that INT0/INT2/$\overline{\text{INTA0}}$ may be programmed as an interrupt/acknowledge pair, while INT1 and INT3/$\overline{\text{INTA1}}$ each provide separate internally vectored interrupt inputs.

When an interrupt is received on a cascaded interrupt, the priority mask bits and the in-service bits in the particular interrupt control register will be set into the interrupt controller's mask and priority mask registers. This will prevent the controller from generating an 80186 CPU interrupt request from a lower priority interrupt. Also, since the in-service bit is set, any subsequent interrupt requests on the particular interrupt input line will not cause the integrated interrupt controller to generate an interrupt request to the 80186 CPU. This means that if the external interrupt controller receives a higher priority interrupt request on one of its interrupt request lines and presents it to the 80186 interrupt request line, it will not subsequently be presented to the 80186 CPU by the integrated interrupt controller until the in-service bit for the interrupt line has been cleared.

6.6.3 SPECIAL FULLY NESTED MODE

When both the Cascade mode bit and the SFNM bit are set, the interrupt input lines are configured in Special Fully Nested Mode. The external interface in this mode is exactly as in Cascade Mode. The only difference is in the conditions allowing an interrupt from the external interrupt controller to the integrated interrupt controller to interrupt the 80186 CPU.

When an interrupt is received from a special fully nested mode interrupt line, it will interrupt the 80186 CPU if it is the highest priority interrupt pending regardless of the state of the in-service bit for the interrupt source in the interrupt controller. When an interrupt is acknowledged from a special fully nested mode interrupt line, the priority mask bits and the in-service bits in the particular interrupt control register will be set into the interrupt controller's in-service and priority mask registers. This will prevent the interrupt controller from generating an 80186 CPU interrupt request from a lower priority interrupt. Unlike cascade mode, however, the interrupt controller will not prevent additional interrupt requests generated by the same external interrupt controller from interrupting the 80186 CPU. This means that if the external (cascaded) interrupt controller receives a higher priority interrupt request on one of its interrupt request lines and presents it to the integrated controller's interrupt request line, it may cause an interrupt to be generated to the 80186 CPU, regardless of the state of the in-service bit for the interrupt line.

If the SFNM mode bit is set and the Cascade mode bit is not also set, the controller will provide internal interrupt vectoring. It will also ignore the state of the in-service bit in determining whether to present an interrupt request to the CPU. In other words, it will use the SFNM conditions of interrupt generation with an internally vectored interrupt response, i.e., if the interrupt pending is the highest priority type pending, it will cause a CPU interrupt regardless of the state of the in-service bit for the interrupt.

6.6.4 iRMX™ 86 MODE

When the RMX bit in the peripheral relocation register is set, the interrupt controller is set into iRMX 86 mode. In this mode, all four interrupt controller input lines are used to perform the necessary handshaking with the external master interrupt controller. Figure 63 shows the hardware configuration of the 80186 interrupt lines with an external controller in iRMX 86 mode.

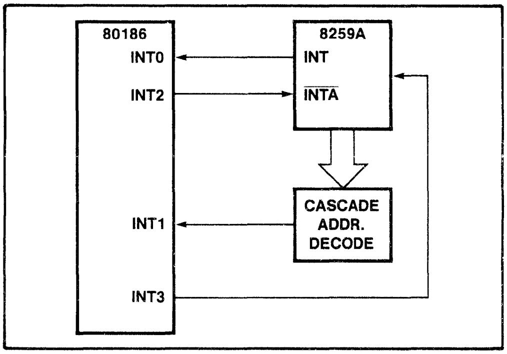


Figure 63. 80186 iRMX86 Mode Interface

Because the integrated interrupt controller is a slave controller, it must be able to generate an interrupt input for an external interrupt controller. It also must be signaled when it has the highest priority pending interrupt to know when to place its interrupt vector on the bus. These two signals are provided by the INT3/Slave Interrupt Output and INT1/Slave Select lines, respectively. The external master interrupt controller must be able to interrupt the 80186 CPU, and needs to know when the interrupt request is acknowledged. The INT0 and INT2/INTA0 lines provide these two functions.

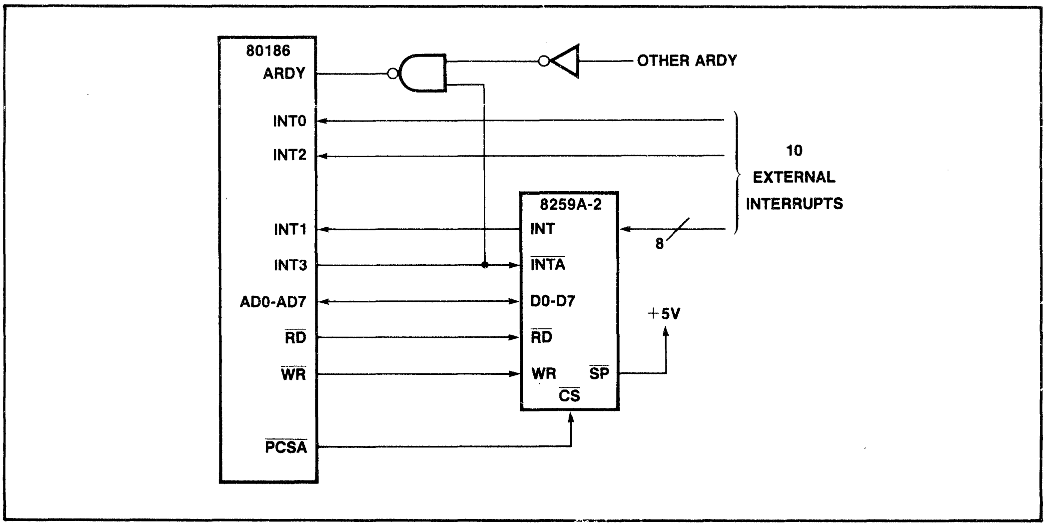


Figure 64. 80186/8259A Interrupt Cascading

6.7 Example 8259A/Cascade Mode Interface

Figure 64 shows the 80186 and 8259A in cascade interrupt mode. The code to initialize the 80186 interrupt controller is given in Appendix E. Notice that an "interrupt ready" signal must be returned to the 80186 to prevent the generation of wait states in response to the interrupt acknowledge cycles. In this configuration the INT0 and INT2 lines are used as direct interrupt input lines. Thus, this configuration provides 10 external interrupt lines: 2 provided by the 80186 interrupt controller itself, and 8 from the external 8259A. Also, the 8259A is configured as a master interrupt controller. It will only receive interrupt acknowledge pulses in response to an interrupt it has generated. It may be cascaded again to up to 8 additional 8259As (each of which would be configured in slave mode).

6.8 Example 80130 iRMX™ 86 Mode Interface

Figure 65 shows the 80186 and 80130 connected in iRMX 86 mode. In this mode, the 80130 interrupt controller is the master interrupt controller of the system. The 80186 generates an interrupt request to the 80130 interrupt controller when one of the 80186 integrated peripherals has created an interrupt condition, and that condition is sufficient to generate an interrupt from the 80186 integrated interrupt controller. Note that the 80130 decodes the interrupt acknowledge status directly from the 80186 status lines; thus, the INT2/$\overline{\text{INTA0}}$ line of the 80186 need not be connected to the 80130. Figure 65 uses this interrupt acknowledge signal to enable the cascade address decoder. The 80130 drives the cascade address on AD8-AD10 during T_1 of the second interrupt acknowledge cycle. This cascade address is latched into the system address latches, and if the proper cascade address is decoded by the 8205 decoder, the 80186 INT1/$\overline{\text{SLAVE SELECT}}$ line will be driven active, enabling the 80186 integrated interrupt controller to place its interrupt vector on the internal bus. The code to configure the 80186 into iRMX 86 mode is presented in appendix E.

6.9 Interrupt Latency

Interrupt latency time is the time from when the 80186 receives the interrupt to the time it begins to respond to the interrupt. This is different from interrupt response

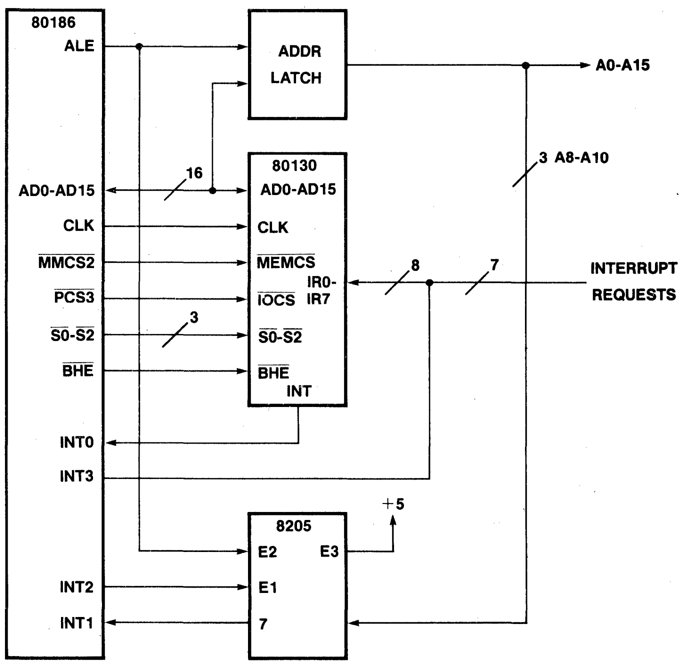


Figure 65. 80186/80130 iRMX86 Mode Interface

time, which is the time from when the processor actually begins processing the interrupt to when it actually executes the first instruction of the interrupt service routine. The factors affecting interrupt latency are the instruction being executed and the state of the interrupt enable flip-flop.

Interrupts will be acknowledged only if the interrupt enable flip-flop in the CPU is set. Thus, interrupt latency will be very long indeed if interrupts are never enabled by the processor!

When interrupts are enabled in the CPU, the interrupt latency is a function of the instructions being executed. Only repeated instructions will be interrupted before being completed, and those only between their respective iterations. This means that the interrupt latency time could be as long as 69 CPU clocks, which is the time it takes the processor to execute an integer divide instruction (with a segment override prefix, see below), the longest single instruction on the 80186.

Other factors can affect interrupt latency. An interrupt will not be accepted between the execution of a prefix (such as segment override prefixes and lock prefixes) and the instruction. In addition, an interrupt will not be accepted between an instruction which modifies any of the segment registers and the instruction immediately following the instruction. This is required to allow the stack to be changed. If the interrupt were accepted, the return address from the interrupt would be placed on a stack which was not valid (the Stack Segment register would have been modified but the Stack Pointer register would not have been). Finally, an interrupt will not be accepted between the execution of the WAIT instruction and the instruction immediately following it if the $\overline{\text{TEST}}$ input is active. If the WAIT sees the $\overline{\text{TEST}}$ input inactive, however, the interrupt will be accepted, and the WAIT will be re-executed after the interrupt return. This is required, since the WAIT is used to prevent execution by the 80186 of an 8087 instruction while the 8087 is busy.

7. CLOCK GENERATOR

The 80186 includes a clock generator which generates the main clock signal for all 80186 integrated components, and all CPU synchronous devices in the 80186 system. This clock generator includes a crystal oscillator, divide by two counter, reset circuitry, and ready generation logic. A block diagram of the clock generator is shown in Figure 66.

7.1 Crystal Oscillator

The 80186 crystal oscillator is a parallel resonant, Pierce oscillator. It was designed to be used as shown in Figure 67. The capacitor values shown are approximate. As the crystal frequency drops, they should be increased, so that at the 4 MHz minimum crystal frequency supported by the 80186 they take on a value of 30pF. The output of this oscillator is not directly available outside the 80186.

The following parameters may be used for choosing a crystal:

Temperature Range:	0 to 70° C
ESR (Equivalent Series Resistance):	30Ω max
C_0 (Shunt Capacitance of Crystal):	7.0 pf max
C_l (Load Capacitance):	20 pf ± 2 pf
Drive Level:	1 mw max

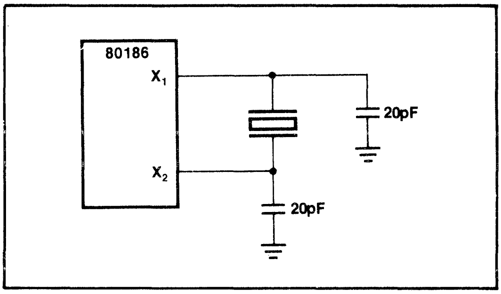


Figure 67. 80186 Crystal Connection

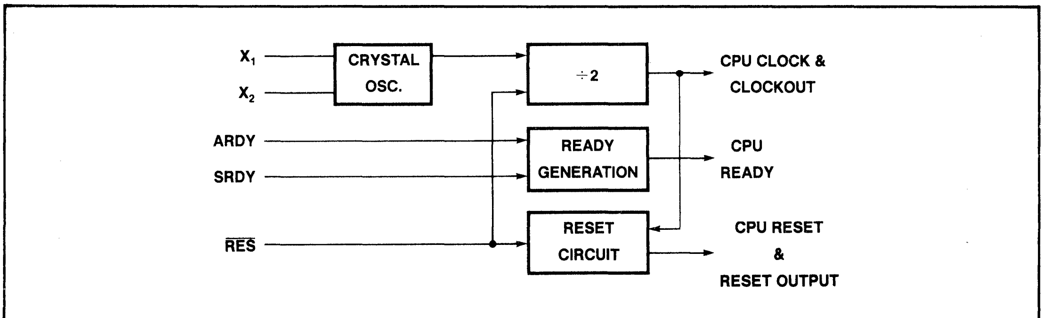


Figure 66. 80186 Clock Generator Block Diagram

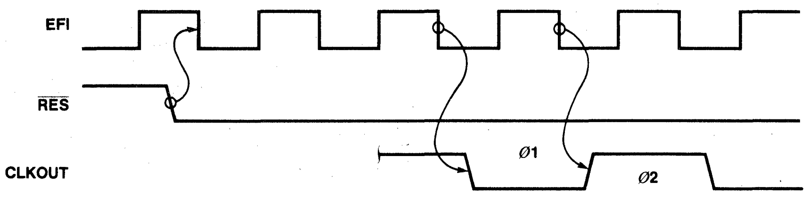


Figure 68. 80186 Clock Generator Reset

7.2 Using an External Oscillator

An external oscillator may be used with the 80186. The external frequency input (EFI) signal is connected directly to the X1 input of the oscillator. X2 should be left open. This oscillator input is used to drive an internal divide-by-two counter to generate the CPU clock signal, so the external frequency input can be of practically any duty cycle, so long as the minimum high and low times for the signal (as stated in the data sheet) are met.

7.3 Clock Generator

The output of the crystal oscillator (or the external frequency input) drives a divide by two circuit which generates a 50% duty cycle clock for the 80186 system. All 80186 timing is referenced to this signal, which is available on the CLKOUT pin of the 80186. This signal will change state on the high-to-low transition of the EFI signal.

7.4 Ready Generation

The clock generator also includes the circuitry required for ready generation. Interfacing to the SRDY and ARDY inputs this provides is covered in section 3.1.6.

7.5 Reset

The 80186 clock generator also provides a synchronized reset signal for the system. This signal is generated from the reset input ($\overline{RES}$) to the 80186. The clock generator synchronizes this signal to the clockout signal.

The reset input signal also resets the divide-by-two counter. A one clock cycle internal clear pulse is generated when the $\overline{RES}$ input signal first goes active. This clear pulse goes active beginning on the first low-to-high transition of the X1 input after $\overline{RES}$ goes active, and goes inactive on the next low-to-high transition of the X1 input. In order to insure that the clear pulse is generated on the next EFI cycle, the $\overline{RES}$ input signal must satisfy a 25ns setup time to the high-to-low EFI input signal (see Figure 68). During this clear, clockout will be high. On the next high-to-low transition of X1, clockout will go low, and will change state on every subsequent high-to-low transition of EFI.

The reset signal presented to the rest of the 80186, and also the signal present on the RESET output pin of the 80186 is synchronized by the high-to-low transition of the clockout signal of the 80186. This signal remains active as long as the $\overline{RES}$ input also remains active. After the $\overline{RES}$ input goes inactive, the 80186 will begin to fetch its first instruction (at memory location FFFF0H) after 6 1/2 CPU clock cycles (i.e., T_1 of the first instruction fetch will occur 6 1/2 clock cycles later). To insure that the RESET output will go inactive on the next CPU clock cycle, the inactive going edge of the $\overline{RES}$ input must satisfy certain hold and setup times to the low-to-high edge of the clockout signal of the 80186 (see Figure 69).

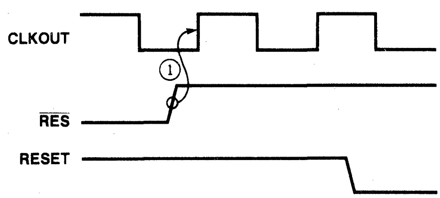


Figure 69. 80186 Coming out of Reset

8. CHIP SELECTS

The 80186 includes a chip select unit which generates hardware chip select signals for memory and I/O accesses generated by the 80186 CPU and DMA units. This unit is programmable such that it can be used to fulfill the chip select requirements (in terms of memory device or bank size and speed) of most small and medium sized 80186 systems.

The chip selects are driven only for internally generated bus cycles. Any cycles generated by an external unit (e.g., an external DMA controller) will not cause the chip selects to go active. Thus, any external bus masters must be responsible for their own chip select generation. Also, during a bus HOLD, the 80186 does not float the chip select lines. Therefore, logic must be included to enable the devices which the external bus master wishes to access (see Figure 70).

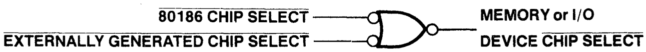


Figure 70. 80186/External Chip Select/Device Chip Select Generation

8.1 Memory Chip Selects

The 80186 provides six discrete chip select lines which are meant to be connected to memory components in an 80186 system. These signals are named UCS, LCS, and MCS0-3 for Upper Memory Chip Select, Lower Memory Chip Select and Midrange Memory Chip Selects 0-3. They are meant (but not limited) to be connected to the three major areas of the 80186 system memory (see Figure 71).

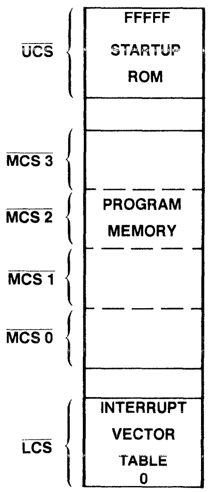


Figure 71. 80186 Memory Areas & Chip Selects

As could be guessed by their names, upper memory, lower memory, and mid-range memory chip selects are designed to address upper, lower, and middle areas of memory in an 80186 system. The upper limit of UCS and the lower limit of LCS are fixed at FFFFFH and 00000H in memory space, respectively. The other limit of these is set by the memory size programmed into the control register for the chip select line. Mid-range memory allows both the base address and the block size of the memory area to be programmed. The only limitation is that the base address must be programmed to be an integer multiple of the total block size. For example, if the block size was 128K bytes (4 32K byte chunks) the base address could be 0 or 20000H, but not 10000H.

The memory chip selects are controlled by 4 registers in the peripheral control block (see Figure 72). These include 1 each for UCS and LCS, the values of which determine the size of the memory blocks addressed by these two lines. The other two registers are used to control the size and base address of the mid-range memory block.

On reset, only UCS is active. It is programmed by reset to be active for the top 1K memory block, to insert 3 wait states to all memory fetches, and to factor external ready for every memory fetch (see section 8.3 for more information on internal ready generation). All other chip select registers assume indeterminate states after reset, but none of the other chip select lines will be active until all necessary registers for a signal have been accessed (not necessarily written, a read to an uninitialized register will enable the chip select function controlled by that register).

8.2 Peripheral Chip Selects

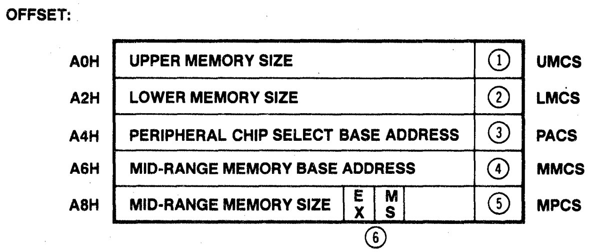
The 80186 provides seven discrete chip select lines which are meant to be connected to peripheral components in an 80186 system. These signals are named PCS0-6. Each of these lines is active for one of seven contiguous 128 byte areas in memory or I/O space above a programmed base address.

The peripheral chip selects are controlled by two registers in the internal peripheral control block (see Figure 72). These registers allow the base address of the peripherals to be set, and allow the peripherals to be mapped into memory or I/O space. Both of these registers must be accessed before any of the peripheral chip selects will become active.

A bit in the MPCS register allows PCS5 and PCS6 to become latched A1 and A2 outputs. When this option is selected, PCS5 and PCS6 will reflect the state of A1 and A2 throughout a bus cycle. These are provided to allow external peripheral register selection in a system in which the addresses are not latched. Upon reset, these lines are driven high. They will only reflect A1 and A2 after both PACS and MPCS have been accessed (and are programmed to provide A1 and A2!).

8.3 Ready Generation

The 80186 includes a ready generation unit. This unit generates an internal ready signal for all accesses to memory or I/O areas to which the chip select circuitry of the 80186 responds.



1. Upper memory ready bits
2. Lower memory ready bits
3. PCS0-PCS3 ready bits
4. Mid-range memory ready bits
5. PCS4-PCS6 ready bits
6. MS: 1 = Peripherals active in memory space
   0 = Peripherals active in I/O space
   EX:1 = 7 PCS lines
   0 = PCS5 = A1, PCS6 = A2

Not all bits of every field are used

Figure 72. 80186 Chip Select Control Registers

For each ready generation area, 0-3 wait states may be inserted by the internal unit. Table 6 shows how the ready control bits should be programmed to provide this. In addition, the ready generation circuit may be programmed to ignore the state of the external ready (i.e., only the internal ready circuit will be used) or to factor the state of the external ready (i.e., a ready will be returned to the processor only after both the internal ready circuit has gone ready and the external ready has gone ready). Some kind of circuit must be included to generate an external ready, however, since upon reset the ready generator is programmed to factor external ready to all accesses to the top 1K byte memory block. If a ready was not returned on one of the external ready lines (ARDY or SRDY) the processor would wait forever to fetch its first instruction.

Table 6. 80186 Wait State Programming

R2	R1	R0	Number of Wait States
0	0	0	0 + external ready
0	0	1	1 + external ready
0	1	0	2 + external ready
0	1	1	3 + external ready
1	0	0	0 (no external ready required)
1	0	1	1 (no external ready required)
1	1	0	2 (no external ready required)
1	1	1	3 (no external ready required)

8.4 Examples of Chip Select Usage

Many examples of the use of the chip select lines are given in the bus interface section of this note (section 3.2). These examples show how simple it is to use the chip select function provided by the 80186. The key point to remember when using the chip select function is that they are only activated during bus cycles generated by the 80186 CPU or DMA units. When another master has the bus, it must generate its own chip select function. In addition, whenever the bus is given by the 80186 to an external master (through the HOLD/ HLDA arrangement) the 80186 does NOT float the chip select lines.

8.5 Overlapping Chip Select Areas

Generally, the chip selects of the 80186 should not be programmed such that any two areas overlap. In addition, none of the programmed chip select areas should overlap any of the locations of the integrated 256-byte control register block. The consequences of doing this are:

> Whenever two chip select lines are programmed to respond to the same area, both will be activated during any access to that area. When this is done, the ready bits for both areas *must* be programmed to the same value. If this is not done, the processor response to an access in this area is indeterminate. This rule also applies to overlapping chip selects with the integrated control block.

> If any of the chip select area overlap the integrated 256-byte control register block, the timing on the

210973-003

chip select line is altered. As always, any values returned on the external bus from this access are ignored.

9. SOFTWARE IN AN 80186 SYSTEM

Since the 80186 is object code compatible with the 8086 and 8088, the software in an 80186 system is very similar to that in an 8086 system. Because of the hardware chip select functions, however, a certain amount of initialization code must be included when using those functions on the 80186.

9.1 System Initialization in an 80186 System

Most programmable components of a computer system must be initialized before they are used. This is also true for the 80186. The 80186 includes circuitry which directly affects the ability of the system to address memory and I/O devices, namely the chip select circuitry. This circuitry must be initialized before the memory areas and peripheral devices addressed by the chip select signals are used.

Upon reset, the UMCS register is programmed to be active for all memory fetches within the top 1K byte of memory space. It is also programmed to insert three wait states to all memory accesses within this space. If the hardware chip selects are used, they must be programmed before the processor leaves this 1K byte area of memory. If a jump to an area for which the chips are not selected occurs, the microcomputer system will cease to operate (since the processor will fetch garbage from the data bus). Appendix F shows a typical initialization sequence for the 80186 chip select unit.

Once the chip selects have been properly initialized, the rest of the 80186 system may be initialized much like an 8086 system. For example, the interrupt vector table might get set up, the interrupt controller initialized, a serial I/O channel initialized, and the main program begun. Note that the integrated peripherals included in the 80186 do not share the same programming model as the standard Intel peripherals used to implement these functions in a typical 8086 system, i.e., different values must be programmed into different registers to achieve the same function using the integrated peripherals. Appendix F shows a typical initialization sequence for an interrupt driven system using the 80186 interrupt controller.

9.2 Initialization for iRMX™ 86 System

Using the iRMX 86 operating system with the 80186 requires an external 8259A and an external 8253/4 or alternatively an external 80130 OSF component. These are required because the operating system is interrupt driven, and expects the interrupt controller and timers to have the register model of these external devices. This model is not the same as is implemented by the 80186. Because of this, the 80186 interrupt controller must be placed in iRMX 86 mode after reset. This initialization can be done at any time after reset before jump to the root task of iRMX 86 System is actually performed. If need be, a small section of code which initializes both the 80186 chip selects and the 80186 interrupt controller can be inserted between the reset vector location and the beginning of iRMX 86 System (see Figure 73). In this case, upon reset, the processor would jump to the 80186 initialization code, and when this has been completed, would jump to the iRMX 86 initialization code (in the root task). It is important that the 80186 hardware be initialized before iRMX 86 operation is begun, since some of the resources addressed by the 80186 system may not be initialized properly by iRMX 86 System if the initialization is done in the reverse manner.

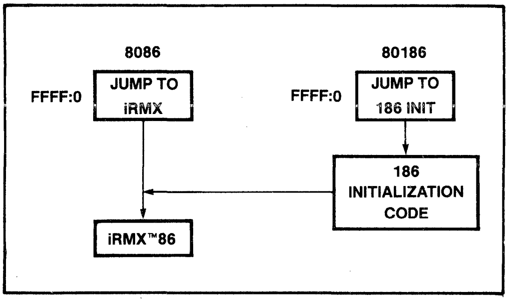


Figure 73. iRMX-86 Initialization with 8086 & 80186

9.3 Instruction Execution Differences Between the 8086 and 80186

There are a few instruction execution differences between the 8086 and the 80186. These differences are:

Undefined Opcodes:

When the opcodes 63H,64H,65H,66H,67H,F1H, FEH XX111XXXB and FFH XX111XXXB are executed, the 80186 will execute an illegal instruction exception, interrupt type 6. The 8086 will ignore the opcode.

0FH opcode:

When the opcode 0FH is encountered, the 8086 will execute a POP CS, while the 80186 will execute an illegal instruction exception, interrupt type 6.

Word Write at Offset FFFFH:

When a word write is performed at offset FFFFH in a segment, the 8086 will write one byte at offset FFFFH, and the other at offset 0, while the 80186 will write one byte at offset

FFFFH, and the other at offset 10000H (one byte beyond the end of the segment). One byte segment underflow will also occur (on the 80186) if a stack PUSH is executed and the Stack Pointer contains the value 1.

Shift/Rotate by Value Greater Then 31:

Before the 80186 performs a shift or rotate by a value (either in the CL register, or by an immediate value) it ANDs the value with 1FH, limiting the number of bits rotated to less than 32. The 8086 does not do this.

LOCK prefix:

The 8086 activates its LOCK signal immediately after executing the LOCK prefix. The 80186 does not activate the LOCK signal until the processor is ready to begin the data cycles associated with the LOCKed instruction.

NOTE: When executing more than one LOCKed instruction, always make sure there are 6 bytes of code between the end of the first LOCKed instruction and the start of the second LOCKed instruction.

Interrupted String Move Instructions:

If an 8086 is interrupted during the execution of a repeated string move instruction, the return value it will push on the stack will point to the last prefix instruction before the string move instruction. If the instruction had more than one prefix (e.g., a segment override prefix in addition to the repeat prefix), it will not be re-executed upon returning from the interrupt. The 80186 will push the value of the first prefix to the repeated instruction, so long as prefixes are not repeated, allowing the string instruction to properly resume.

Conditions causing divide error with an integer divide:

The 8086 will cause a divide error whenever the absolute value of the quotient is greater then 7FFFH (for word operations) or if the absolute value of the quotient is greater than 7FH (for byte operations). The 80186 has expanded the range of negative numbers allowed as a quotient by 1 to include 8000H and 80H. These numbers represent the most negative numbers representable using 2's complement arithmetic (equaling −32768 and −128 in decimal, respectively).

ESC Opcode:

The 80186 may be programmed to cause an interrupt type 7 whenever an ESCape instruction (used for co-processors like the 8087) is executed. The 8086 has no such provision. Before the 80186 performs this trap, it must be programmed to do so.

These differences can be used to determine whether the program is being executed on an 8086 or an 80186. Probably the safest execution difference to use for this purpose is the difference in multiple bit shifts. For example, if a multiple bit shift is programmed where the shift count (stored in the CL register!) is 33, the 8086 will shift the value 33 bits, whereas the 80186 will shift it only a single bit.

In addition to the instruction execution differences noted above, the 80186 includes a number of new instruction types, which simplify assembly language programming of the processor, and enhance the performance of higher level languages running on the processor. These new instructions are covered in depth in the 8086/80186 users manual and in appendix H of this note.

10. CONCLUSIONS

The 80186 is a glittering example of state-of-the art integrated circuit technology applied to make the job of the microprocessor system designer simpler and faster. Because many of the required peripherals and their interfaces have been cast in silicon, and because of the timing and drive latitudes provided by the part, the designer is free to concentrate on other issues of system design. As a result, systems designed around the 80186 allow applications where no other processor has been able to provide the necessary performance at a comparable size or cost.

APPENDIX A
APPENDIX B
APPENDIX C
APPENDIX D
APPENDIX E
APPENDIX F
APPENDIX G
APPENDIX H
APPENDIX I

APPENDIX A: PERIPHERAL CONTROL BLOCK

All the integrated peripherals within the 80186 microprocessor are controlled by sets of registers contained within an integrated peripheral control block. The registers are physically located within the peripheral devices they control, but are addressed as a single block of registers. This set of registers fills 256 contiguous bytes and can be located beginning on any 256 byte boundary of the 80186 memory or I/O space. A map of these registers is shown in Figure A-1.

A.1 Setting the Base Location of the Peripheral Control Block

In addition to the control registers for each of the integrated 80186 peripheral devices, the peripheral control block contains the peripheral control block relocation register. This register allows the peripheral control block to be re-located on any 256 byte boundary within the processor's memory or I/O space. Figure A-2 shows the layout of this register.

This register is located at offset FEH within the peripheral control block. Since it is itself contained within the peripheral control block, any time the location of the peripheral control block is moved, the location of the relocation register will also move.

In addition to the peripheral control block relocation information, the relocation register contains two additional bits. One is used to set the interrupt controller into iRMX86 compatibility mode. The other is used to force the processor to trap whenever an ESCape (coprocessor) instruction is encountered.

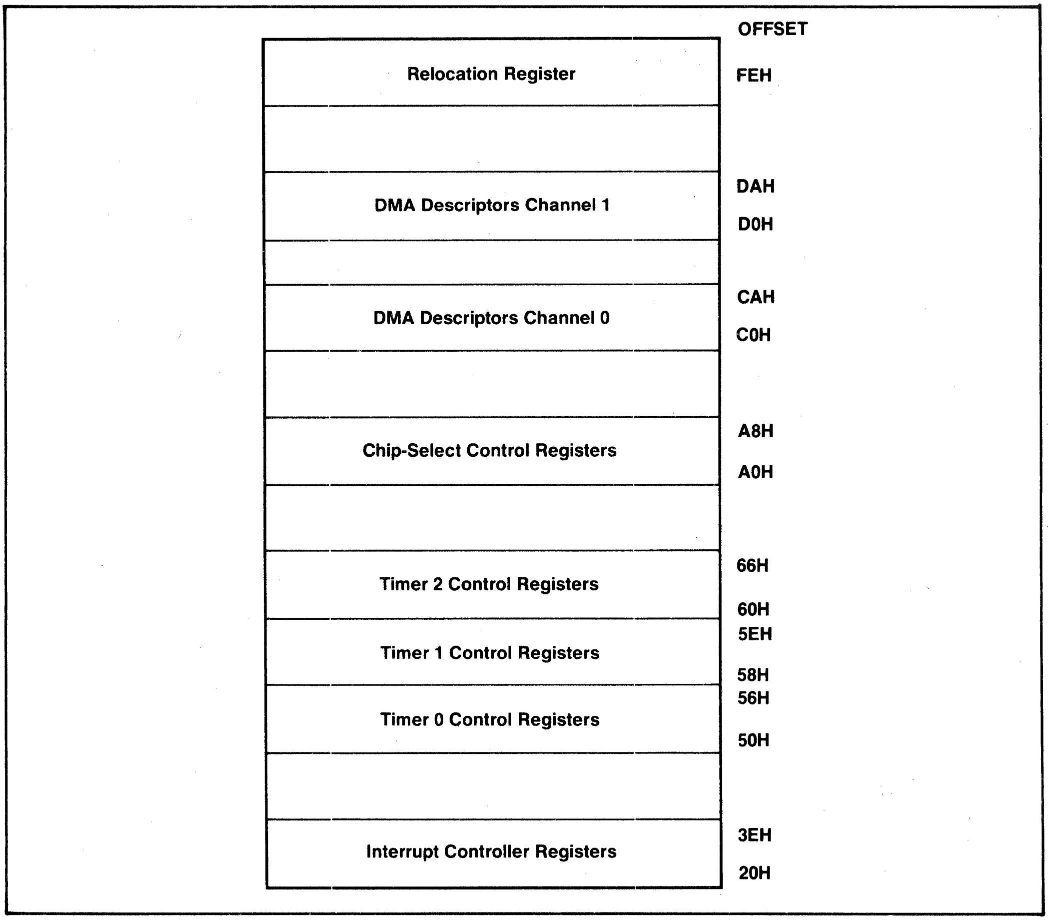


Figure A-1. 80186 Integrated Peripheral Control Block

210973-003

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: FEH	ET	RMX	X	M/IO	Relocation Address Bits R19–R8											

ET = ESC Trap / No ESC Trap (1/0)
M/IO = Register block located in Memory / I/O Space (1/0)
RMX = Master Interrupt Controller mode / iRMX compatible Interrupt Controller mode (0/1)

Figure A-2. 80186 Relocation Register Layout

Because the relocation register is contained within the peripheral control block, upon reset the relocation register is automatically programmed with the value 20FFH. This means that the peripheral control block will be located at the very top (FF00H to FFFFH) of I/O space. Thus, after reset the relocation register will be located at word location FFFEH in I/O space.

If the user wished to locate the peripheral control block starting at memory location 10000H he would program the peripheral control register with the value 1100H. By doing this, he would move all registers within the integrated peripheral control block to memory locations 10000H to 100FFH. Note that since the relocation register is contained within the peripheral control block, it too would move to word location 100FEH in memory space.

Whenever mapping the 188 peripheral control block to another location, the programming of the relocation register should be done with a byte write (i.e. OUT DX,AL). Any access to the control block is done 16 bits at a time. Thus, internally, the relocation register will get written with 16 bits of the AX register while externally, the BIU will run only one 8 bit bus cycle. If a word instruction is used (i.e. OUT DX,AX), the relocation register will be written on the first bus cycle. The BIU will then run a second bus cycle which is unnecessary. The address of the second bus cycle will no longer be within the control block (i.e. the control block was moved on the first cycle), and therefore, will require the generation of an external ready signal to complete the cycle. For this reason we recommend byte operations to the relocation register. Byte instructions may also be used for the other registers in the control block and will eliminate half of the bus cycles required if a word operation had been specified. Byte operations are only valid on even addresses though, and are undefined on odd addresses.

A.2 Peripheral Control Block Registers

Each of the integrated peripherals' control and status registers are located at a fixed location above the programmed base location of the peripheral control block. There are many locations within the peripheral control block which are not assigned to any peripheral. If a write is made to any of these locations, the bus cycle will be run, but the value will not be stored in any internal location. This means that if a subsequent read is made to the same location, the value written will not be read back.

The processor will run an external bus cycle for any memory or I/O cycle which accesses a location within the integrated control block. This means that the address, data, and control information will be driven on the 80186 external pins just as if a "normal" bus cycle had been run. Any information returned by an external device will be ignored, however, even if the access was to a location which does not correspond to any of the integrated peripheral control registers. The above is also true for the 80188, except that the word access made to the integrated registers will be performed in a single bus cycle internally, while externally, the BIU runs two bus cycles.

The processor internally generates a ready signal whenever any of the integrated peripherals are accessed; thus any external ready signals are ignored whenever an access is made to any location within the integrated peripheral register control block. This ready will also be returned if an access is made to a location within the 256 byte area of the periperal control block which does not correspond to any integrated peripheral control register. The processor will insert 0 wait states to any access within the integrated peripheral control block except for accesses to the timer registers. ANY access to the timer control and counting registers will incur 1 wait state. This wait state is required to properly multiplex processor and counter element accesses to the timer control registers.

All accesses made to the integrated peripheral control block will be WORD accesses. Any write to the integrated registers will modify all 16 bits of the register, whether the opcode specified a byte write or a word write. A byte read from an even location should cause no problems, but the data returned when a byte read is performed from an odd address within the peripheral control block is undefined. This is true both for the 80186 AND the 80188. As stated above, even though the 80188 has an external 8 bit data bus, internally it is still a 16 bit machine. Thus, the word accesses performed to the integrated registers by the 80188 will each occur in a single bus cycle internally while externally the BIU runs two bus cycles.

APPENDIX B: 80186 SYNCHRONIZATION INFORMATION

Many input signals to the 80186 are asynchronous, that is, a specified set up or hold time is not required to insure proper functioning of the device. Associated with each of these inputs is a synchronizer which samples this external asynchronous signal, and synchronizes it to the internal 80186 clock.

B.1 Why Synchronizers Are Required

Every data latch requires a certain set up and hold time in order to operate properly. At a certain window within the specified set up and hold time, the part will actually try to latch the data. If the input makes a transition within this window, the output will not attain a stable state within the given output delay time. The size of this sampling window is typically much smaller than the actual window specified by the data sheet, however part to part variation could move this window around within the specified window in the data sheet.

Even if the input to a data latch makes a transition while a data latch is attempting to latch this input, the output of the latch will attain a stable state after a certain amount of time, typically much longer than the normal strobe to output delay time. Figure B-1 shows a normal input to output strobed transition and one in which the input signal makes a transition during the latch's sample window. In order to synchronize an asynchronous signal, all one needs to do is to sample the signal into one data latch, wait a certain amount of time, then latch it into a second data latch. Since the time between the strobe into the first data latch and the strobe into the second data latch allows the first data latch to attain a steady state (or to resolve the asynchronous signal), the second data latch will be presented with an input signal which satisfies any set up and hold time requirements it may have.

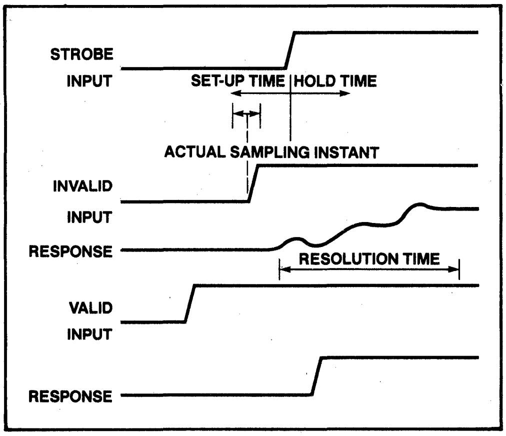


Figure B-1. Valid & Invalid Latch Input Transitions & Responses

Thus, the output of this second latch is a synchronous signal with respect to its strobe input.

A synchronization failure can occur if the synchronizer fails to resolve the asynchronous transition within the time between the two latch's strobe signals. The rate of failure is determined by the actual size of the sampling window of the data latch, and by the amount of time between the strobe signals of the two latches. Obviously, as the sampling window gets smaller, the number of times an asynchronous transition will occur during the sampling window will drop. In addition, however, a smaller sampling window is also indicative of a faster resolution time for an input transition which manages to fall within the sampling window.

B.2 80186 Synchronizers

The 80186 contains synchronizers on the $\overline{RES}$, $\overline{TEST}$, TmrIn0-1, DRQ0-1, NMI, INT0-3, ARDY, and HOLD input lines. Each of these synchronizers use the two stage synchronization technique described above (with some minor modifications for the ARDY line, see section 3.1.6). The sampling window of the latches is designed to be in the tens of pico-seconds, and should allow operation of the synchronizers with a mean time between failures of over 30 years assuming continuous operation.

APPENDIX C: 80186 EXAMPLE DMA INTERFACE CODE

```
$mod186
name                        assembly_example_80186_DMA_support
;
;  This file contains an example procedure which initializes the 80186 DMA
;       controller to perform the DMA transfers between the 80186 system and the
;       8272 Floppy Disk Controller (FDC). It assumes that the 80186
;       peripheral control block has not been moved from its reset location.
;
arg1                    equu    word ptr [BP + 4]
arg2                    equ     word ptr [BP + 6]
arg3                    equ     word ptr [BP + 8]
DMA_FROM_LOWER          equ     0FFC0h              ; DMA register locations
DMA_FROM_UPPER          equ     0FFC2h
DMA_TO_LOWER            equ     0FFC4h
DMA_TO_UPPER            equ     0FFC6h
DMA_COUNT               equ     0FFC8h
DMA_CONTROL             equ     0FFCAh
DMA_TO_DISK_CONTROL     equ     01486h              ; destination synchronization
                                                    ; source to memory, incremented
                                                    ; destination to I/O
                                                    ; no terminal count
                                                    ; byte transfers

DMA_FROM_DISK_CONTROLequ        0A046h              ; source synchronization
                                                    ; source to I/O
                                                    ; destination to memory, incr
                                                    ; no terminal count
                                                    ; byte transfers
FDC_DMA                 equ     6B8h                ; FDC DMA address
FDC_DATA                equ     688h                ; FDC data register
FDC_STATUS              equ     680h                ; FDC status register

cgroup                  group   code
code                    segment                     public 'code'
                        public  set_dma_
                        assume  cs:cgroup
;
;  set_dma (offset,to) programs the DMA channel to point one side to the
;       disk DMA address, and the other to memory pointed to by ds:offset. If
;       'to' = 0 then will be a transfer from disk to memory; if
;       'to' = 1 then will be a transfer from memory to disk. The parameters to
;       the routine are passed on the stack.
;
set_dma_                proc    near
                        enter   0,0                 ; set stack addressability
                        push    AX                  ; save registers used
                        push    BX
                        push    DX
                        test    arg2,1              ; check to see direction of
                                                    ; transfer
                        jz      from_disk
;  performing a transfer from memory to the disk controller
;
                        mov     AX,DS               ; get the segment value
                        rol     AX,4                ; gen the upper 4 bits of the
                                                    ; physical address in the lower 4
                                                    ; bits of the register
```

```
                mov     BX,AX                   ; save the result...
                mov     DX,DMA_FROM_UPPER       ; prgm the upper 4 bits of the
                out     DX,AX                   ; DMA source register
                and     AX,0FFF0h               ; form the lower 16 bits of the
                                                ; physical address
                add     AX,arg1                 ; add the offset
                mov     DX,DMA_FROM_LOWER       ; prgm the lower 16 bits of the
                out     DX,AX                   ; DMA source register
                jnc     no_carry_from           ; check for carry out of addition
                inc     BX                      ; if carry out, then need to adj
                mov     AX,BX                   ; the upper 4 bits of the pointer
                mov     DX,DMA_FROM_UPPER
                out     DX,AX
no_carry_from:
                mov     AX,FDC_DMA              ; prgm the low 16 bits of the DMA
                mov     DX,DMA_TO_LOWER         ; destination register
                out     DX,AX
                xor     AX,AX                   ; zero the up 4 bits of the DMA
                mov     DX,DMA_TO_UPPER         ; destination register
                out     DX,AX
                mov     AX,DMA_TO_DISK_CONTROL; prgm the DMA ctl reg
                mov     DX,DMA_CONTROL          ; note: DMA may begin immediatly
                out     DX,AX                   ; after this word is output
                pop     DX
                pop     BX
                pop     AX
                leave
                ret
from_disk:
;
; performing a transfer from the disk to memory
;
                mov     AX,DS
                rol     AX,4
                mov     DX,DMA_TO_UPPER
                out     DX,AX
                mov     BX,AX
                and     AX,0FFF0h
                add     AX,arg1
                mov     DX,DMA_TO_LOWER
                out     DX,AX
                jnc     no_carry_to
                inc     BX
                mov     AX,BX
                mov     DX,DMA_TO_UPPER
                out     DX,AX
no_carry_to:
                mov     AX,FDC_DMA

                mov     DX,DMA_FROM_LOWER
                out     DX,AX
                xor     AX,AX
                mov     DX,DMA_FROM_UPPER
                out     DX,AX
                mov     AX,DMA_FROM_DISK_CONTROL
                mov     DX,DMA_CONTROL
```

```
                out     DX,AX
                pop     DX
                pop     BX
                pop     AX
                leave
                ret
set_dma_        endp

code            ends
                end
```

APPENDIX D: 80186 EXAMPLE TIMER INTERFACE CODE

```
$mod186
name                                example_80186_timer_code
;
;   this file contains example 80186 timer routines. The first routine
;           sets up the timer and interrupt controller to cause the timer
;           to generate an interrupt every 10 milliseconds, and to service
;           interrupt to implement a real time clock. Timer 2 is used in
;           this example because no input or output signals are required.
;           The code example assumes that the peripheral control block has
;           not been moved from its reset location (FF00-FFFF in I/O space).
;
arg1                equ       word ptr [BP + 4]
arg2                equ       word ptr [BP + 6]
arg3                equ       word ptr [BP + 8]
timer_2int          equ       19                    ; timer 2 has vector type 19
timer_2control      equ       0FF66h
timer_2max_ctl      equ       0FF62h
timer_int_ctl       equ       0FF32h                ; interrupt controller regs
eoi_register        equ       0FF22h
interrupt_stat      equ       0FF30h

data                segment                         public 'data'
                    public    hour_,minute_,second_,msec_
msec_               db        ?
hour_               db        ?
minute_             db        ?
second_             db        ?
data                ends

cgroup              group     code
dgroup              group     data

code                segment                         public 'code'
                    public    set_time_
                    assume    cs:code,ds:dgroup
;
;   set_time(hour,minute,second) sets the time variables, initializes the
;           80186 timer2 to provide interrupts every 10 milliseconds, and
;           programs the interrupt vector for timer 2
;
set_time_           proc      near
                    enter     0,0                   ; set stack addressability
                    push      AX                    ; save registers used
                    push      DX
                    push      SI
                    push      DS

                    xor       AX,AX                 ; set the interrupt vector
                                                    ; the timers have unique
                                                    ; interrupt
                                                    ; vectors even though they share
                                                    ; the same control register
                    mov       DS,AX

                    mov       SI,4 * timer2_int
```

```
                        mov     DS:[SI],offset timer_2_interrupt_routine
                        inc     SI
                        inc     SI
                        mov     DS:[SI],CS
                        pop     DS

                        mov     AX,arg1                     ; set the time values
                        mov     hour_,AL
                        mov     AX,arg2
                        mov     minute_,AL
                        mov     AX,arg3
                        mov     second_,AL
                        mov     msec_,0

                        mov     DX,timer2_max_ctl           ; set the max count value
                        mov     AX,20000                    ; 10 ms / 500 ns (timer 2 counts
                                                            ; at 1/4 the CPU clock rate)
                        out     DX,AX
                        mov     DX,timer2_control           ; set the control word
                        mov     AX,1110000000000001b        ; enable counting
                                                            ; generate interrupts on TC
                                                            ; continuous counting
                        out     DX,AX

                        mov     DX,timer_int_ctl            ; set up the interrupt controller
                        mov     AX,0000b                    ; unmask interrupts
                                                            ; highest priority interrupt
                        out     DX,AX
                        sti                                 ; enable processor interrupts

                        pop     SI
                        pop     DX
                        pop     AX
                        leave
                        ret
set_time_               endp

timer2_interrupt_routine proc   far
                        push    AX
                        push    DX

                        cmp     msec_,99                    ; see if one second has passed
                        jae     bump_second                 ; if above or equal...
                        inc     msec_
                        jmp     reset_int_ctl
bump_second:
                        mov     msec_,0                     ; reset millisecond
                        cmp     second_,59                  ; see if one minute has passed
                        jae     bump_minute
                        inc     second_
                        jmp     reset_int_ctl
bump_minute:
                        mov     second_,0
                        cmp     minute_,59                  ; see if one hour has passed
                        jae     bump_hour
                        inc     minute_
                        jmp     reset_int_ctl
```

```
                pop     AX
                pop     DX
                ret
set_rmx_        endp
code            ends
                end
```

```
bump_hour:
                            mov     minute_,0
                            cmp     hour_,12                ;  see if 12 hours have passed
                            jae     reset_hour
                            inc     hour_
                            jmp     reset_int_ctl
reset_hour:
                            mov     hour_,1
reset_int_ctl:

                            mov     DX,eoi_register
                            mov     AX,8000h                ;  non-specific end of interrupt
                            out     DX,AX

                            pop     DX
                            pop     AX
                            iret
timer2_interrupt_routine    endp
code                        ends
                            end
$mod186
name                        example_80186_baud_code
;
;  this file contains example 80186 timer routines. The second routine
;          sets up the timer as a baud rate generator. In this mode,
;          Timer 1 is used to continually output pulses with a period of
;          6.5 usec for use with a serial controller at 9600 baud
;          programmed in divide by 16 mode (the actual period required
;          for 9600 baud is 6.51 usec). This assumes that the 80186 is
;          running at 8 MHz. The code example also assumes that the
;          peripheral control block has not been moved from its reset
;          location (FF00-FFFF in I/O space).
;
timer1_control              equ     0FF5Eh
timer1_max_cnt              equ     0FF5Ah

code                        segment                         public 'code'
                            assume  cs:code
;
;  set_baud() initializes the 80186 timer1 as a baud rate generator for
;          a serial port running at 9600 baud
;
set_baud_                   proc    near
                            push    AX                      ;  save registers used
                            push    DX

                            mov     DX,timer1_max_cnt       ;  set the max count value
                            mov     AX,13                   ;  500ns * 13 = 6.5 usec
                            out     DX,AX
                            mov     DX,timer1_control       ;  set the control word
                            mov     AX,1100000000000001b    ;  enable counting
                                                            ;  no interrupt on TC
                                                            ;  continuous counting
                                                            ;  single max count register
                            out     DX,AX

                            pop     DX
                            pop     AX
```

```
                ret
set_baud_       endp
code            ends
                end
$mod186
name            example_80186_count_code
;
; this file contains example 80186 timer routines. The third routine
;     sets up the timer as an external event counter. In this mode,
;     Timer 1 is used to count transitions on its input pin. After
;     the timer has been set up by the routine, the number of
;     events counted can be directly read from the timer count
;     register at location FF58H in I/O space. The timer will
;     count a maximum of 65535 timer events before wrapping
;     around to zero. This code example also assumes that the
;     peripheral control block has not been moved from its reset
;     location (FF00-FFFF in I/O space).
;
timer1_control  equu    0FF5Eh
timer1_max_cnt  equ     0FF5Ah
timer1_cnt_reg  equ     0FF58H

code            segment                         public 'code'
                assume  cs:code
;
; set_count() initializes the 80186 timer1 as an event counter
;
set_count_      proc    near
                push    AX                      ; save registers used
                push    DX

                mov     DX,timer1_max_cnt       ; set the max count value
                mov     AX,0                    ; allows the timer to count
                                                ; all the way to FFFFH
                out     DX,AX
                mov     DX,timer1_control       ; set the control word
                mov     AX,1100000000000101b    ; enable counting
                                                ; no interrupt on TC
                                                ; continuous counting
                                                ; single max count register
                                                ; external clocking
                out     DX,AX

                xor     AX,AX                   ; zero AX
                mov     DX,timer1_cnt_reg       ; and zero the count in the timer
                out     DX,AX                   ; count register

                pop     DX
                pop     AX
                ret

set_count_      endp
code            ends
                end
```

APPENDIX E: 80186 EXAMPLE INTERRUPT CONTROLLER INTERFACE CODE

```
$mod186
name                    example_80186_interrupt_code
;
; This routine configures the 80186 interrupt controller to provide
;       two cascaded interrupt inputs (through an external 8259A
;       interrupt controller on pins INT0/INT2) and two direct
;       interrupt inputs (on pins INT1 and INT3). The default priority
;       levels are used. Because of this, the priority level programmed
;       into the control register is set the 111, the level all
;       interrupts are programmed to at reset.
;
int0_control            equ     0FF38H
int_mask                equ     0FF28H
;
code                    segment                 public 'code'
                        assume  CS:code
set_int_                proc    near
                        push    DX
                        push    AX

                        mov     AX,0100111B     ; cascade mode
                                                ; interrupt unmasked
                        mov     DX,int0_control
                        out     DX,AX

                        mov     AX,01001101B    ; now unmask the other external
                                                ; interrupts
                        mov     DX,int_mask
                        out     DX,AX
                        pop     AX
                        pop     DX
                        ret
set_int_                endp
code                    ends
                        end
$mod186
name                    example_80186_interrupt_code
;
; This routine configures the 80186 interrupt controller into iRMX 86
;       mode. This code does not initialize any of the 80186
;       integrated peripheral control registers, nor does it initialize
;       the external 8259A or 80130 interrupt controller.
;
relocation_reg          equ     0FFFEH
;
code                    segment                 public 'code'
                        assume  CS:code
set_rmx_                proc    near
                        push    DX
                        push    AX

                        mov     DX,relocation_reg
                        in      AX,DX                   ; read old contents of register
                        or      AX,0100000000000000B    ; set the RMX mode bit
                        out     DX,AX
```

APPENDIX F: 80186/8086 EXAMPLE SYSTEM INITIALIZATION CODE

```
name            example_80186_system_init
;
; This file contains a system initialization routine for the 80186
;       or the 8086. The code determines whether it is running on
;       an 80186 or an 8086, and if it is running on an 80186, it
;       initializes the integrated chip select registers.
;
restart         segment  at                     0FFFFh
;
; This is the processor reset address at 0FFFF0H
;
                org      0
                jmp      far ptr initialize
restart         ends
;
                extrn    monitor:far
init_hw         segment  at                     0FFF0h
                assume   CS:init_hw
;
; This segment initializes the chip selects. It must be located in the
;       top 1K to insure that the ROM remains selected in the 80186
;       system until the proper size of the select area can be programmed.
;
UMCS_reg        equ      0FFA0H                 ; chip select register locations
LMCS_reg        equ      0FFA2H
PACS_reg        equ      0FFA4H
MPCS_reg        equ      0FFA8H
UMCS_value      equ      0F038H                 ; 64K, no wait states
LMCS_value      equ      07F8H                  ; 32K, no wait states
PACS_value      equ      007EH                  ; peripheral base at 400H, 2 ws
MPCS_value      equ      81B8H                  ; PCS5 and 6 supplies,
                                                ; peripherals in I/O space

initialize      proc     far
                mov      AX,2                   ; determine if this is an
                mov      CL,33                  ; 8086 or an 80186 (checks
                shr      AX,CL                  ; to see if the multiple bit
                test     AX,1                   ; shift value was ANDed)
                jz       not_80186

                mov      DX,UMCS_reg            ; program the UMCS register
                mov      AX,UMCS_value
                out      DX,AX

                mov      DX,LMCS_reg            ; program the LMCS register
                mov      AX,LMCS_value
                out      DX,AX

                mov      DX,PACS_reg            ; set up the peripheral chip
                                                ; selects (note the mid-range
                                                ; memory chip selects are not
                                                ; needed in this system, and
                                                ; are thus not initialized

                mov      AX,PACS_value
                out      DX,AX
```

```
                        mov     DX,MPCS_reg
                        mov     AX,MPCS_value
                        out     DX,AX
;
;  Now that the chip selects are all set up, the main program of the
;          computer may be executed.
;
not_80186:
                        jmp     far ptr monitor
initialize              endp
init_hw                 ends
                        end
```

APPENDIX G: 80186 WAIT STATE PERFORMANCE

Because the 80186 contains seperate bus interface and execution units, the actual performance of the processor will not degrade at a constant rate as wait states are added to the memory cycle time from the processor. The actual rate of performace degradation will depend on the type and mix of instructions actually encountered in the user's program.

Shown below are two 80186 assembly language programs, and the actual execution time for the two programs as wait states are added to the memory system of the processor. These programs show the two extremes to which wait states will or will not effect system performance as wait states are introduced.

Program 1 is very memory intensive. It performs many memory reads and writes using the more extensive memory addressing modes of the processor (which also take a greater number of bytes in the opcode for the instruction). As a result, the execution unit must constantly wait for the bus interface unit to fetch and perform the memory cycles to allow it to continue. Thus, the execution time of this type of routine will grow quickly as wait states are added, since the execution time is almost totally limited to the speed at which the processor can run bus cycles.

Note also that this program execution times calculated by merely summing up the number of clock cycles given in the data sheet will typically be less than the actual number of clock cycles actually required to run the program. This is because the numbers quoted in the data sheet assume that the opcode bytes have been prefetched and reside in the 80186 prefetch queue for immediate access by the execution unit. If the execution unit cannot access the opcode bytes immediatly upon request, dead clock cycles will be inserted in which the execution unit will remain idle, thus increasing the number of clock cycles required to complete execution of the program.

On the other hand, program 2 is more CPU intensive. It performs many integer multiplies, during which time the bus interface unit can fill up the instruction prefetch queue in parallel with the execution unit performing the multiply. In this program, the bus interface unit can perform bus operations faster than the execution unit actually requires them to be run. In this case, the performance degradation is much less as wait states are added to the memory interface. The execution time of this program is closer to the number of clock cycles calculated by adding the number of cycles per instruction because the execution unit does not have to wait for the bus interface unit to place an opcode byte in the prefetch queue as often. Thus, fewer clock cycles are wasted by the execution unit laying idle for want of instructions. Table G-1 lists the execution times measured for these two programs as wait states were introduced with the 80186 running at 8 MHz.

Table G-1

# of Wait States	Program 1 Exec Time (μsec)	Program 1 Perf Degr	Program 2 Exec Time (μsec)	Program 2 Perf Degr
0	505		294	
1	595	18%	311	6%
2	669	12%	337	8%
3	752	12%	347	3%

```
$mod186
name                    example_wait_state_performance
;
; This file contains two programs which demonstrate the 80186 performance
;       degradation as wait states are inserted. Program 1 performs a
;       transformation between two types of characters sets, then copies
;       the transformed characters back to the original buffer (which is 64
;       bytes long. Program 2 performs the same type of transformation, however
;       instead of performing a table lookup, it multiplies each number in the
;       original 32 word buffer by a constant (3, note the use of the integer
;       immediate multiply instruction). Program "nothing" is used to measure
;       the call and return times from the driver program only.
;
cgroup                  group   code
dgroup                  group   data
data                    segment                 public 'data'
```

```
t_table         db        256 dup (?)
t_string        db        64 dup (?)
m_array         dw        32 dup (?)
data            ends

code            segment                                 public 'code'
                assume    CS:cgroup,DS:dgroup
                public    bench_1,bench_2,nothing_,wait_state_,set_timer_
bench_1         proc      near
                push      SI                            ; save registers used
                push      CX
                push      BX
                push      AX

                mov       CX,64                         ; translate 64 bytes
                mov       SI,0
                mov       BH,0

loop_back:
                mov       BL,t_string[SI]               ; get the byte
                mov       AL,t_table[BX]                ; translate byte
                mov       t_string[SI],AL               ; and store it
                inc       SI                            ; increment index
                loop      loop_back                     ; do the next byte

                pop       AX
                pop       BX
                pop       CX
                pop       SI
                ret
bench_1         endp

bench_2         proc      near
                push      AX                            ; save registers used
                push      SI
                push      CX

                mov       CX,32                         ; multiply 32 numbers
                mov       SI,offset m_array

loop_back_2:
                imul      AX,word ptr [SI],3            ; immediate multiply
                mov       word ptr [SI],AX
                inc       SI
                inc       SI
                loop      loop_back_2

                pop       CX
                pop       SI
                pop       AX
                ret
bench_2_        endp
```

```
nothing_            proc    near
                    ret
nothing_            endp
;
; wait_state(n) sets the 80186 LMCS register to the number of wait states
;         (0 to 3) indicated by the parameter n (which is passed on the stack).
;         No other bits of the LMCS register are modified.
;
wait_state_         proc    near
                    enter   0,0                     ; set up stack frame
                    push    AX                      ; save registers used
                    push    BX
                    push    DX

                    mov     BX,word ptr [BP + 4]    ; get argument
                    mov     DX,0FFA2h               ; get current LMCS register
contents
                    in      AX,DX

                    and     AX,0FFFCh               ; and off existing ready bits
                    and     BX,3                    ; insure ws count is good
                    or      AX,BX                   ; adjust the ready bits
                    out     DX,AX                   ; and write to LMCS

                    pop     DX
                    pop     BX
                    pop     AX
                    leave                           ; tear down stack frame
                    ret
wait_state_         endp
;
; set_timer() initializes the 80186 timers to count microseconds. Timer 2
;       is set up as a prescaler to timer 0, the microsecond count can be read
;         directly out of the timer 0 count register at location FF50H in I/O
;         space.
;
set_timer_          proc    near
                    push    AX
                    push    DX

                    mov     DX,0ff66h               ; stop timer 2
                    mov     AX,4000h
                    out     DX,AX

                    mov     DX,0ff50h               ; clear timer 0 count
                    mov     AX,0
                    out     DX,AX

                    mov     DX,0ff52h               ; timer 0 counts up to 65535
                    mov     AX,0
                    out     DX,AX
```

```
                mov     DX,0ff56h           ; enable timer 0
                mov     AX,0c009h
                out     DX,AX

                mov     DX,0ff60h           ; clear timer 2 count
                mov     AX,0
                out     DX,AX

                mov     DX,0ff62h           ; set maximum count of timer 2
                mov     AX,2
                out     DX,AX

                mov     DX,0ff66h           ; re-enable timer 2
                mov     AX,0c001h
                out     DX,AX

                pop     DX
                pop     AX
                ret
set_timer_      endp
code            ends
                end
```

APPENDIX H: 80186 NEW INSTRUCTIONS

The 80186 performs many additional instructions to those of the 8086. These instructions appear shaded in the instruction set summary at the back of the 80186 data sheet. This appendix explains the operation of these new instructions. In order to use these new instructions with the 8086/186 assembler, the "$mod186" switch must be given to the assembler. This can be done by placing the line: "$mod186" at the beginning of the assembly language file.

- **PUSH immediate**

This instruction allows immediate data to be pushed onto the processor stack. The data can be either an immediate byte or an immediate word. If the data is a byte, it will be sign extended to a word before it is pushed onto the stack (since all stack operations are word operations).

- **PUSHA, POPA**

These instructions allow all of the general purpose 80186 registers to be saved on the stack, or restored from the stack. The registers saved by this instruction (in the order they are pushed onto the stack) are AX, CX, DX, BX, SP, BP, SI, and DI. The SP value pushed onto the stack is the value of the register before the first PUSH (AX) is performed; the value popped for the SP register is ignored.

This instruction does not save any of the segment registers (CS, DS, SS, ES), the instruction pointer (IP), the flag register, or any of the integrated peripheral registers.

- **IMUL by an immediate value**

This instruction allows a value to be multiplied by an immediate value. The result of this operation is 16 bits long. One operand for this instruction is obtained using one of the 80186 addressing modes (meaning it can be in a register or in memory). The immediate value can be either a byte or a word, but will be sign extended if it is a byte. The 16-bit result of the multiplication can be placed in any of the 80186 general purpose or pointer registers.

This instruction requires three operands: the register in which the result is to be placed, the immediate value, and the second operand. Again, this second operand can be any of the 80186 general purpose registers or a specified memory location.

- **shifts/rotates by an immediate value**

The 80186 can perform multiple bit shifts or rotates where the number of bits to be shifted is specified by an immediate value. This is different from the 8086, where only a single bit shift can be performed, or a multiple shift can be performed where the number of bits to be shifted is specified in the CL register.

All of the shift/rotate instructions of the 80186 allow the number of bits shifted to be specified by an immediate value. Like all multiple bit shift operations performed by the 80186, the number of bits shifted is the number of bits specified modulus 32 (i.e. the maximum number of bits shifted by the 80186 multiple bit shifts is 31).

These instructions require two operands: the operand to be shifted (which may be a register or a memory location specified by any of the 80186 addressing modes) and the number of bits to be shifted.

- **block input/output**

The 80186 adds two new input/output instructions: INS and OUTS. These instructions perform block input or output operations. They operate similarly to the string move instructions of the processor.

The INS instruction performs block input from an I/O port to memory. The I/O address is specified by the DX register; the memory location is pointed to by the DI register. After the operation is performed, the DI register is adjusted by 1 (if a byte input is specified) or by 2 (if a word input is specified). The adjustment is either an increment or a decrement, as determined by the Direction bit in the flag register of the processor. The ES segment register is used for memory addressing, and cannot be overridden. When preceeded by a REPeat prefix, this instruction allows blocks of data to be moved from an I/O address to a block of memory Note that the I/O address in the DX register is not modified by this operation.

The OUTS instruction performs block output from memory to an I/O port. The I/O address is specified by the DX register; the memory location is pointed to by the SI register. After the operation is performed, the SI register is adjusted by 1 (if a byte output is specified) or by 2 (if a word output is specified). The adjustment is either an increment or a decrement, as determined by the Direction bit in the flag register of the processor. The DS segment register is used for memory addressing, but can be overridden by using a segment override prefix. When preceeded by a REPeat prefix, this instruction allows blocks of data to be moved from a block of memory to an I/O address. Again note that the I/O address in the DX register is not modified by this operation.

Like the string move instruction, these two instructions require two operands to specify whether word or byte operations are to take place. Additionally, this determination can be supplied by the mnemonic itself by adding a "B" or "W" to the basic mnemonic, for example:

```
INSB        ; perform byte input
REP OUTSW   ; perform word block output
```

- **BOUND**

The 80186 supplies a BOUND instruction to facilitate bound checking of arrays. In this instruction, the calculated index into the array is placed in one of the general purpose registers of the 80186. Located in two adjacent word memory locations are the lower and upper bounds for the array index. The BOUND instruction compares the register contents to the memory locations, and if the value in the register is not between the values in the memory locations, an interrupt type 5 is generated. The comparisons performed are SIGNED comparisons. A register value equal to either the upper bound or the lower bound will not cause an interrupt.

This instruction requires two arguments: the register in which the calculated array index is placed, and the word memory location which contains the lower bound of the array (which can be specified by any of the 80186 memory addressing modes). The memory location containing the upper bound of the array must follow immediatly the memory location containing the lower bound of the array.

- **ENTER and LEAVE**

The 80186 contains two instructions which are used to build and tear down stack frames of higher level, block structured languages. The instruction used to build these stack frames is the ENTER instruction. The algorithm for this instruction is:

```
PUSH BP                  /* save the previous    frame
                         pointer */
if level = 0 then
  BP := SP;
else     temp1 := SP; /* save current frame    pointer
                         */
    temp2 := level - 1;
    do while temp2 > 0 /* copy down previous    level
                          frame */
      BP := BP - 2;       /* pointers */
      PUSH [BP];
    BP := temp1;
    PUSH BP;              /* put current level    frame
                          pointer */

/* in the save area */
SP := SP - disp;          /* create space on the    stack
                          for */

/* local variables */
```

Figure H-1 shows the layout of the stack before and after this operation.

This instruction requires two operands: the first value (disp) specifies the number of bytes the local variables of this routine require. This is an unsigned value and can be as large as 65535. The second value (level) is an unsigned value which specifies the level of the procedure. It can be as great as 255.

The 80186 includes the LEAVE instruction to tear down stack frames built up by the ENTER instruction. As can be seen from the layout of the stack left by the ENTER instruction, this involves only moving the contents of the BP register to the SP register, and popping the old BP value from the stack.

Neither the ENTER nor the LEAVE instructions save any of the 80186 general purpose registers. If they must be saved, this must be done in addition to the ENTER and the LEAVE. In addition, the LEAVE instruction does not perform a return from a subroutine. If this is desired, the LEAVE instruction must be explicitly followed by the RET instruction.

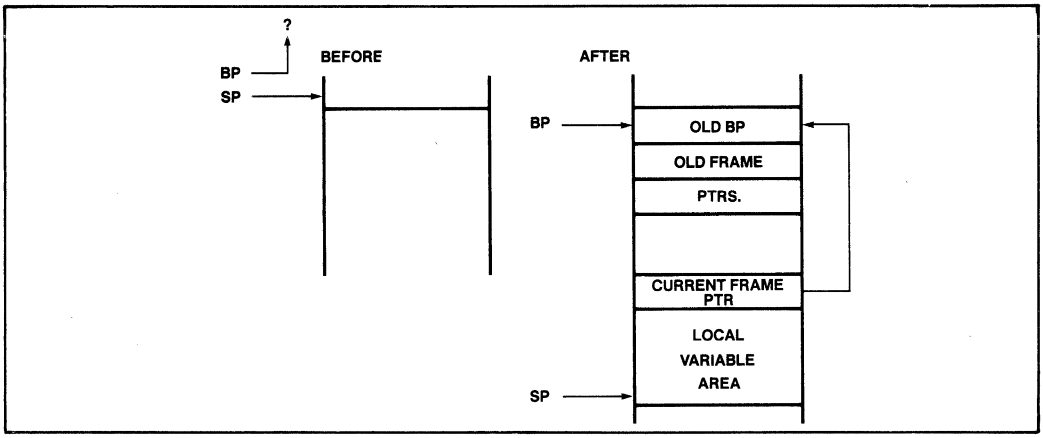


Figure H-1. ENTER Instruction Stack Frame

APPENDIX I: 80186/80188 DIFFERENCES

The 80188 is exactly like the 80186, except it has an 8 bit external bus. It shares the same execution unit, timers, peripheral control block, interrupt controller, chip select, and DMA logic. The differences between the two caused by the narrower data bus are:

- The 80188 has a 4 byte prefetch queue, rather than the 6 byte prefetch queue present on the 80186. The reason for this is since the 80188 fetches opcodes one byte at a time, the number of bus cycles required to fill the smaller queue of the 80188 is actually greater than the number of bus cycles required to fill the queue of the 80186. As a result, a smaller queue is required to prevent an inordinate number of bus cycles being wasted by prefetching opcodes to be discarded during a jump.
- AD8-AD15 on the 80186 are transformed to A8-A15 on the 80188. Valid address information is present on these lines throughout the bus cycle of the 80188. Valid address information is not guaranteed on these lines during idle T states.
- $\overline{\text{BHE}}$/S7 is always defined HIGH by the 80188, since the upper half of the data bus is non-existant.
- The DMA controller of the 80188 only performs byte transfers. The $\overline{\text{B}}$/W bit in the DMA control word is ignored.
- Execution times for many memory access instructions are increased because the memory access must be funnelled through a narrower data bus. The 80188 also will be more bus limited than the 80186 (that is, the execution unit will be required to wait for the opcode information to be fetched more often) because the data bus is narrower. The execution time within the processor, however, has not changed between the 80186 and the 80188.

Another important point is that the 80188 internally is a 16-bit machine. This means that any access to the integrated peripheral registers of the 80188 will be done in 16-bit chunks, NOT in 8-bit chunks. All internal peripheral registers are still 16-bits wide, and only a single read or write is required to access the registers. When a word access is made to the internal registers, the BIU will run two bus cycles externally.

Access to the control block may also be done with byte operations. Internally the full 16-bits of the AX register will be written, while externally, only one bus cycle will be executed.

80286 Microprocessors

4

intel®

iAPX 286/10
High Performance Microprocessor with Memory Management and Protection

(80286-10, 80286-8, 80286-6)

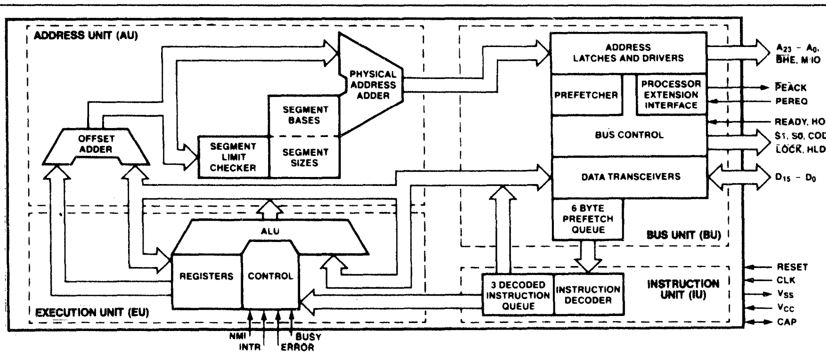
- **High Performance Processor (Up to six times iAPX 86)**
- **Large Address Space:**
  - **— 16 Megabytes Physical**
  - **— 1 Gigabyte Virtual per Task**
- **Integrated Memory Management, Four-Level Memory Protection and Support for Virtual Memory and Operating Systems**
- **Two iAPX 86 Upward Compatible Operating Modes:**
  - **— iAPX 86 Real Address Mode**
  - **— Protected Virtual Address Mode**
- **Optional Processor Extension:**
  - **— 80287 High Performance 80-bit Numeric Data Processor**
- **Range of clock rates**
  - **— 10 MHz for 80286-10**
  - **— 8 MHz for 80826-8**
  - **— 6 MHz for 80286-6**
- **Complete System Development Support:**
  - **— Development Software: Assembler, PL/M, Pascal, FORTRAN, and System Utilities**
  - **— In-Circuit-Emulator (ICE™-286)**
- **High Bandwidth Bus Interface (10 Megabyte/Sec)**
- **Available in EXPRESS:**
  - **— Standard Temperature Range**
- **Available in 68 Pin Ceramic LCC (Leadless Chip Carrier) and PGA (Pin Grid Array) Packages**

(See Packaging Spec., Order #231369)

The iAPX 286/10 (80286 part number) is an advanced, high-performance microprocessor with specially optimized capabilities for multiple user and multi-tasking systems. The 80286 has built-in memory protection that supports operating system and task isolation as well as program and data privacy within tasks. A 10 MHz iAPX 286/10 provides five times or more throughput than the standard 5 MHz iAPX 86/10. The 80286 includes memory management capabilities that map 2^{30} (one gigabyte) of virtual address space per task into 2^{24} bytes (16 megabytes) of physical memory.

The iAPX 286 is upward compatible with iAPX 86 and 88 software. Using iAPX 86 real address mode, the 80286 is object code compatible with existing iAPX 86, 88 software. In protected virtual address mode, the 80286 is source code compatible with iAPX 86, 88 software and may require upgrading to use virtual addresses supported by the 80286's integrated memory management and protection mechanism. Both modes operate at full 80286 performance and execute a superset of the iAPX 86 and 88 instructions.

The 80286 provides special operations to support the efficient implementation and execution of operating systems. For example, one instruction can end execution of one task, save its state, switch to a new task, load its state, and start execution of the new task. The 80286 also supports virtual memory systems by providing a segment-not-present exception and restartable instructions.



210253-1

Figure 1. 80286 Internal Block Diagram

Order Number: 210253-009

Component Pad Views—As viewed from underside of component when mounted on the board.

P.C. Board Views—As viewed from the component side of the P.C. board.

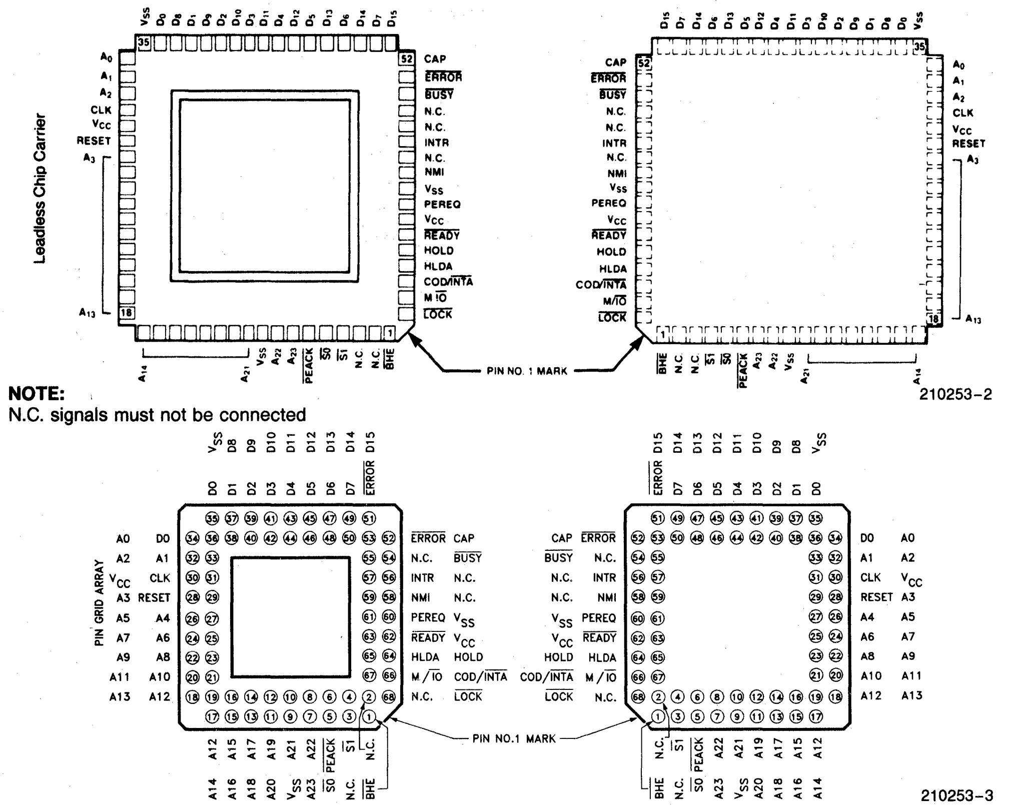


FIGURE 2. 80286 Pin Configuration

Table 1. Pin Description

The following pin function descriptions are for the 80286 microprocessor:

Symbol	Type	Name and Function
CLK	I	**SYSTEM CLOCK** provides the fundamental timing for iAPX 286 systems. It is divided by two inside the 80286 to generate the processor clock. The internal divide-by-two circuitry can be synchronized to an external clock generator by a LOW to HIGH transition on the RESET input.
$D_{15}-D_0$	I/O	**DATA BUS** inputs data during memory, I/O, and interrupt acknowledge read cycles; outputs data during memory and I/O write cycles. The data bus is active HIGH and floats to 3-state OFF during bus hold acknowledge.
$A_{23}-A_0$	O	**ADDRESS BUS** outputs physical memory and I/O port addresses. A0 is LOW when data is to be transferred on pins D_{7-0}. $A_{23}-A_{16}$ are LOW during I/O transfers. The address bus is active HIGH and floats to 3-state OFF during bus hold acknowledge.
$\overline{BHE}$	O	**BUS HIGH ENABLE** indicates transfer or data on the upper byte of the data bus. D_{15-8}. Eight-bit oriented devices assigned to the upper byte of the data bus would normally use $\overline{BHE}$ to condition chip select functions. $\overline{BHE}$ is active LOW and floats to 3-state OFF during bus hold acknowledge.

$\overline{BHE}$ and A0 Encodings

$\overline{BHE}$ Value	A0 Value	Function
0	0	Word transfer
0	1	Byte transfer on upper half of data bus ($D_{15}-D_8$)
1	0	Byte transfer on lower half of data bus (D_{7-0})
1	1	Will never occur

Table I. Pin Description (Continued)

Symbol	Type	Name and Function
$\overline{S1}$, $\overline{S0}$	O	**BUS CYCLE STATUS** indicates initiation of a bus cycle and, along with M/$\overline{IO}$ and COD/$\overline{INTA}$, defines the type of bus cycle. The bus is in a T_S state whenever one or both are LOW, $\overline{S1}$ and $\overline{S0}$ are active LOW and float to 3-state OFF during bus hold acknowledge.

80286 Bus Cycle Status Definition

COD/$\overline{INTA}$	M/$\overline{IO}$	$\overline{S1}$	$\overline{S0}$	Bus Cycle Initiated
0 (LOW)	0	0	0	Interrupt acknowledge
0	0	0	1	Will not occur
0	0	1	0	Will not occur
0	0	1	1	None; not a status cycle
0	1	0	0	IF A1 = 1 then halt; else shutdown
0	1	0	1	Memory data read
0	1	1	0	Memory data write
0	1	1	1	None; not a status cycle
1 (HIGH)	0	0	0	Will not occur
1	0	0	1	I/O read
1	0	1	0	I/O write
1	0	1	1	None; not a status cycle
1	1	0	0	Will not occur
1	1	0	1	Memory instruction read
1	1	1	0	Will not occur
1	1	1	1	None; not a status cycle

Symbol	Type	Name and Function
M/$\overline{IO}$	O	**MEMORY I/O SELECT** distinguishes memory access from I/O access. If HIGH during T_S, a memory cycle or a halt/shutdown cycle is in progress. If LOW, an I/O cycle or an interrupt acknowledge cycle is in progress. M/$\overline{IO}$ floats to 3-state OFF during bus hold acknowledge.
COD/$\overline{INTA}$	O	**CODE/INTERRUPT ACKNOWLEDGE** distinguishes instruction fetch cycles from memory data read cycles. Also distinguishes interrupt acknowledge cycles from I/O cycles. COD/$\overline{INTA}$ floats to 3-state OFF during bus hold acknowledge. Its timing is the same as M/$\overline{IO}$.
$\overline{LOCK}$	O	**BUS LOCK** indicates that other system bus masters are not to gain control of the system bus for the current and the following bus cycle. The $\overline{LOCK}$ signal may be activated explicitly by the "LOCK" instruction prefix or automatically by 80286 hardware during memory XCHG instructions, interrupt acknowledge, or descriptor table access. $\overline{LOCK}$ is active LOW and floats to 3-state OFF during bus hold acknowledge.
$\overline{READY}$	I	**BUS READY** terminates a bus cycle. Bus cycles are extended without limit until terminated by $\overline{READY}$ LOW. $\overline{READY}$ is an active LOW synchronous input requiring setup and hold times relative to the system clock be met for correct operation. $\overline{READY}$ is ignored during bus hold acknowledge.
HOLD HLDA	I O	**BUS HOLD REQUEST AND HOLD ACKNOWLEDGE** control ownership of the 80286 local bus. The HOLD input allows another local bus master to request control of the local bus. When control is granted, the 80286 will float its bus drivers to 3-state OFF and then activate HLDA, thus entering the bus hold acknowledge condition. The local bus will remain granted to the requesting master until HOLD becomes inactive which results in the 80286 deactivating HLDA and regaining control of the local bus. This terminates the bus hold acknowledge condition. HOLD may be asynchronous to the system clock. These signals are active HIGH.
INTR	I	**INTERRUPT REQUEST** requests the 80286 to suspend its current program execution and service a pending external request. Interrupt requests are masked whenever the interrupt enable bit in the flag word is cleared. When the 80286 responds to an interrupt request, it performs two interrupt acknowledge bus cycles to read an 8-bit interrupt vector that identifies the source of the interrupt. To assure program interruption, INTR must remain active until the first interrupt acknowledge cycle is completed. INTR is sampled at the beginning of each processor cycle and must be active HIGH at least two processor cycles before the current instruction ends in order to interrupt before the next instruction. INTR is level sensitive, active HIGH, and may be asynchronous to the system clock.
NMI	I	**NON-MASKABLE INTERRUPT REQUEST** interrupts the 80286 with an internally supplied vector value of 2. No interrupt acknowledge cycles are performed. The interrupt enable bit in the 80286 flag word does not affect this input. The NMI input is active HIGH, may be asynchronous to the system clock, and is edge triggered after internal synchronization. For proper recognition, the input must have been previously LOW for at least four system clock cycles and remain HIGH for at least four system clock cycles.

Table 1. Pin Description (Continued)

<table>
<tr><th>Symbol</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>PEREQ
$\overline{\text{PEACK}}$</td><td>I
O</td><td>PROCESSOR EXTENSION OPERAND REQUEST AND ACKNOWLEDGE extend the memory management and protection capabilities of the 80286 to processor extensions. The PEREQ input requests the 80286 to perform a data operand transfer for a processor extension. The $\overline{\text{PEACK}}$ output signals the processor extension when the requested operand is being transferred. PEREQ is active HIGH and floats to 3-state OFF during bus hold acknowledge. $\overline{\text{PEACK}}$ may be asynchronous to the system clock. $\overline{\text{PEACK}}$ is active LOW.</td></tr>
<tr><td>$\overline{\text{BUSY}}$
$\overline{\text{ERROR}}$</td><td>I
I</td><td>PROCESSOR EXTENSION BUSY AND ERROR indicate the operating condition of a processor extension to the 80286. An active $\overline{\text{BUSY}}$ input stops 80286 program execution on WAIT and some ESC instructions until $\overline{\text{BUSY}}$ becomes inactive (HIGH). The 80286 may be interrupted while waiting for $\overline{\text{BUSY}}$ to become inactive. An active $\overline{\text{ERROR}}$ input causes the 80286 to perform a processor extension interrupt when executing WAIT or some ESC instructions. These inputs are active LOW and may be asynchronous to the system clock.</td></tr>
<tr><td>RESET</td><td>I</td><td>SYSTEM RESET clears the internal logic of the 80286 and is active HIGH. The 80286 may be reinitialized at any time with a LOW to HIGH transition on RESET which remains active for more than 16 system clock cycles. During RESET active, the output pins of the 80286 enter the state shown below:
<table>
<tr><th colspan="2">80286 Pin State During Reset</th></tr>
<tr><th>Pin Value</th><th>Pin Names</th></tr>
<tr><td>1 (HIGH)
0 (LOW)
3-state OFF</td><td>$\overline{\text{S0}}$, $\overline{\text{S1}}$, $\overline{\text{PEACK}}$, A23–A0, $\overline{\text{BHE}}$, $\overline{\text{LOCK}}$
M/$\overline{\text{IO}}$, COD/$\overline{\text{INTA}}$, HLDA (Note 1)
D_{15}–D_0</td></tr>
</table>
Operation of the 80286 begins after a HIGH to LOW transition on RESET. The HIGH to LOW transition of RESET must be synchronous to the system clock. Approximately 50 system clock cycles are required by the 80286 for internal initializations before the first bus cycle to fetch code from the power-on execution address is performed.
A LOW to HIGH transition of RESET synchronous to the system clock will end a processor cycle at the second HIGH to LOW transition of the system clock. The LOW to HIGH transition of RESET may be asynchronous to the system clock; however, in this case it cannot be predetermined which phase of the processor clock will occur during the next system clock period. Synchronous LOW to HIGH transitions of RESET are required only for systems where the processor clock must be phase synchronous to another clock.</td></tr>
<tr><td>V_{SS}</td><td>I</td><td>SYSTEM GROUND: 0 Volts.</td></tr>
<tr><td>V_{CC}</td><td>I</td><td>SYSTEM POWER: +5 Volt Power Supply.</td></tr>
<tr><td>CAP</td><td>I</td><td>SUBSTRATE FILTER CAPACITOR: a 0.047 μF ± 20% 12V capacitor must be connected between this pin and ground. This capacitor filters the output of the internal substrate bias generator. A maximum DC leakage current of 1 μA is allowed through the capacitor.
For correct operation of the 80286, the substrate bias generator must charge this capacitor to its operating voltage. The capacitor chargeup time is 5 milliseconds (max.) after V_{CC} and CLK reach their specified AC and DC parameters. RESET may be applied to prevent spurious activity by the CPU during this time. After this time, the 80286 processor clock can be synchronized to another clock by pulsing RESET LOW synchronous to the system clock.</td></tr>
</table>

NOTE:
1. HLDA is only Low if HOLD is inactive (Low).

FUNCTIONAL DESCRIPTION

Introduction

The 80286 is an advanced, high-performance microprocessor with specially optimized capabilities for multiple user and multi-tasking systems. Depending on the application, the 80286's performance is up to six times faster than the standard 5 MHz 8086's, while providing complete upward software compatibility with Intel's iAPX 86, 88, and 186 family of CPU's.

The 80286 operates in two modes: iAPX 86 real address mode and protected virtual address mode. Both modes execute a superset of the iAPX 86 and 88 instruction set.

In iAPX 86 real address mode programs use real addresses with up to one megabyte of address space. Programs use virtual addresses in protected virtual address mode, also called protected mode. In protected mode, the 80286 CPU automatically maps 1 gigabyte of virtual addresses per task into a 16 megabyte real address space. This mode also provides memory protection to isolate the operating system and ensure privacy of each tasks' programs and data. Both modes provide the same base instruction set, registers, and addressing modes.

The following Functional Description describes first, the base 80286 architecture common to both modes, second, iAPX 86 real address mode, and third, protected mode.

iAPX 286/10 BASE ARCHITECTURE

The iAPX 86, 88, 186, and 286 CPU family all contain the same basic set of registers, instructions, and addressing modes. The 80286 processor is upward compatible with the 8086, 8088, and 80186 CPU's.

Register Set

The 80286 base architecture has fifteen registers as shown in Figure 3. These registers are grouped into the following four categories:

General Registers: Eight 16-bit general purpose registers used to contain arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used either in their entirety as 16-bit words or split into pairs of separate 8-bit registers.

Segment Registers: Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

Base and Index Registers: Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode determines the specific registers used for operand address calculations.

Status and Control Registers: The 3 16-bit special purpose registers in figure 3A record or control certain aspects of the 80286 processor state including the Instruction Pointer, which contains the offset address of the next sequential instruction to be executed.

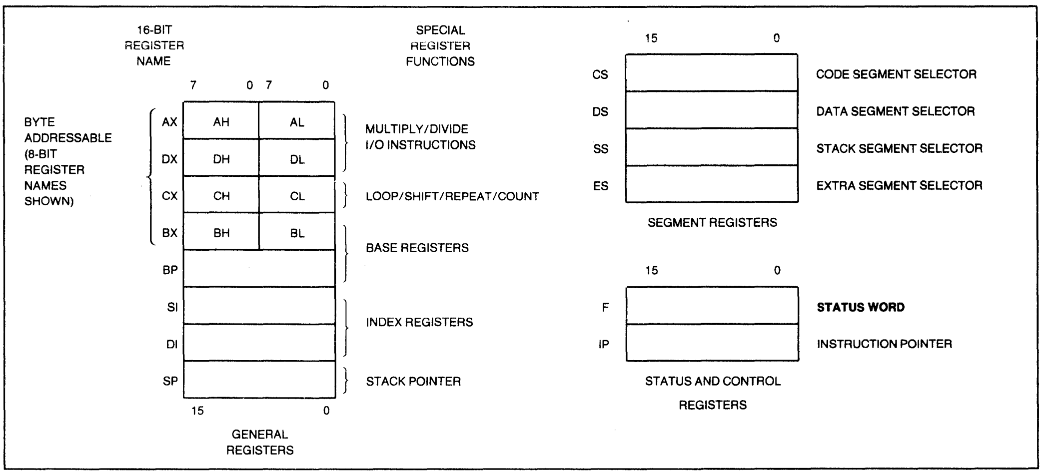


Figure 3. Register Set

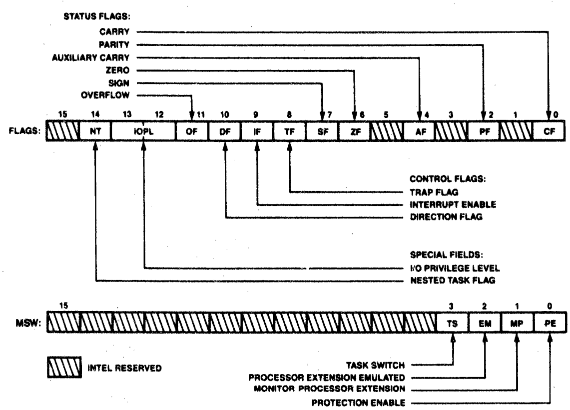


Figure 3a. Status and Control Register Bit Functions

Flags Word Description

The Flags word (Flags) records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80286 within a given operating mode (bits 8 and 9). Flags is a 16-bit register. The function of the flag bits is given in Table 2.

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high level instructions, and processor control. These categories are summarized in Figure 4.

An 80286 instruction can reference zero, one, or two operands; where an operand resides in a register, in the instruction itself, or in memory. Zero-operand instructions (e.g. NOP and HLT) are usually one byte long. One-operand instructions (e.g. INC and DEC) are usually two bytes long but some are encoded in only one byte. One-operand instructions may reference a register or memory location. Two-operand instructions permit the following six types of instruction operations:

—Register to Register
—Memory to Register
—Immediate to Register
—Memory to Memory
—Register to Memory
—Immediate to Memory

Table 2. Flags Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag—Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
11	OF	Overflow Flag—Set if result is a too-large positive number or a too-small negative number (excluding sign-bit) to fit in destination operand; cleared otherwise
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index registers when set. Clearing DF causes auto increment.

Two-operand instructions (e.g. MOV and ADD) are usually three to six bytes long. Memory to memory operations are provided by a special class of string instructions requiring one to three bytes. For detailed instruction formats and encodings refer to the instruction set summary at the end of this document.

For detailed operation and usage of each instruction, see Appendix of iAPX 286 Programmer's Reference Manual (Order No. 210498)

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack

Figure 4a. Data Transfer Instructions

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero

Figure 4c. String Instructions

ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiple byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

Figure 4b. Arithmetic Instructions

LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word

Figure 4d. Shift/Rotate Logical Instructions

CONDITIONAL TRANSFERS		UNCONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal	CALL	Call procedure
JAE/JNB	Jump if above or equal/not below	RET	Return from procedure
JB/JNAE	Jump if below/not above nor equal	JMP	Jump
JBE/JNA	Jump if below or equal/not above		
JC	Jump if carry	ITERATION CONTROLS	
JE/JZ	Jump if equal/zero	LOOP	Loop
JG/JNLE	Jump if greater/not less nor equal		
JGE/JNL	Jump if greater or equal/not less	LOOPE/LOOPZ	Loop if equal/zero
JL/JNGE	Jump if less/not greater nor equal	LOOPNE/LOOPNZ	Loop if not equal/not zero
JLE/JNG	Jump if less or equal/not greater	JCXZ	Jump if register CX = 0
JNC	Jump if not carry		
JNE/JNZ	Jump if not equal/not zero	INTERRUPTS	
JNO	Jump if not overflow	INT	Interrupt
JNP/JPO	Jump if not parity/parity odd		
JNS	Jump if not sign	INTO	Interrupt if overflow
JO	Jump if overflow	IRET	Interrupt return
JP/JPE	Jump if parity/parity even		
JS	Jump if sign		

Figure 4e. Program Transfer Instructions

FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{BUSY}}$ not active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
EXECUTION ENVIRONMENT CONTROL	
LMSW	Load machine status word
SMSW	Store machine status word

Figure 4f. Processor Control Instructions

ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4g. High Level Instructions

Memory Organization

Memory is organized as sets of variable length segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two component address (a pointer) that consists of a 16-bit segment selector, and a 16-bit offset. The segment selector indicates the desired segment in memory. The offset component indicates the desired byte address within the segment.

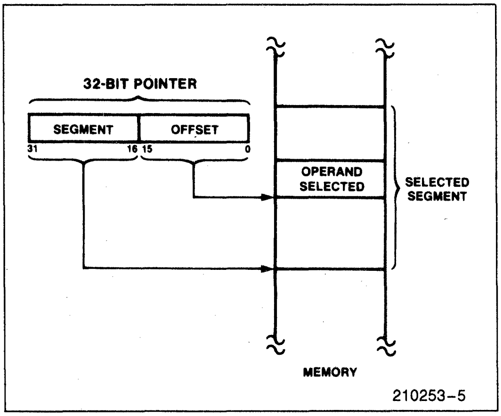


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Automatic with instruction prefetch
Stack	Stack (SS)	All stack pushes and pops. Any memory reference which uses BP as a base register.
Local Data	Data (DS)	All data references except when relative to stack or string destination
External (Global) Data	Extra (ES)	Alternate data segment and destination of string operation

All instructions that address operands in memory must specify the segment and the offset. For speed and compact instruction encoding, segment selectors are usually stored in the high speed segment registers. An instruction need specify only the desired segment register and an offset in order to address a memory operand.

Most instructions need not explicitly specify which segment register is used. The correct segment register is automatically chosen according to the rules of Table 3. These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs. To access operands not residing in one of the four immediately available segments, a full 32-bit pointer or a new segment selector must be loaded.

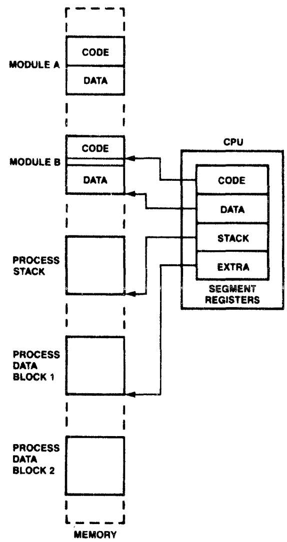


Figure 6. Segmented Memory Helps Structure Software

Addressing Modes

The 80286 provides a total of eight addressing modes for instructions to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

Register Operand Mode: The operand is located in one of the 8 or 16-bit general registers.

Immediate Operand Mode: The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: segment selector and offset. The segment selector is supplied by a segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset is calculated by summing any combination of the following three address elements:

the **displacement** (an 8 or 16-bit immediate value contained in the instruction)

the **base** (contents of either the BX or BP base registers)

the **index** (contents of either the SI or DI index registers)

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

Direct Mode: The operand's offset is contained in the instruction as an 8 or 16-bit displacement element.

Register Indirect Mode: The operand's offset is in one of the registers SI, DI, BX, or BP.

Based Mode: The operand's offset is the sum of an 8 or 16-bit displacement and the contents of a base register (BX or BP).

Indexed Mode: The operand's offset is the sum of an 8 or 16-bit displacement and the contents of an index register (SI or DI).

Based Indexed Mode: The operand's offset is the sum of the contents of a base register and an index register.

Based Indexed Mode with Displacement: The operand's offset is the sum of a base register's contents, an index register's contents, and an 8 or 16-bit displacement.

Data Types

The 80286 directly supports the following data types:

Integer: A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32 and 64-bit integers are supported using the iAPX 286/20 Numeric Data Processor.

Ordinal: An unsigned binary numeric value contained in an 8-bit byte or 16-bit word.

Pointer: A 32-bit quantity, composed of a segment selector component and an offset component. Each component is a 16-bit word.

String: A contiguous sequence of bytes or words. A string may contain from 1 byte to 64K bytes.

ASCII: A byte representation of alphanumeric and control characters using the ASCII standard of character representation.

BCD: A byte (unpacked) representation of the decimal digits 0–9.

Packed BCD: A byte (packed) representation of two decimal digits 0–9 storing one digit in each nibble of the byte.

Floating Point: A signed 32, 64, or 80-bit real number representation. (Floating point operands are supported using the iAPX 286/20 Numeric Processor configuration).

Figure 7 graphically represents the data types supported by the iAPX286.

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. I/O instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that $A_{15}-A_8$ are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

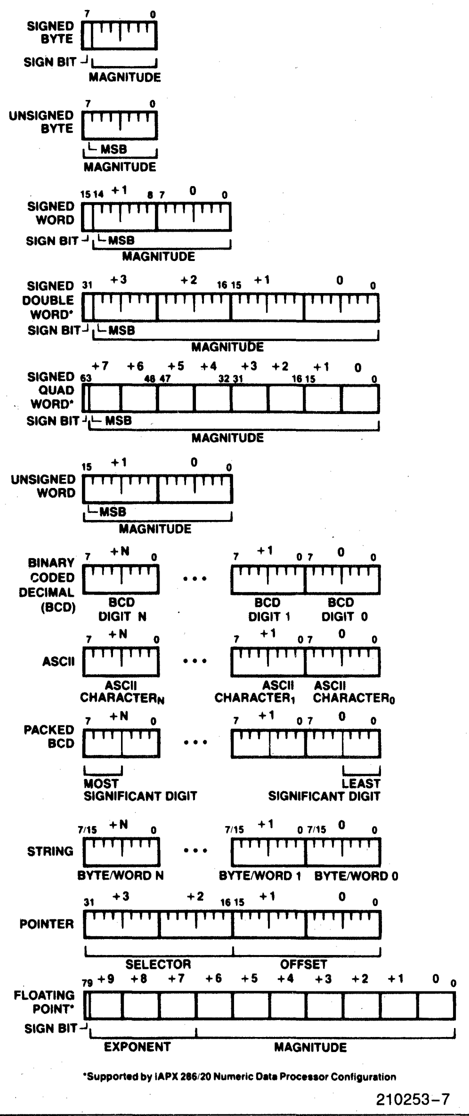


Figure 7. iAPX286 Supported Data Types

Table 4. Interrupt Vector Assignments

Function	Interrupt Number	Related Instructions	Does Return Address Point to Instruction Causing Exception?
Divide error exception	0	DIV, IDIV	Yes
Single step interrupt	1	All	
NMI interrupt	2	INT 2 or NMI pin	
Breakpoint interrupt	3	INT 3	
INTO detected overflow exception	4	INTO	No
BOUND range exceeded exception	5	BOUND	Yes
Invalid opcode exception	6	Any undefined opcode	Yes
Processor extension not available exception	7	ESC or WAIT	Yes
Intel reserved–do not use	8-15		
Processor extension error interrupt	16	ESC or WAIT	
Intel reserved–do not use	17-31		
User defined	32-255		

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Flags) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable. Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. The return address from an exception will always point at the instruction causing the exception and include any leading instruction prefixes.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. For each interrupt, an 8-bit vector must be supplied to the 80286 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

MASKABLE INTERRUPT (INTR)

The 80286 provides a maskable hardware interrupt request pin, INTR. Software enables this input by setting the interrupt flag bit (IF) in the flag word. All 224 user-defined interrupt sources can share this input, yet they can retain separate interrupt handlers. An 8-bit vector read by the CPU during the interrupt acknowledge sequence (discussed in System Interface section) identifies the source of the interrupt.

Further maskable interrupts are disabled while servicing an interrupt by resetting the IF but as part of the response to an interrupt or exception. The saved flag word will reflect the enable status of the processor prior to the interrupt. Until the flag word is restored to the flag register, the interrupt flag will be zero unless specifically set. The interrupt return instruction includes restoring the flag word, thereby restoring the original status of IF.

NON-MASKABLE INTERRUPT REQUEST (NMI)

A non-maskable interrupt input (NMI) is also provided. NMI has higher priority than INTR. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed.

While executing the NMI servicing procedure, the 80286 will service neither further NMI requests, INTR requests, nor the processor extension segment overrun interrupt until an interrupt return (IRET) instruction is executed or the CPU is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. IF is cleared at the beginning of an NMI interrupt to inhibit INTR interrupts.

SINGLE STEP INTERRUPT

The 80286 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single step interrupt and is controlled by the single step flag bit (TF) in the flag word. Once this bit is set, an internal single step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single stepped.

Interrupt Priorities

When simultaneous interrupt requests occur, they are processed in a fixed order as shown in Table 5. Interrupt processing involves saving the flags, return address, and setting CS:IP to point at the first instruction of the interrupt handler. If other interrupts remain enabled they are processed before the first instruction of the current interrupt handler is executed. The last interrupt processed is therefore the first one serviced.

Table 5. Interrupt Processing Order

Order	Interrupt
1	Instruction exception
2	Single step
3	NMI
4	Processor extension segment overrun
5	INTR
6	INT instruction

Initialization and Processor Reset

Processor initialization or start up is accomplished by driving the RESET input pin HIGH. RESET forces the 80286 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as RESET is active. After RESET becomes inactive and an internal processing interval elapses, the 80286 begins execution in real address mode with the instruction at physical location FFFFF0(H). RESET also sets some registers to predefined values as shown in Table 6.

Table 6. 80286 Initial Register State after RESET

Flag word	0002(H)
Machine Status Word	FFF0(H)
Instruction pointer	FFF0(H)
Code segment	F000(H)
Data segment	0000(H)
Extra segment	0000(H)
Stack segment	0000(H)

HOLD must not be active during the time from the leading edge of the initial RESET to 34 CLKs after the trailing edge of the initial RESET of an 80286 system.

Machine Status Word Description

The machine status word (MSW) records when a task switch takes place and controls the operating mode of the 80286. It is a 16-bit register of which the lower four bits are used. One bit places the CPU into protected mode, while the other three bits, as shown in Table 7, control the processor extension interface. After RESET, this register contains FFF0(H) which places the 80286 in iAPX 86 real address mode.

Table 7. MSW Bit Functions

Bit Position	Name	Function
0	PE	Protected mode enable places the 80286 into protected mode and cannot be cleared except by RESET.
1	MP	Monitor processor extension allows WAIT instructions to cause a processor extension not present exception (number 7).
2	EM	Emulate processor extension causes a processor extension not present exception (number 7) on ESC instructions to allow emulating a processor extension.
3	TS	Task switched indicates the next instruction using a processor extension will cause exception 7, allowing software to test whether the current processor extension context belongs to the current task.

The LMSW and SMSW instructions can load and store the MSW in real address mode. The recommended use of TS, EM, and MP is shown in Table 8.

Table 8. Recommended MSW Encodings For Processor Extension Control

TS	MP	EM	Recommended Use	Instructions Causing Exception 7
0	0	0	Initial encoding after RESET. iAPX 286 operation is identical to iAPX 86, 88.	None
0	0	1	No processor extension is available. Software will emulate its function.	ESC
1	0	1	No processor extension is available. Software will emulate its function. The current processor extension context may belong to another task.	ESC
0	1	0	A processor extension exists.	None
1	1	0	A processor extension exists. The current processor extension context may belong to another task. The Exception 7 on WAIT allows software to test for an error pending from a previous processor extension operation.	ESC or WAIT

Halt

The HLT instruction stops program execution and prevents the CPU from using the local bus until restarted. Either NMI, INTR with IF = 1, or RESET will force the 80286 out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

iAPX 86 REAL ADDRESS MODE

The 80286 executes a fully upward-compatible superset of the 8086 instruction set in real address mode. In real address mode the 80286 is object code compatible with 8086 and 8088 software. The real address mode architecture (registers and addressing modes) is exactly as described in the iAPX 286/10 Base Architecture section of this Functional Description.

Memory Size

Physical memory is a contiguous array of up to 1,048,576 bytes (one megabyte) addressed by pins A_0 through A_{19} and $\overline{BHE}$. A_{20} through A_{23} should be ignored.

Memory Addressing

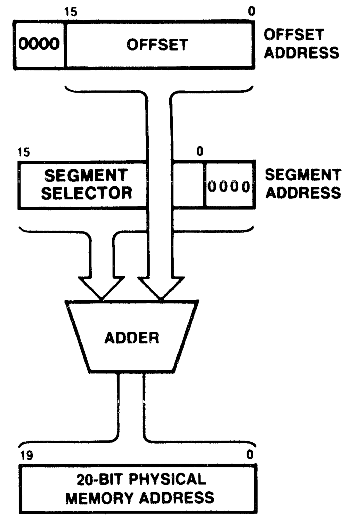
In real address mode physical memory is a contiguous array of up to 1,048,576 bytes (one megabyte) addressed by pins A_0 through A_{19} and $\overline{BHE}$. Address bits A_{20}–A_{23} may not always be zero in real mode. A_{20}–A_{23} should not be used by the system while the 80286 is operating in Real Mode.

The selector portion of a pointer is interpreted as the upper 16 bits of a 20-bit segment address. The lower four bits of the 20-bit segment address are always zero. Segment addresses, therefore, begin on multiples of 16 bytes. See Figure 8 for a graphic representation of address information.

All segments in real address mode are 64K bytes in size and may be read, written, or executed. An exception or interrupt can occur if data operands or instructions attempt to wrap around the end of a segment (e.g. a word with its low order byte at offset FFFF(H) and its high order byte at offset 0000(H). If, in real address mode, the information contained in a segment does not use the full 64K bytes, the unused end of the segment may be overlayed by another segment to reduce physical memory requirements.

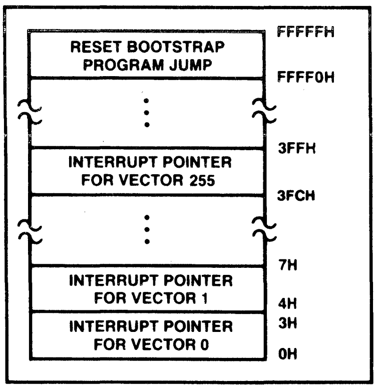
Reserved Memory Locations

The 80286 reserves two fixed areas of memory in real address mode (see Figure 9); system initialization area and interrupt table area. Locations from addresses FFFF0(H) through FFFFF(H) are reserved for system initialization. Initial execution begins at location FFFF0(H). Locations 00000(H) through 003FF(H) are reserved for interrupt vectors.



210253-8

Figure 8. iAPX 86 Real Address Mode Address Calculation



INITIAL CS:IP VALUE IS F000:FFF0.

210253-9

Figure 9. iAPX 86 Real Address Mode Initially Reserved Memory Locations

Table 9. Real Address Mode Addressing Interrupts

Function	Interrupt Number	Related Instructions	Return Address Before Instruction?
Interrupt table limit too small exception	8	INT vector is not within table limit	Yes
Processor extension segment overrun interrupt	9	ESC with memory operand extending beyond offset FFFF(H)	No
Segment overrun exception	13	Word memory reference with offset = FFFF(H) or an attempt to execute past the end of a segment	Yes

Interrupts

Table 9 shows the interrupt vectors reserved for exceptions and interrupts which indicate an addressing error. The exceptions leave the CPU in the state existing before attempting to execute the failing instruction (except for PUSH, POP, PUSHA, or POPA). Refer to the next section on protected mode initialization for a discussion on exception 8.

Protected Mode Initialization

To prepare the 80286 for protected mode, the LIDT instruction is used to load the 24-bit interrupt table base and 16-bit limit for the protected mode interrupt table. This instruction can also set a base and limit for the interrupt vector table in real address mode. After reset, the interrupt table base is initialized to 000000(H) and its size set to 03FF(H). These values are compatible with iAPX 86, 88 software. LIDT should only be executed in preparation for protected mode.

Shutdown

Shutdown occurs when a severe error is detected that prevents further instruction processing by the CPU. Shutdown and halt are externally signalled via a halt bus operation. They can be distinguished by A_1 HIGH for halt and A_1 LOW for shutdown. In real address mode, shutdown can occur under two conditions:

- Exceptions 8 or 13 happen and the IDT limit does not include the interrupt vector.
- A CALL INT or PUSH instruction attempts to wrap around the stack segment when SP is not even.

An NMI input can bring the CPU out of shutdown if the IDT limit is at least 000F(H) and SP is greater than 0005(H), otherwise shutdown can only be exited via the RESET input.

PROTECTED VIRTUAL ADDRESS MODE

The 80286 executes a fully upward-compatible superset of the 8086 instruction set in protected virtual address mode (protected mode). Protected mode also provides memory management and protection mechanisms and associated instructions.

The 80286 enters protected virtual address mode from real address mode by setting the PE (Protection Enable) bit of the machine status word with the Load Machine Status Word (LMSW) instruction. Protected mode offers extended physical and virtual memory address space, memory protection mechanisms, and new operations to support operating systems and virtual memory.

All registers, instructions, and addressing modes described in the iAPX 286/10 Base Architecture section of this Functional Description remain the same. Programs for the iAPX 86, 88, 186, and real address mode 80286 can be run in protected mode; however, embedded constants for segment selectors are different.

Memory Size

The protected mode 80286 provides a 1 gigabyte virtual address space per task mapped into a 16 megabyte physical address space defined by the address pin A_{23}–A_0 and $\overline{BHE}$. The virtual address space may be larger than the physical address space since any use of an address that does not map to a physical memory location will cause a restartable exception.

Memory Addressing

As in real address mode, protected mode uses 32-bit pointers, consisting of 16-bit selector and offset components. The selector, however, specifies an index into a memory resident table rather than the upper 16-bits of a real memory address. The 24-bit base address of the desired segment is obtained from the tables in memory. The 16-bit offset is added to the segment base address to form the physical address as shown in Figure 10. The tables are automatically referenced by the CPU whenever a segment register is loaded with a selector. All iAPX 286 instructions which load a segment register will reference the memory based tables without additional software. The memory based tables contain 8 byte values called descriptors.

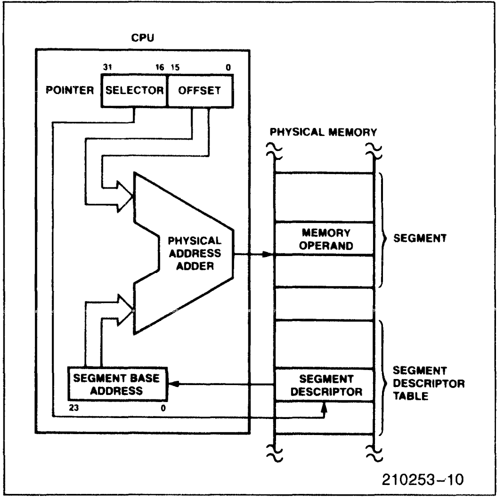


Figure 10. Protected Mode Memory Addressing

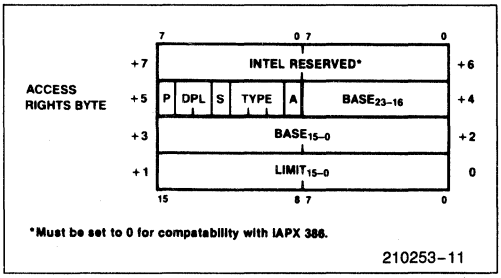
DESCRIPTORS

Descriptors define the use of memory. Special types of descriptors also define new functions for transfer of control and task switching. The 80286 has segment descriptors for code, stack and data segments, and system control descriptors for special system data segments and control transfer operations. Descriptor accesses are performed as locked bus operations to assure descriptor integrity in multi-processor systems.

CODE AND DATA SEGMENT DESCRIPTORS (S = 1)

Besides segment base addresses, code and data descriptors contain other segment attributes including segment size (1 to 64K bytes), access rights (read only, read/write, execute only, and execute/read), and presence in memory (for virtual memory systems) (See Figure 11). Any segment usage violating a segment attribute indicated by the segment descriptor will prevent the memory cycle and cause an exception or interrupt.

Code or Data Segment Descriptor



Access Rights Byte Definition

	Bit Position	Name	Function		
	7	Present (P)	P = 1	Segment is mapped into physical memory.	
			P = 0	No mapping to physical memory exits, base and limit are not used.	
	6–5	Descriptor Privilege Level (DPL)		Segment privilege attribute used in privilege tests.	
	4	Segment Descriptor (S)	S = 1	Code or Data (includes stacks) segment descriptor	
			S = 0	System Segment Descriptor or Gate Descriptor	
Type Field Definition	3	Executable (E)	E = 0	Data segment descriptor type is:	If Data Segment (S = 1, E = 0)
	2	Expansion Direction (ED)	ED 0	Expand up segment, offsets must be ≤ limit.	
			ED = 1	Expand down segment, offsets must be > limit.	
	1	Writeable (W)	W = 0	Data segment may not be written into.	
			W = 1	Data segment may be written into.	
	3	Executable (E)	E = 1	Code Segment Descriptor type is:	If Code Segment (S = 1, E = 1)
	2	Conforming (C)	C = 1	Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.	
	1	Readable (R)	R = 0	Code segment may not be read	
			R = 1	Code segment may be read.	
	0	Accessed (A)	A = 0	Segment has not been accessed.	
			A = 1	Segment selector has been loaded into segment register or used by selector test instructions.	

Figure 11. Code and Data Segment Descriptor Formats

Code and data (including stack data) are stored in two types of segments: code segments and data segments. Both types are identified and defined by segment descriptors (S = 1). Code segments are identified by the executable (E) bit set to 1 in the descriptor access rights byte. The access rights byte of both code and data segment descriptor types have three fields in common: present (P) bit, Descriptor Privilege Level (DPL), and accessed (A) bit. If P = 0, any attempted use of this segment will cause a not-present exception. DPL specifies the privilege level of the segment descriptor. DPL controls when the descriptor may be used by a task (refer to privilege discussion below). The A bit shows whether the segment has been previously accessed for usage profiling, a necessity for virtual memory systems. The CPU will always set this bit when accessing the descriptor.

Data segments (S = 1, E = 0) may be either read-only or read-write as controlled by the W bit of the access rights byte. Read-only (W = 0) data segments may not be written into. Data segments may grow in two directions, as determined by the Expansion Direction (ED) bit: upwards (ED = 0) for data segments, and downwards (ED = 1) for a segment containing a stack. The limit field for a data segment descriptor is interpreted differently depending on the ED bit (see Figure 11).

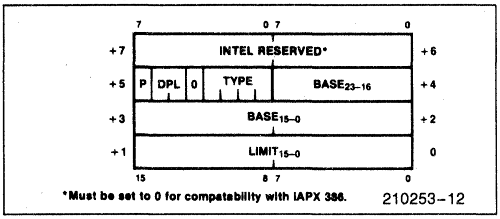
A code segment (S = 1, E = 1) may be execute-only or execute/read as determined by the Readable (R) bit. Code segments may never be written into and execute-only code segments (R = 0) may not be read. A code segment may also have an attribute called conforming (C). A conforming code segment may be shared by programs that execute at different privilege levels. The DPL of a conforming code segment defines the range of privilege levels at which the segment may be executed (refer to privilege discussion below). The limit field identifies the last byte of a code segment.

SYSTEM SEGMENT DESCRIPTORS (S = 0, TYPE = 1–3)

In addition to code and data segment descriptors, the protected mode 80286 defines System Segment Descriptors. These descriptors define special system data segments which contain a table of descriptors (Local Descriptor Table Descriptor) or segments which contain the execution state of a task (Task State Segment Descriptor).

Figure 12 gives the formats for the special system data segment descriptors. The descriptors contain a 24-bit base address of the segment and a 16-bit limit. The access byte defines the type of descriptor, its state and privilege level. The descriptor contents are valid and the segment is in physical memory if P = 1. If P = 0, the segment is not valid. The DPL field is only used in Task State Segment descriptors and indicates the privilege level at which the descriptor may be used (see Privilege). Since the Local Descriptor Table descriptor may only be used by a special privileged instruction, the DPL field is not used. Bit 4 of the access byte is 0 to indicate that it is a system control descriptor. The type field specifies the descriptor type as indicated in Figure 12.

System Segment Descriptor



System Segment Descriptor Fields

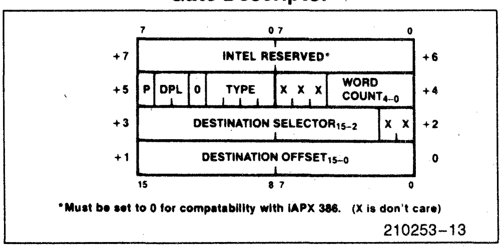
Name	Value	Description
TYPE	1 2 3	Available Task State Segment (TSS) Local Descriptor Table Busy Task State Segment (TSS)
P	0 1	Descriptor contents are not valid Descriptor contents are valid
DPL	0–3	Descriptor Privilege Level
BASE	24-bit number	Base Address of special system data segment in real memory
LIMIT	16-bit number	Offset of last byte in segment

Figure 12. System Segment Descriptor Format

GATE DESCRIPTORS (S = 0, TYPE = 4–7)

Gates are used to control access to entry points within the target code segment. The gate descriptors are call gates, task gates, interrupt gates and trap gates. Gates provide a level of indirection between the source and destination of the control transfer. This indirection allows the CPU to automatically perform protection checks and control entry point of the destination. Call gates are used to change privilege levels (see Privilege), task gates are used to perform a task switch, and interrupt and trap gates are used to specify interrupt service routines. The interrupt gate disables interrupts (resets IF) while the trap gate does not.

Gate Descriptor



Gate Descriptor Fields

Name	Value	Description
TYPE	4 5 6 7	–Call Gate –Task Gate –Interrupt Gate –Trap Gate
P	0 1	–Descriptor Contents are not valid –Descriptor Contents are valid
DPL	0–3	Descriptor Privilege Level
WORD COUNT	0–31	Number of words to copy from callers stack to called procedures stack. Only used with call gate.
DESTINATION SELECTOR	16-bit selector	Selector to the target code segment (Call, Interrupt or Trap Gate) Selector to the target task state segment (Task Gate)
DESTINATION OFFSET	16-bit offset	Entry point within the target code segment

Figure 13. Gate Descriptor Format

Figure 13 shows the format of the gate descriptors. The descriptor contains a destination pointer that points to the descriptor of the target segment and the entry point offset. The destination selector in an interrupt gate, trap gate, and call gate must refer to a code segment descriptor. These gate descriptors contain the entry point to prevent a program from constructing and using an illegal entry point. Task gates may only refer to a task state segment. Since task gates invoke a task switch, the destination offset is not used in the task gate.

Exception 13 is generated when the gate is used if a destination selector does not refer to the correct descriptor type. The word count field is used in the call gate descriptor to indicate the number of parameters (0–31 words) to be automatically copied from the caller's stack to the stack of the called routine when a control transfer changes privilege levels. The word count field is not used by any other gate descriptor.

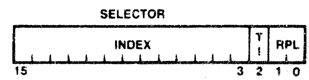
The access byte format is the same for all gate descriptors. P = 1 indicates that the gate contents are valid. P = 0 indicates the contents are not valid and causes exception 11 if referenced. DPL is the descriptor privilege level and specifies when this descriptor may be used by a task (refer to privilege discussion below). Bit 4 must equal 0 to indicate a system control descriptor. The type field specifies the descriptor type as indicated in Figure 13.

SEGMENT DESCRIPTOR CACHE REGISTERS

A segment descriptor cache register is assigned to each of the four segment registers (CS, SS, DS, ES). Segment descriptors are automatically loaded (cached) into a segment descriptor cache register (Figure 14) whenever the associated segment register is loaded with a selector. Only segment descriptors may be loaded into segment descriptor cache registers. Once loaded, all references to that segment of memory use the cached descriptor information instead of reaccessing the descriptor. The descriptor cache registers are not visible to programs. No instructions exist to store their contents. They only change when a segment register is loaded.

SELECTOR FIELDS

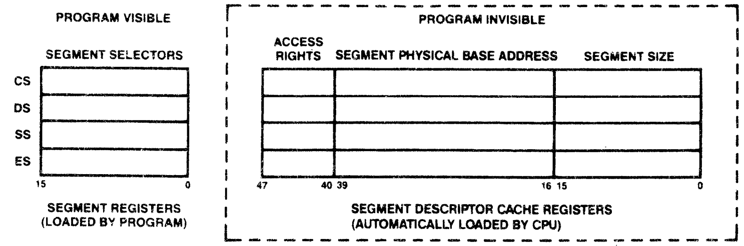
A protected mode selector has three fields: descriptor entry index, local or global descriptor table indicator (TI), and selector privilege (RPL) as shown in Figure 15. These fields select one of two memory based tables of descriptors, select the appropriate table entry and allow highspeed testing of the selector's privilege attribute (refer to privilege discussion below).



BITS	NAME	FUNCTION
1–0	REQUESTED PRIVILEGE LEVEL (RPL)	INDICATES SELECTOR PRIVILEGE LEVEL DESIRED
2	TABLE INDICATOR (TI)	TI = 0 USE GLOBAL DESCRIPTOR TABLE (GDT) TI = 1 USE LOCAL DESCRIPTOR TABLE (LDT)
15–3	INDEX	SELECT DESCRIPTOR ENTRY IN TABLE

210253–15

Figure 15. Selector Fields



210253–14

Figure 14. Descriptor Cache Registers

LOCAL AND GLOBAL DESCRIPTOR TABLES

Two tables of descriptors, called descriptor tables, contain all descriptors accessible by a task at any given time. A descriptor table is a linear array of up to 8192 descriptors. The upper 13 bits of the selector value are an index into a descriptor table. Each table has a 24-bit base register to locate the descriptor table in physical memory and a 16-bit limit register that confine descriptor access to the defined limits of the table as shown in Figure 16. A restartable exception (13) will occur if an attempt is made to reference a descriptor outside the table limits.

One table, called the Global Descriptor table (GDT), contains descriptors available to all tasks. The other table, called the Local Descriptor Table (LDT), contains descriptors that can be private to a task. Each task may have its own private LDT. The GDT may contain all descriptor types except interrupt and trap descriptors. The LDT may contain only segment, task gate, and call gate descriptors. A segment cannot be accessed by a task if its segment descriptor does not exist in either descriptor table at the time of access.

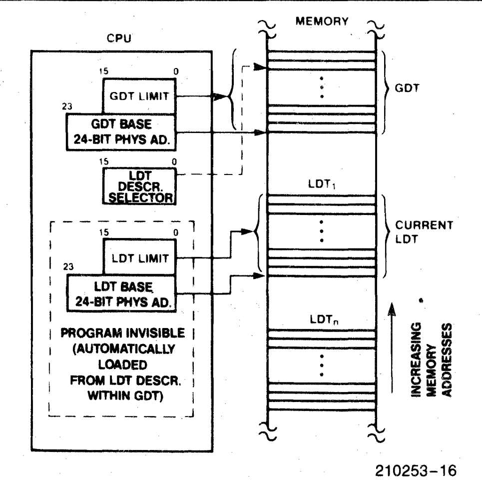


Figure 16. Local and Global Descriptor Table Definition

The LGDT and LLDT instructions load the base and limit of the global and local descriptor tables. LGDT and LLDT are privileged, i.e. they may only be executed by trusted programs operating at level 0. The LGDT instruction loads a six byte field containing the 16-bit table limit and 24-bit physical base address of the Global Descriptor Table as shown in Figure 17. The LDT instruction loads a selector which refers to a Local Descriptor Table descriptor containing the base address and limit for an LDT, as shown in Figure 12.

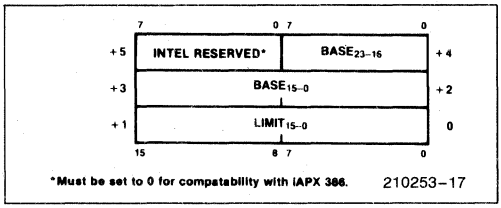


Figure 17. Global Descriptor Table and Interrupt Descriptor Table Data Type

INTERRUPT DESCRIPTOR TABLE

The protected mode 80286 has a third descriptor table, called the Interrupt Descriptor Table (IDT) (see Figure 18), used to define up to 256 interrupts. It may contain only task gates, interrupt gates and trap gates. The IDT (Interrupt Descriptor Table) has a 24-bit physical base and 16-bit limit register in the CPU. The privileged LIDT instruction loads these registers with a six byte value of identical form to that of the LGDT instruction (see Figure 17 and Protected Mode Initialization).

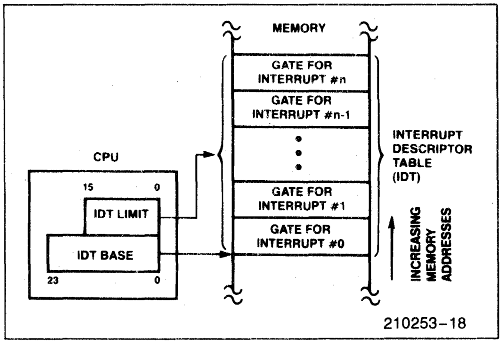
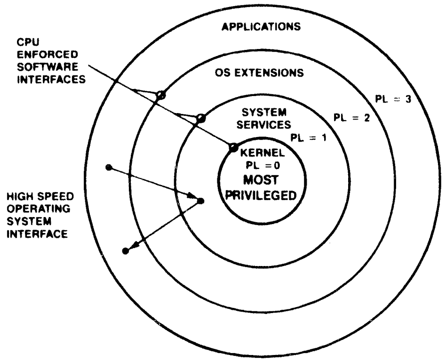


Figure 18. Interrupt Descriptor Table Definition

References to IDT entries are made via INT instructions, external interrupt vectors, or exceptions. The IDT must be at least 256 bytes in size to allocate space for all reserved interrupts.

Privilege

The 80286 has a four-level hierarchical privilege system which controls the use of privileged instructions and access to descriptors (and their associated segments) within a task. Four-level privilege, as shown in Figure 19, is an extension of the user/supervisor mode commonly found in minicomputers. The privilege levels are numbered 0 through 3. Level 0 is the



NOTE: PL BECOMES NUMERICALLY LOWER AS PRIVILEGE LEVEL INCREASES

210253-19

most privileged level. Privilege levels provide protection within a task. (Tasks are isolated by providing private LDT's for each task.) Operating system routines, interrupt handlers, and other system software can be included and protected within the virtual address space of each task using the four levels of privilege. Each task in the system has a separate stack for each of its privilege levels.

Tasks, descriptors, and selectors have a privilege level attribute that determines whether the descriptor may be used. Task privilege effects the use of instructions and descriptors. Descriptor and selector privilege only effect access to the descriptor.

TASK PRIVILEGE

A task always executes at one of the four privilege levels. The task privilege level at any specific instant is called the Current Privilege Level (CPL) and is defined by the lower two bits of the CS register. CPL cannot change during execution in a single code segment. A task's CPL may only be changed by control transfers through gate descriptors to a new code segment (See Control Transfer). Tasks begin executing at the CPL value specified by the code segment selector within TSS when the task is initiated via a task switch operation (See Figure 20). A task executing at Level 0 can access all data segments defined in the GDT and the task's LDT and is considered the most trusted level. A task executing a Level 3 has the most restricted access to data and is considered the least trusted level.

DESCRIPTOR PRIVILEGE

Descriptor privilege is specified by the Descriptor Privilege Level (DPL) field of the descriptor access byte. DPL specifies the least trusted task privilege level (CPL) at which a task may access the descriptor. Descriptors with DPL = 0 are the most protected. Only tasks executing at privilege level 0 (CPL = 0) may access them. Descriptors with DPL = 3 are the least protected (i.e. have the least restricted access) since tasks can access them when CPL = 0, 1, 2, or 3. This rule applies to all descriptors, except LDT descriptors.

SELECTOR PRIVILEGE

Selector privilege is specified by the Requested Privilege Level (RPL) field in the least significant two bits of a selector. Selector RPL may establish a less trusted privilege level than the current privilege level for the use of a selector. This level is called the task's effective privilege level (EPL). RPL can only reduce the scope of a task's access to data with this selector. A task's effective privilege is the numeric maximum of RPL and CPL. A selector with RPL = 0 imposes no additional restriction on its use while a selector with RPL = 3 can only refer to segments at privilege Level 3 regardless of the task's CPL. RPL is generally used to verify that pointer parameters passed to a more trusted procedure are not allowed to use data at a more privileged level than the caller (refer to pointer testing instructions).

Descriptor Access and Privilege Validation

Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL. The two basic types of segment accesses are control transfer (selectors loaded into CS) and data (selectors loaded into DS, ES or SS).

DATA SEGMENT ACCESS

Instructions that load selectors into DS and ES must refer to a data segment descriptor or readable code segment descriptor. The CPL of the task and the RPL of the selector must be the same as or more privileged (numerically equal to or lower than) than the descriptor DPL. In general, a task can only access data segments at the same or less privileged levels than the CPL or RPL (whichever is numerically higher) to prevent a program from accessing data it cannot be trusted to use.

An exception to the rule is a readable conforming code segment. This type of code segment can be read from any privilege level.

If the privilege checks fail (e.g. DPL is numerically less than the maximum of CPL and RPL) or an incorrect type of descriptor is referenced (e.g. gate descriptor or execute only code segment) exception 13 occurs. If the segment is not present, exception 11 is generated.

Instructions that load selectors into SS must refer to data segment descriptors for writable data segments. The descriptor privilege (DPL) and RPL must equal CPL. All other descriptor types or a privilege level violation will cause exception 13. A not present fault causes exception 12.

CONTROL TRANSFER

Four types of control transfer can occur when a selector is loaded into CS by a control transfer operation (see Table 10). Each transfer type can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules (e.g. JMP through a call gate or RET to a Task State Segment) will cause exception 13.

The ability to reference a descriptor for control transfer is also subject to rules of privilege. A CALL or JUMP instruction may only reference a code segment descriptor with DPL equal to the task CPL or a conforming segment with DPL of equal or greater privilege than CPL. The RPL of the selector used to reference the code descriptor must have as much privilege as CPL.

RET and IRET instructions may only reference code segment descriptors with descriptor privilege equal to or less privileged than the task CPL. The selector loaded into CS is the return address from the stack. After the return, the selector RPL is the task's new CPL. If CPL changes, the old stack pointer is popped after the return address.

When a JMP or CALL references a Task State Segment descriptor, the descriptor DPL must be the same or less privileged than the task's CPL. Reference to a valid Task State Segment descriptor causes a task switch (see Task Switch Operation). Reference to a Task State Segment descriptor at a more privileged level than the task's CPL generates exception 13.

When an instruction or interrupt references a gate descriptor, the gate DPL must have the same or less privilege than the task CPL. If DPL is at a more privileged level than CPL, exeception 13 occurs. If the destination selector contained in the gate references a code segment descriptor, the code segment descriptor DPL must be the same or more privileged than the task CPL. If not, Exception 13 is issued. After the control transfer, the code segment descriptors DPL is the task's new CPL. If the destination selector in the gate references a task state segment, a task switch is automatically performed (see Task Switch Operation).

The privilege rules on control transfer require:

— JMP or CALL direct to a code segment (code segment descriptor) can only be to a conforming segment with DPL of equal or greater privilege than CPL or a non-conforming segment at the same privilege level.

— interrupts within the task or calls that may change privilege levels, can only transfer control through a gate at the same or a less privileged level than CPL to a code segment at the same or more privileged level than CPL.

— return instructions that don't switch tasks can only return control to a code segment at the same or less privileged level.

— task switch can be performed by a call, jump or interrupt which references either a task gate or task state segment at the same or less privileged level.

Table 10. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL.	CALL	Call Gate	GDT/LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
	CALL, JMP	Task State Segment	GDT
Task Switch	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag word) = 0
**NT (Nested Task bit of flag word) = 1

PRIVILEGE LEVEL CHANGES

Any control transfer that changes CPL within the task, causes a change of stacks as part of the operation. Initial values of SS:SP for privilege levels 0, 1, and 2 are kept in the task state segment (refer to Task Switch Operation). During a JMP or CALL control transfer, the new stack pointer is loaded into the SS and SP registers and the previous stack pointer is pushed onto the new stack.

When returning to the original privilege level, its stack is restored as part of the RET or IRET instruction operation. For subroutine calls that pass parameters on the stack and cross privilege levels, a fixed number of words, as specified in the gate, are copied from the previous stack to the current stack. The inter-segment RET instruction with a stack adjustment value will correctly restore the previous stack pointer upon return.

Protection

The 80286 includes mechanisms to protect critical instructions that affect the CPU execution state (e.g. HLT) and code or data segments from improper usage. These protection mechanisms are grouped into three forms:

Restricted *usage* of segments (e.g. no write allowed to read-only data segments). The only segments available for use are defined by descriptors in the Local Descriptor Table (LDT) and Global Descriptor Table (GDT).

Restricted *access* to segments via the rules of privilege and descriptor usage.

Privileged instructions or operations that may only be executed at certain privilege levels as determined by the CPL and I/O Privilege Level (IOPL). The IOPL is defined by bits 14 and 13 of the flag word.

These checks are performed for all instructions and can be split into three categories: segment load checks (Table 11), operand reference checks (Table 12), and privileged instruction checks (Table 13). Any violation of the rules shown will result in an exception. A not-present exception related to the stack segment causes exception 12.

The IRET and POPF instructions do not perform some of their defined functions if CPL is not of sufficient privilege (numerically small enough). Precisely these are:

- The IF bit is not changed if CPL > IOPL.
- The IOPL field of the flag word is not changed if CPL > 0.

No exceptions or other indication are given when these conditions occur.

Table 11
Segment Register Load Checks

Error Description	Exception Number
Descriptor table limit exceeded	13
Segment descriptor not-present	11 or 12
Privilege rules violated	13
Invalid descriptor/segment type segment register load: —Read only data segment load to SS —Special Control descriptor load to DS, ES, SS —Execute only segment load to DS, ES, SS —Data segment load to CS —Read/Execute code segment load to SS	13

Table 12. Operand Reference Checks

Error Description	Exception Number
Write into code segment	13
Read from execute-only code segment	13
Write to read-only data segment	13
Segment limit exceeded[1]	12 or 13

NOTE:
Carry out in offset calculations is ignored.

Table 13. Privileged Instruction Checks

Error Description	Exception Number
CPL ≠ 0 when executing the following instructions: LIDT, LLDT, LGDT, LTR, LMSW, CTS, HLT	13
CPL > IOPL when executing the following instructions: INS, IN, OUTS, OUT, STI, CLI, LOCK	13

EXCEPTIONS

The 80286 detects several types of exceptions and interrupts, in protected mode (see Table 14). Most are restartable after the exceptional condition is removed. Interrupt handlers for most exceptions can read an error code, pushed on the stack after the return address, that identifies the selector involved (0 if none). The return address normally points to the failing instruction, including all leading prefixes. For a processor extension segment overrun exception, the return address will not point at the ESC instruction that caused the exception; however, the processor extension registers may contain the address of the failing instruction.

Table 14. Protected Mode Exceptions

Interrupt Vector	Function	Return Address At Falling Instruction?	Always Restart-able?	Error Code on Stack?
8	Double exception detected	Yes	No[2]	Yes
9	Processor extension segment overrun	No	No[2]	No
10	Invalid task state segment	Yes	Yes	Yes
11	Segment not present	Yes	Yes	Yes
12	Stack segment overrun or stack segment not present	Yes	Yes[1]	Yes
13	General protection	Yes	No[2]	Yes

NOTE:
1. When a PUSHA or POPA instruction attempts to wrap around the stack segment, the machine state after the exception will not be restartable because stack segment wrap around is not permitted. This condition is identified by the value of the saved SP being either 0000(H), 0001(H), FFFE(H), or FFFF(H).
2. These exceptions indicate a violation to privilege rules or usage rules has occurred. Restart is generally not attempted under those conditions.

These exceptions indicate a violation to privilege rules or usage rules has occurred. Restart is generally not attempted under those conditions.

All these checks are performed for all instructions and can be split into three categories: segment load checks (Table 11), operand reference checks (Table 12), and privileged instruction checks (Table 13). Any violation of the rules shown will result in an exception. A not-present exception causes exception 11 or 12 and is restartable.

Special Operations

TASK SWITCH OPERATION

The 80286 provides a built-in task switch operation which saves the entire 80286 execution state (registers, address space, and a link to the previous task), loads a new execution state, and commences execution in the new task. Like gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS) or task gate descriptor in the GDT or LDT. An INT n instruction, exception, or external interrupt may also invoke the task switch operation by selecting a task gate descriptor in the associated IDT descriptor entry.

The TSS descriptor points at a segment (see Figure 20) containing the entire 80286 execution state while a task gate descriptor contains a TSS selector. The limit field of the descriptor must be >002B(H).

Each task must have a TSS associated with it. The current TSS is identified by a special register in the 80286 called the Task Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TR are loaded whenever TR is loaded with a new selector.

The IRET instruction is used to return control to the task that called the current task or was interrupted. Bit 14 in the flag register is called the Nested Task (NT) bit. It controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular current task by popping values off the stack; when NT = 1, IRET performs a task switch operation back to the previous task.

When a CALL, JMP, or INT instruction initiates a task switch, the old (except for case of JMP) and new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT. NT may also be set or cleared by POPF or IRET instructions.

The task state segment is marked busy by changing the descriptor type field from Type 1 to Type 3. Use of a selector that references a busy task state segment causes Exception 13.

PROCESSOR EXTENSION CONTEXT SWITCHING

The context of a processor extension (such as the 80287 numerics processor) is not changed by the task switch operation. A processor extension context need only be changed when a different task attempts to use the processor extension (which still contains the context of a previous task). The 80286 detects the first use of a processor extension after a task switch by causing the processor extension not present exception (7). The interrupt handler may then decide whether a context change is necessary.

Whenever the 80286 switches tasks, it sets the Task Switched (TS) bit of the MSW. TS indicates that a processor extension context may belong to a different task than the current one. The processor extension not present exception (7) will occur when attempting to execute an ESC or WAIT instruction if TS = 1 and a processor extension is present (MP = 1 in MSW).

POINTER TESTING INSTRUCTIONS

The iAPX 286 provides several instructions to speed pointer testing and consistency checks for maintaining system integrity (see Table 15). These instructions use the memory management hardware to verify that a selector value refers to an appropriate segment without risking an exception. A condition flag (ZF) indicates whether use of the selector or segment will cause an exception.

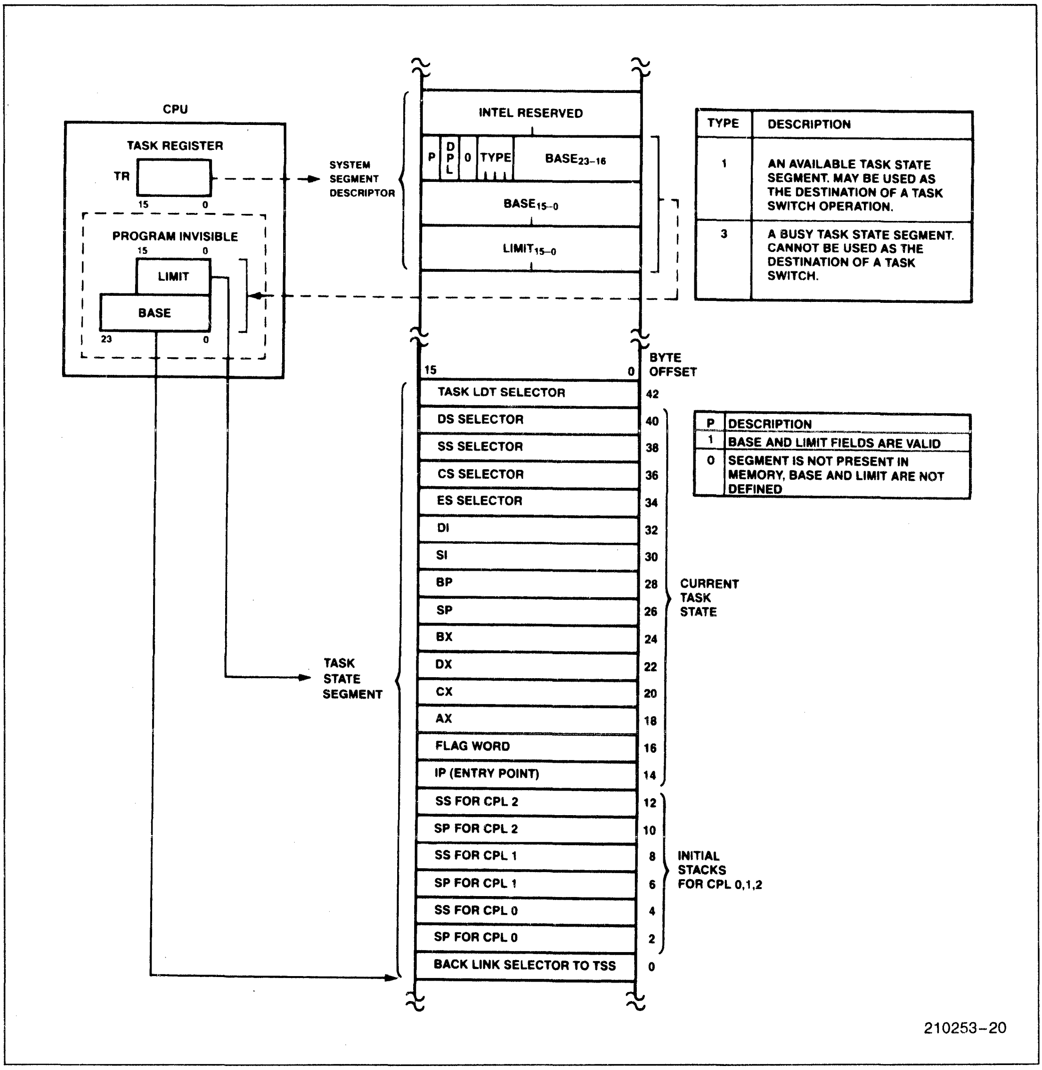


Figure 20. Task State Segment and TSS Registers

Table 15. 80286 Pointer Test Instructions

Instruction	Operands	Function
ARPL	Selector, Register	Adjust Requested Privilege Level: adjusts the RPL of the selector to the numeric maximum of current selector RPL value and the RPL value in the register. Set zero flag if selector RPL was changed by ARPL.
VERR	Selector	VERify for Read: sets the zero flag if the segment referred to by the selector can be read.
VERW	Selector	VERify for Write: sets the zero flag if the segment referred to by the selector can be written.
LSL	Register, Selector	Load Segment Limit: reads the segment limit into the register if privilege rules and descriptor type allow. Set zero flag if successful.
LAR	Register, Selector	Load Access Rights: reads the descriptor access rights byte into the register if privilege rules allow. Set zero flag if successful.

DOUBLE FAULT AND SHUTDOWN

If two separate exceptions are detected during a single instruction execution, the 80286 performs the double fault exception (8). If an execution occurs during processing of the double fault exception, the 80286 will enter shutdown. During shutdown no further instructions or exceptions are processed. Either NMI (CPU remains in protected mode) or RESET (CPU exits protected mode) can force the 80286 out of shutdown. Shutdown is externally signalled via a HALT bus operation with A_1 HIGH.

PROTECTED MODE INITIALIZATION

The 80286 initially executes in real address mode after RESET. To allow initialization code to be placed at the top of physical memory, A_{23-20} will be HIGH when the 80286 performs memory references relative to the CS register until CS is changed. A_{23-20} will be zero for references to the DS, ES, or SS segments. Changing CS in real address mode will force A_{23-20} LOW whenever CS is used again. The initial CS:IP value of F000:FFF0 provides 64K bytes of code space for initialization code without changing CS.

Protected mode operation requires several registers to be initialized. The GDT and IDT base registers must refer to a valid GDT and IDT. After executing the LMSW instruction to set PE, the 80286 must immediately execute an intra-segment JMP instruction to clear the instruction queue of instructions decoded in real address mode.

To force the 80286 CPU registers to match the initial protected mode state assumed by software, execute a JMP instruction with a selector referring to the initial TSS used in the system. This will load the task register, local descriptor table register, segment registers and initial general register state. The TR should point at a valid TSS since any task switch operation involves saving the current task state.

SYSTEM INTERFACE

The 80286 system interface appears in two forms: a local bus and a system bus. The local bus consists of address, data, status, and control signals at the pins of the CPU. A system bus is any buffered version of the local bus. A system bus may also differ from the local bus in terms of coding of status and control lines and/or timing and loading of signals. The iAPX 286 family includes several devices to generate standard system buses such as the IEEE 796 standard MULTIBUS.

Bus Interface Signals and Timing

The iAPX 286 microsystem local bus interfaces the 80286 to local memory and I/O components. The interface has 24 address lines, 16 data lines, and 8 status and control signals.

The 80286 CPU, 82284 clock generator, 82288 bus controller, 82289 bus arbiter, 8286/7 tranceivers, and 8282/3 latches provide a buffered and decoded system bus interface. The 82284 generates the system clock and synchronizes $\overline{\text{READY}}$ and RESET. The 82288 converts bus operation status encoded by the 80286 into command and bus control signals. The 82289 bus arbiter generates Multibus bus arbitration signals. These components can provide the timing and electrical power drive levels required for most system bus interfaces including the Multibus.

Physical Memory and I/O Interface

A maximum of 16 megabytes of physical memory can be addressed in protected mode. One megabyte can be addressed in real address mode. Memory is accessible as bytes or words. Words consist of any two consecutive bytes addressed with the least significant byte stored in the lowest address.

Byte transfers occur on either half of the 16-bit local data bus. Even bytes are accessed over D_{7-0} while odd bytes are transferred over D_{15-8}. Even-addressed words are transferred over D_{15-0} in one bus cycle, while odd-addressed word require *two* bus operations. The first transfers data on D_{15-8}, and the second transfers data on D_{7-0}. Both byte data transfers occur automatically, transparent to software.

Two bus signals, A_0 and $\overline{BHE}$, control transfers over the lower and upper halves of the data bus. Even address byte transfers are indicated by A_0 LOW and $\overline{BHE}$ HIGH. Odd address byte transfers are indicated by A_0 HIGH and $\overline{BHE}$ LOW. Both A_0 and $\overline{BHE}$ are LOW for even address word transfers.

The I/O address space contains 64K addresses in both modes. The I/O space is accessible as either bytes or words, as is memory. Byte wide peripheral devices may be attached to either the upper or lower byte of the data bus. Byte-wide I/O devices attached to the upper data byte (D_{15-8}) are accessed with odd I/O addresses. Devices on the lower data byte are accessed with even I/O addresses. An interrupt controller such as Intel's 8259A must be connected to the lower data byte (D_{7-0}) for proper return of the interrupt vector.

Bus Operation

The 80286 uses a double frequency system clock (CLK input) to control bus timing. All signals on the local bus are measured relative to the system CLK input. The CPU divides the system clock by 2 to produce the internal processor clock, which determines bus state. Each processor clock is composed of two system clock cycles named phase 1 and phase 2. The 82284 clock generator output (PCLK) identifies the next phase of the processor clock. (See Figure 21.)

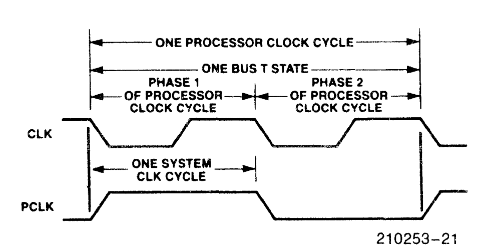


Figure 21. System and Processor Clock Relationships

Six types of bus operations are supported; memory read, memory write, I/O read, I/O write, interrupt acknowledge, and halt/shutdown. Data can be transferred at a maximum rate of one word per two processor clock cycles.

The iAPX 286 bus has three basic states: idle (T_i), send status (T_S), and perform command (T_C). The 80286 CPU also has a fourth local bus state called hold (T_h). T_h indicates that the 80286 has surrendered control of the local bus to another bus master in response to a HOLD request.

Each bus state is one processor clock long. Figure 22 shows the four 80286 local bus states and allowed transitions.

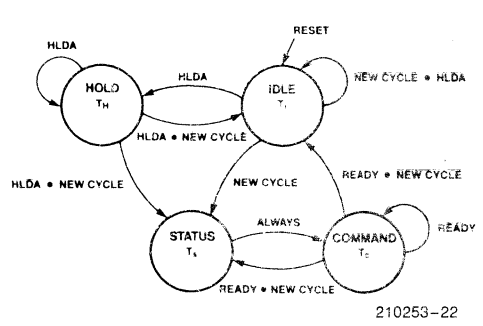


Figure 22. 80286 Bus States

Bus States

The idle (T_i) state indicates that no data transfers are in progress or requested. The first active state T_S is signaled by status line $\overline{S1}$ or $\overline{S0}$ going LOW and identifying phase 1 of the processor clock. During T_S, the command encoding, the address, and data (for a write operation) are available on the 80286 output pins. The 82288 bus controller decodes the status signals and generates Multibus compatible read/write command and local transceiver control signals.

After T_S, the perform command (T_C) state is entered. Memory or I/O devices respond to the bus operation during T_C, either transferring read data to the CPU or accepting write data. T_C states may be repeated as often as necessary to assure sufficient time for the memory or I/O device to respond. The $\overline{READY}$ signal determines whether T_C is repeated. A repeated T_C state is called a wait state.

During hold (T_h), the 80286 will float all address, data, and status output pins enabling another bus master to use the local bus. The 80286 HOLD input signal is used to place the 80286 into the T_h state. The 80286 HLDA output signal indicates that the CPU has entered T_h.

Pipelined Addressing

The 80286 uses a local bus interface with pipelined timing to allow as much time as possible for data access. Pipelined timing allows a new bus operation to be initiated every two processor cycles, while allowing each individual bus operation to last for three processor cycles.

The timing of the address outputs is pipelined such that the address of the next bus operation becomes available during the current bus operation. Or in other words, the first clock of the next bus operation is overlapped with the last clock of the current bus operation. Therefore, address decode and routing logic can operate in advance of the next bus operation.

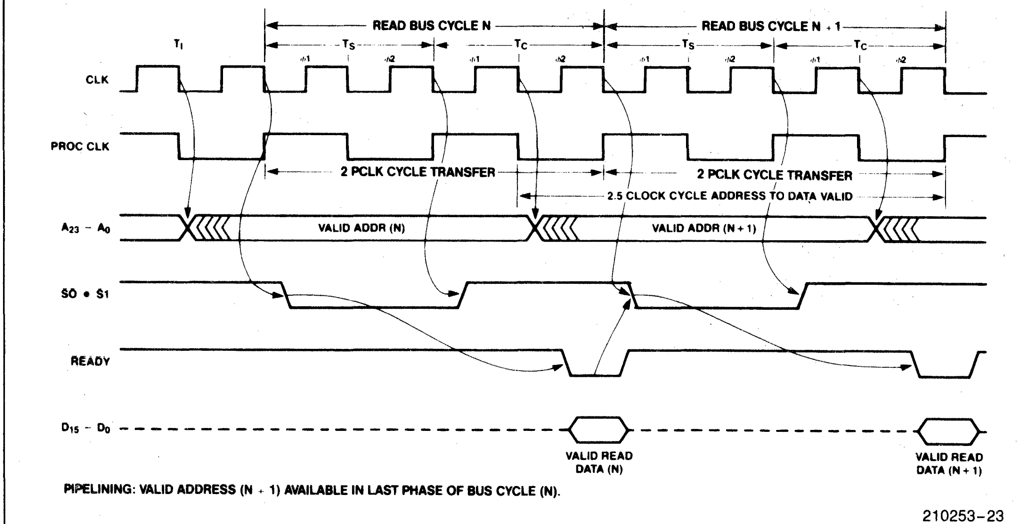


Figure 23. Basic Bus Cycle

External address latches may hold the address stable for the entire bus operation, and provide additional AC and DC buffering.

The 80286 does not maintain the address of the current bus operation during all T_C states. Instead, the address for the next bus operation may be emitted during phase 2 of any T_C. The address remains valid during phase 1 of the first T_C to guarantee hold time, relative to ALE, for the address latch inputs.

Bus Control Signals

The 82288 bus controller provides control signals; address latch enable (ALE), Read/Write commands, data transmit/receive (DT/$\overline{R}$), and data enable (DEN) that control the address latches, data transceivers, write enable, and output enable for memory and I/O systems.

The Address Latch Enable (ALE) output determines when the address may be latched. ALE provides at least one system CLK period of address hold time from the end of the previous bus operation until the address for the next bus operation appears at the latch outputs. This address hold time is required to support MULTIBUS® and common memory systems.

The data bus transceivers are controlled by 82288 outputs Data Enable (DEN) and Data Transmit/Receive (DT/$\overline{R}$). DEN enables the data transceivers; while DT/$\overline{R}$ controls tranceiver direction. DEN and DT/$\overline{R}$ are timed to prevent bus contention between the bus master, data bus transceivers, and system data bus transceivers.

Command Timing Controls

Two system timing customization options, command extension and command delay, are provided on the iAPX 286 local bus.

Command extension allows additional time for external devices to respond to a command and is analogous to inserting wait states on the 8086. External logic can control the duration of any bus operation such that the operation is only as long as necessary. The $\overline{\text{READY}}$ input signal can extend any bus operation for as long as necessary.

Command delay allows an increase of address or write data setup time to system bus command active for any bus operation by delaying when the system bus command becomes active. Command delay is controlled by the 82288 CMDLY input. After T_S, the bus controller samples CMDLY at each failing edge of CLK. If CMDLY is HIGH, the 82288 will not activate the command signal. When CMDLY is LOW, the 82288 will activate the command signal. After the command becomes active, the CMDLY input is not sampled.

When a command is delayed, the available response time from command active to return read data or accept write data is less. To customize system bus timing, an address decoder can determine which bus operations require delaying the command. The CMDLY input does not affect the timing of ALE, DEN, or DT/$\overline{R}$.

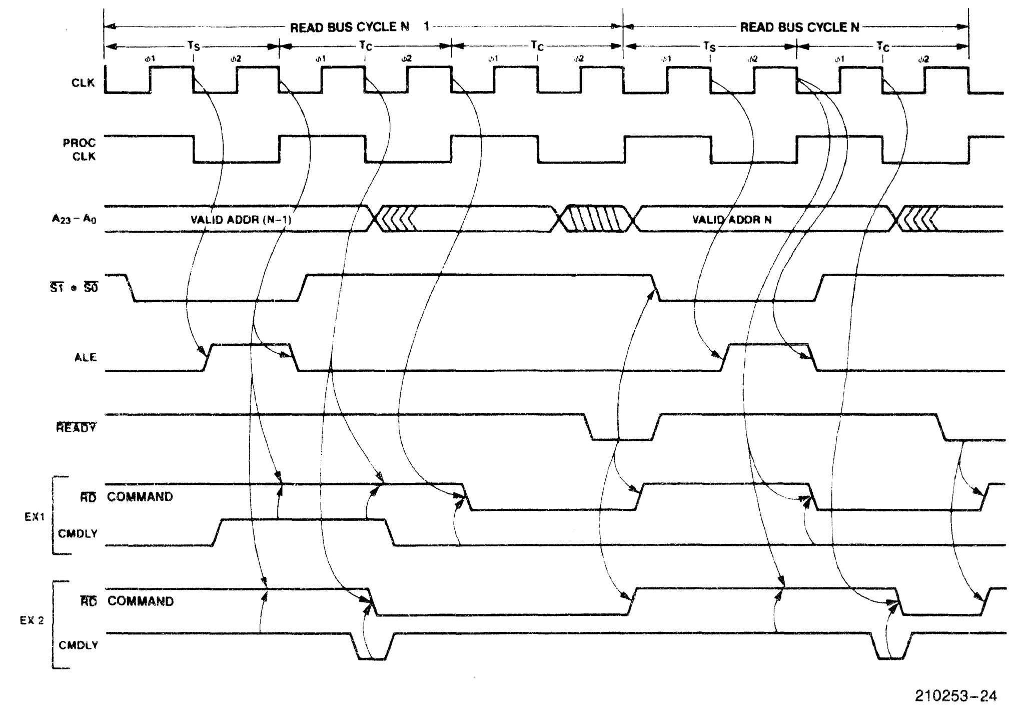


Figure 24. CMDLY Controls the Leading Edge of Command Signal

Figure 24 illustrates four uses of CMDLY. Example 1 shows delaying the read command two system CLKs for cycle N-1 and no delay for cycle N, and example 2 shows delaying the read command one system CLK for cycle N-1 and one system CLK delay for cycle N.

Bus Cycle Termination

At maximum transfer rates, the iAPX 286 bus alternates between the status and command states. The bus status signals become inactive after T_S so that they may correctly signal the start of the next bus operation after the completion of the current cycle. No external indication of T_C exists on the iAPX 286 local bus. The bus master and bus controller enter T_C directly after T_S and continue executing T_C cycles until terminated by $\overline{READY}$.

$\overline{READY}$ Operation

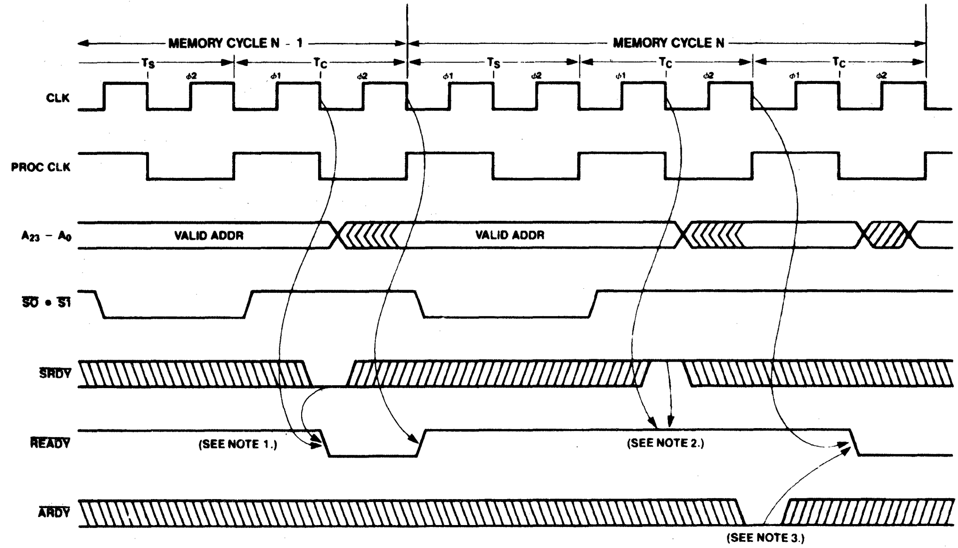
The current bus master and 82288 bus controller terminate each bus operation simultaneously to achieve maximum bus operation bandwidth. Both are informed in advance by $\overline{READY}$ active (open-collector output from 82284) which identifies the last T_C cycle of the current bus operation. The bus master and bus controller must see the same sense of the $\overline{READY}$ signal, thereby requiring $\overline{READY}$ be synchronous to the system clock.

Synchronous Ready

The 82284 clock generator provides $\overline{READY}$ synchronization from both synchronous and asynchronous sources (see Figure 25). The synchronous ready input ($\overline{SRDY}$) of the clock generator is sampled with the falling edge of CLK at the end of phase 1 of each T_C. The state of $\overline{SRDY}$ is then broadcast to the bus master and bus controller via the $\overline{READY}$ output line.

Asynchronous Ready

Many systems have devices or subsystems that are asynchronous to the system clock. As a result, their ready outputs cannot be guaranteed to meet the 82284 $\overline{SRDY}$ setup and hold time requirements. But the 82284 asynchronous ready input ($\overline{ARDY}$) is designed to accept such signals. The $\overline{ARDY}$ input is sampled at the beginning of each T_C cycle by 82284 synchronization logic. This provides one system CLK cycle time to resolve its value before broadcasting it to the bus master and bus controller.



NOTES:
1. $\overline{SRDYEN}$ is active low.
2. If $\overline{SRDYEN}$ is high, the state of $\overline{SRDY}$ will no affect $\overline{READY}$.
3. $\overline{ARDYEN}$ is active low.

Figure 25. Synchronous and Asynchronous Ready

$\overline{ARDY}$ or $\overline{ARDYEN}$ must be HIGH at the end of T_S. $\overline{ARDY}$ cannot be used to terminate bus cycle with no wait states.

Each ready input of the 82284 has an enable pin ($\overline{SRDYEN}$ and $\overline{ARDYEN}$) to select whether the current bus operation will be terminated by the synchronous or asynchronous ready. Either of the ready inputs may terminate a bus operation. These enable inputs are active low and have the same timing as their respective ready inputs. Address decode logic usually selects whether the current bus operation should be terminated by $\overline{ARDY}$ or $\overline{SRDY}$.

Data Bus Control

Figures 26, 27, and 28 show how the DT/$\overline{R}$, DEN, data bus, and address signals operate for different combinations of read, write, and idle bus operations. DT/$\overline{R}$ goes active (LOW) for a read operation. DT/$\overline{R}$ remains HIGH before, during, and between write operations.

The data bus is driven with write data during the second phase of T_S. The delay in write data timing allows the read data drivers, from a previous read cycle, sufficient time to enter 3-state OFF before the 80286 CPU begins driving the local data bus for write operations. Write data will always remain valid for one system clock past the last T_C to provide sufficient hold time for Multibus or other similar memory or I/O systems. During write-read or write-idle sequences the data bus enters 3-state OFF during the second phase of the processor cycle after the last T_C. In a write-write sequence the data bus does not enter 3-state OFF between T_C and T_S.

Bus Usage

The 80286 local bus may be used for several functions: instruction data transfers, data transfers by other bus masters, instruction fetching, processor extension data transfers, interrupt acknowledge, and halt/shutdown. This section describes local bus activities which have special signals or requirements.

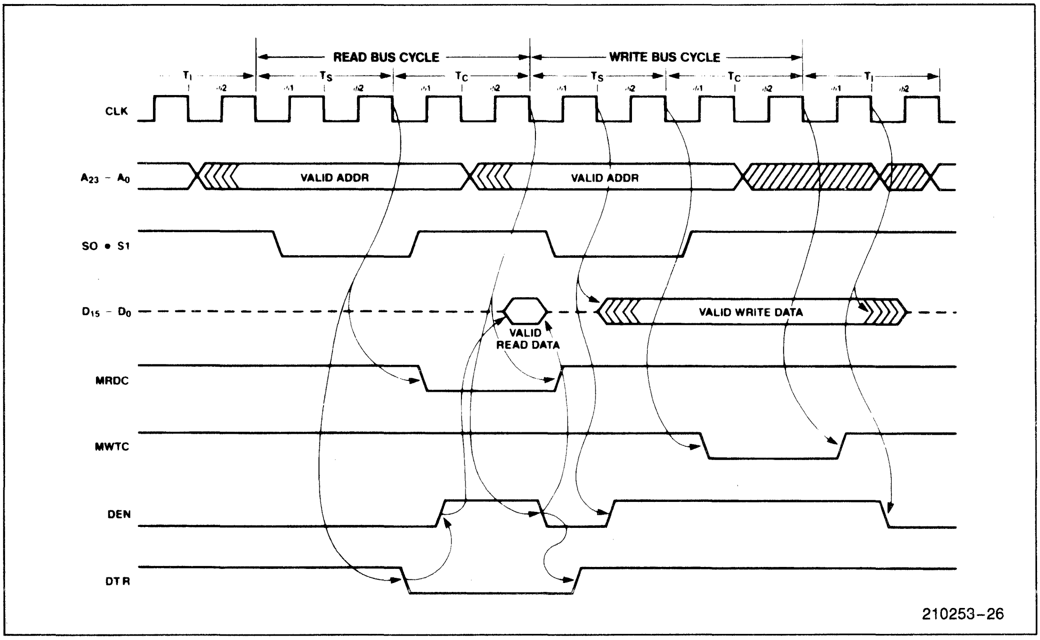


Figure 26. Back to Back Read-Write Cycles

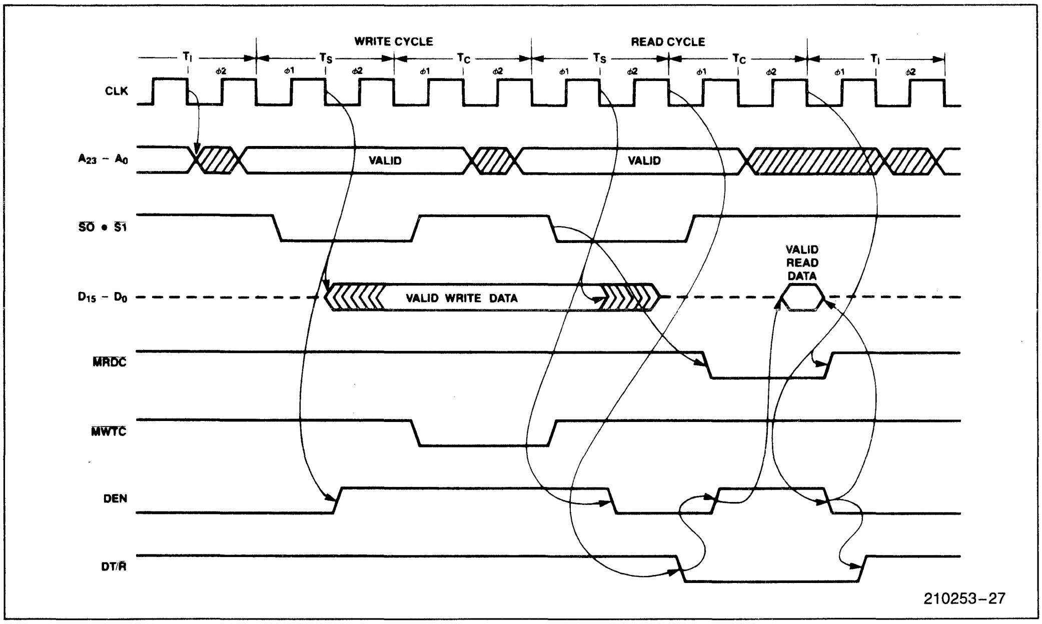


Figure 27. Back to Back Write-Read Cycles

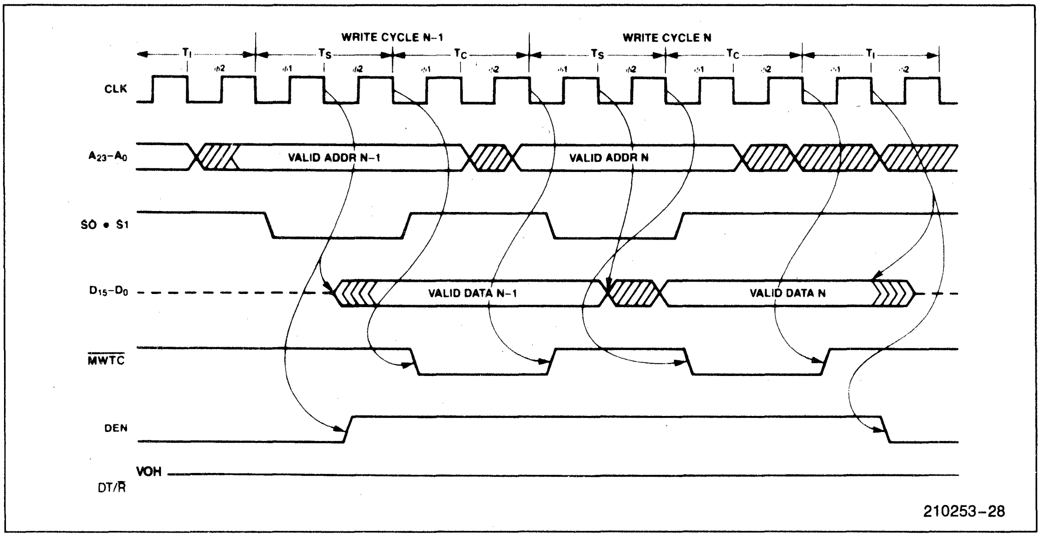


Figure 28. Back to Back Write-Write Cycles

HOLD and HLDA

HOLD AND HLDA allow another bus master to gain control of the local bus by placing the 80286 bus into the T_h state. The sequence of events required to pass control between the 80286 and another local bus master are shown in Figure 29.

In this example, the 80286 is initially in the T_h state as signaled by HLDA being active. Upon leaving T_h, as signaled by HLDA going inactive, a write operation is started. During the write operation another local bus master requests the local bus from the 80286 as shown by the HOLD signal. After completing the write operation, the 80286 performs one T_i bus cycle, to guarantee write data hold time, then enters T_h as signaled by HLDA going active.

The CMDLY signal and $\overline{\text{ARDY}}$ ready are used to start and stop the write bus command, respectively. Note that $\overline{\text{SRDY}}$ must be inactive or disabled by $\overline{\text{SRDYEN}}$ to guarantee $\overline{\text{ARDY}}$ will terminate the cycle.

HOLD must not be active during the time from the leading edge of RESET until 34 CLKs following the trailing edge of RESET unless the 80286 is in the Halt condition. To insure that the 80286 remains in the Halt condition until the processor Reset operation is complete, no interrrupts should occur after the execution of HLT until 34 CLKs after the trailing edge of the RESET pulse.

Lock

The CPU asserts an active lock signal during Interrupt-Acknowledge cycles, the XCHG instruction, and during some descriptor accesses. Lock is also asserted when the LOCK prefix is used. The LOCK prefix may be used with the following ASM-286 assembly instructions; MOVS, INS, and OUTS. For bus cycles other than Interrupt-Acknowledge cycles, Lock will be active for the first and subsequent cycles of a series of cycles to be locked. Lock will not be shown active during the last cycle to be locked. For the next-to-last cycle, Lock will become inactive at the end of the first T_C regardless of the number of wait-states inserted. For Interrupt-Acknowledge cycles, Lock will be active for each cycle, and will become inactive at the end of the first T_C for each cycle regardless of the number of wait-states inserted.

Instruction Fetching

The 80286 Bus Unit (BU) will fetch instructions ahead of the current instruction being executed. This activity is called prefetching. It occurs when the local bus would otherwise be idle and obeys the following rules:

A prefetch bus operation starts when at least two bytes of the 6-byte prefetch queue are empty.

The prefetcher normally performs word prefetches independent of the byte alignment of the code segment base in physical memory.

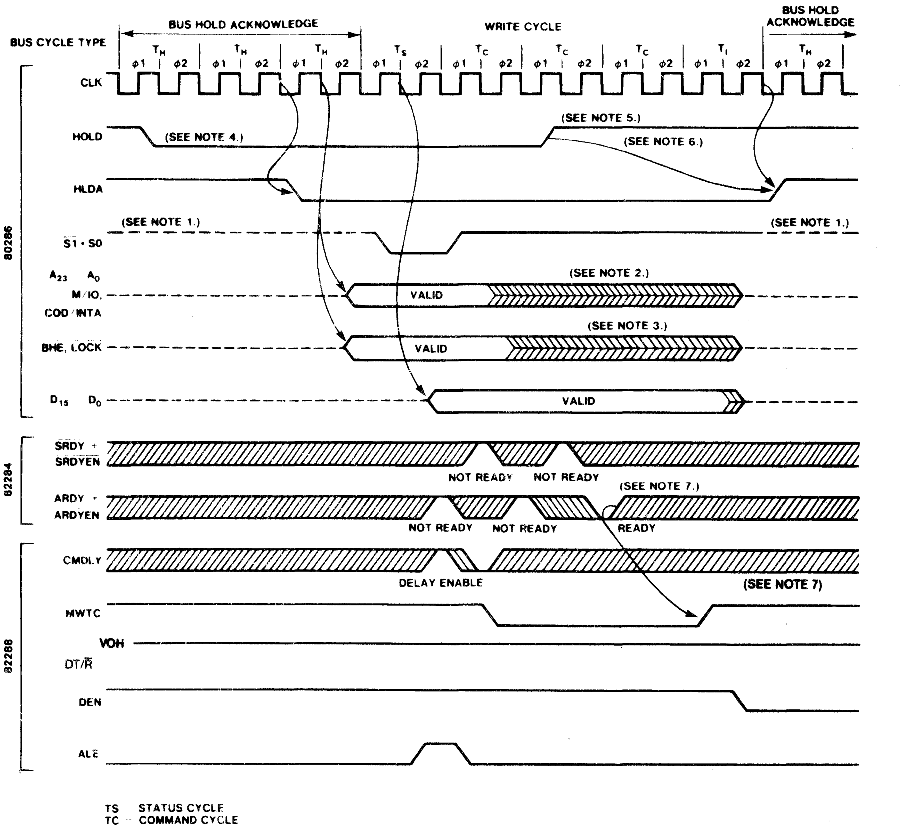
The prefetcher will perform only a byte code fetch operation for control transfers to an instruction beginning on a numerically odd physical address.

Prefetching stops whenever a control transfer or HLT instruction is decoded by the IU and placed into the instruction queue.

In real address mode, the prefetcher may fetch up to 6 bytes beyond the last control transfer or HLT instruction in a code segment.

In protected mode, the prefetcher will never cause a segment overrun exception. The prefetcher stops at the last physical memory word of the code segment. Exception 13 will occur if the program attempts to execute beyond the last full instruction in the code segment.

If the last byte of a code segment appears on an even physical memory address, the prefetcher will read the next physical byte of memory (perform a word code fetch). The value of this byte is ignored and any attempt to execute it causes exception 13.



NOTES:

1. Status lines are not driven by 80286, yet remain high due to pullup resistors in 82288 and 82289 during HOLD state.
2. Address, M/$\overline{IO}$ and COD/$\overline{INTA}$ may start floating during any T_C depending on when internal 80286 bus arbiter decides to release bus to external HOLD. The float starts in $\phi 2$ of T_C.
3. $\overline{BHE}$ and $\overline{LOCK}$ may start floating after the end of any T_C depending on when internal 80286 bus arbiter decides to release bus to external HOLD. The float starts in $\phi 1$ of T_C.
4. The minimum HOLD to HLDA time is shown. Maximum is one T_H longer.
5. The earliest HOLD time is shown. It will always allow a subsequent memory cycle if pending is shown.
6. The minimum HOLD to HLDA time is shown. Maximum is a function of the instruction, type of bus cycle and other machine state (i.e., Interrupts, Waits, Lock, etc.).
7. Asynchronous ready allows termination of the cycle. Synchronous ready does not signal ready in this example. Synchronous ready state is ignored after ready is signaled via the asynchronous input.

Figure 29. Multibus Write Terminated by Asynchronous Ready with Bus Hold

Processor Extension Transfers

The processor extension interface uses I/O port addresses 00F8(H), 00FA(H), and 00FC(H) which are part of the I/O port address range reserved by Intel. An ESC instruction with Machine Status Word bits EM = 0 and TS = 0 will perform I/O bus operations to one or more of these I/O port addresses independent of the value of IOPL and CPL.

ESC instructions with memory references enable the CPU to accept PEREQ inputs for processor extension operand transfers. The CPU will determine the operand starting address and read/write status of the instruction. For each operand transfer, two or three bus operations are performed, one word transfer with I/O port address 00FA(H) and one or two bus operations with memory. Three bus operations are required for each word operand aligned on an odd byte address.

Interrupt Acknowledge Sequence

Figure 30 illustrates an interrupt acknowledge sequence performed by the 80286 in response to an INTR input. An interrupt acknowledge sequence consists of two INTA bus operations. The first allows a master 8259A Programmable Interrupt Controller (PIC) to determine which if any of its slaves should return the interrupt vector. An eight bit vector is read on D0–D7 of the 80286 during the second INTA bus operation to select an interrupt handler routine from the interrupt table.

The Master Cascade Enable (MCE) signal of the 82288 is used to enable the cascade address drivers, during INTA bus operations (See Figure 30), onto the local address bus for distribution to slave interrupt controllers via the system address bus. The 80286 emits the $\overline{LOCK}$ signal (active LOW) during T_s of the first INTA bus operation. A local bus "hold" request will not be honored until the end of the second INTA bus operation.

Three idle processor clocks are provided by the 80286 between INTA bus operations to allow for the minimum INTA to INTA time and CAS (cascade address) out delay of the 8259A. The second INTA bus operation must always have at least one extra T_c state added via logic controlling $\overline{READY}$. A_{23}–A_0 are in 3-state OFF until after the first T_c state of the second INTA bus operation. This prevents bus contention between the cascade address drivers and CPU address drivers. The extra T_c state allows time for the 80286 to resume driving the address lines for subsequent bus operations.

Local Bus Usage Priorities

The 80286 local bus is shared among several internal units and external HOLD requests. In case of simultaneous requests, their relative priorities are:

(Highest) Any transfers which assert $\overline{LOCK}$ either explicitly (via the LOCK instruction prefix) or implicitly (i.e. some segment descriptor accesses, interrupt acknowledge sequence, or an XCHG with memory).

The second of the two byte bus operations required for an odd aligned word operand.

The second or third cycle of a processor extension data transfer.

Local bus request via HOLD input.

Processor extension data operand transfer via PEREQ input.

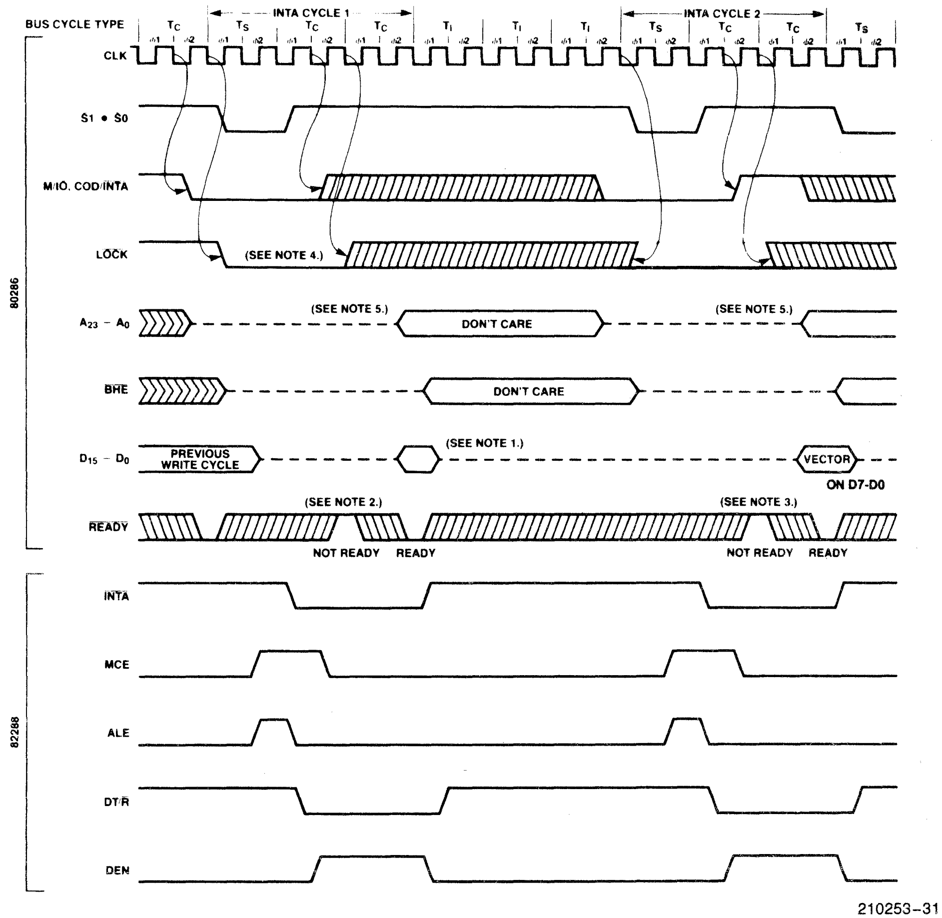
Data transfer performed by EU as part of an instruction.

(Lowest) An instruction prefetch request from BU. The EU will inhibit prefetching two processor clocks in advance of any data transfers to minimize waiting by EU for a prefetch to finish.

Halt or Shutdown Cycles

The 80286 externally indicates halt or shutdown conditions as a bus operation. These conditions occur due to a HLT instruction or multiple protection exceptions while attempting to execute one instruction. A halt or shutdown bus operation is signalled when $\overline{S1}$, $\overline{S0}$ and COD/$\overline{INTA}$ are LOW and M/$\overline{IO}$ is HIGH. A_1 HIGH indicates halt, and A_1 LOW indicates shutdown. The 82288 bus controller does not issue ALE, nor is $\overline{READY}$ required to terminate a halt or shutdown bus operation.

During halt or shutdown, the 80286 may service PEREQ or HOLD requests. A processor extension segment overrun exception during shutdown will inhibit further service of PEREQ. Either NMI or RESET will force the 80286 out of either halt or shutdown. An INTR, if interrupts are enabled, or a processor extension segment overrun exception will also force the 80286 out of halt.



NOTES:
1. Data is ignored.
2. First INTA cycle should have at least one wait state inserted to meet 8259A minimum INTA pulse width.
3. Second INTA cycle must have at least one wait state inserted since the CPU will not drive A_{23}–A_0, $\overline{BHE}$, and $\overline{LOCK}$ until afer the first TC state. The CPU imposed one/clock delay prevents bus contention between cascade address buffer being disabled by MCE ↓ and address outputs. Without the wait state, the 80286 address will not be valid for a memory cycle started immediately after the second INTA cycle. The 8259A also requires one wait state for minimum INTA pulse width.
4. $\overline{LOCK}$ is active for the first INTA cycle to prevent the 82289 from releasing the bus between INTA cycles in a multi-master system. $\overline{LOCK}$ is also active for the second INTA cycle.
5. A_{23}–A_0 exits 3-state OFF during $\phi 2$ of the second T_C in the INTA cycle.

Figure 30. Interrupt Acknowledge Sequence

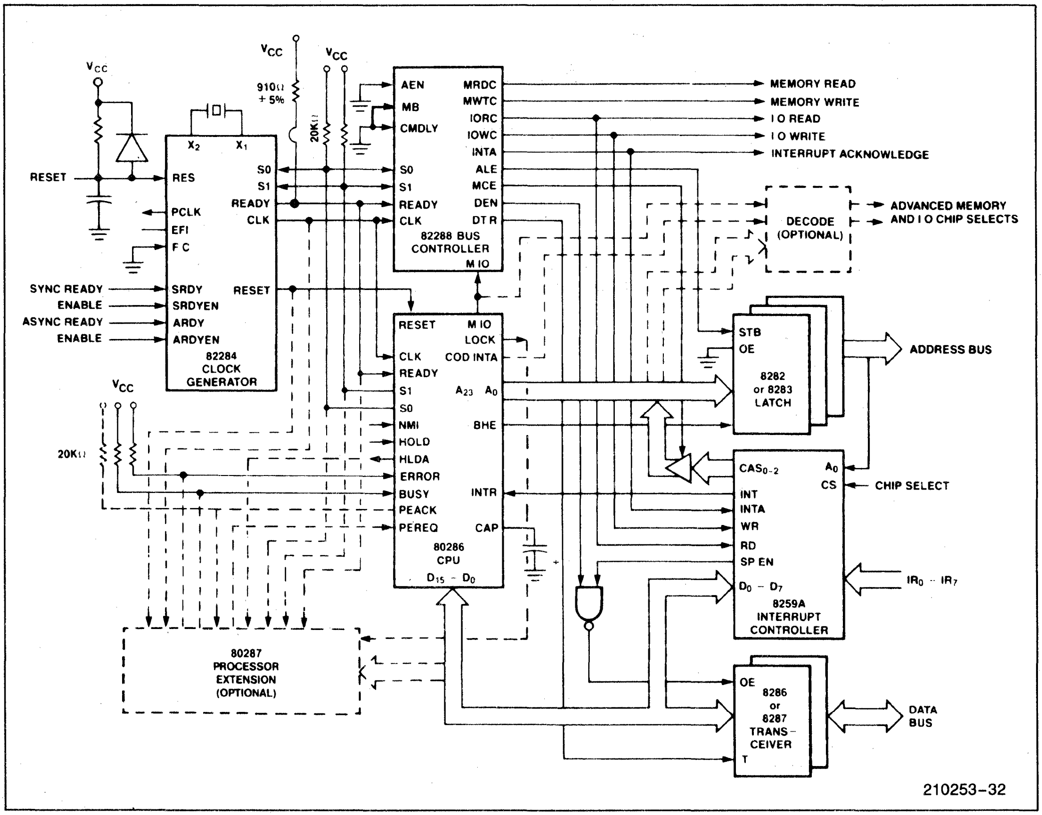


Figure 31. Basic iAPX 286 System Configuration

SYSTEM CONFIGURATIONS

The versatile bus structure of the iAPX 286 micro-system, with a full complement of support chips, allows flexible configuration of a wide range of systems. The basic configuration, shown in Figure 31, is similar to an iAPX 86 maximum mode system. It includes the CPU plus an 8259A interrupt controller, 82284 clock generator, and the 82288 Bus Controller. The iAPX 86 latches (8282 and 8283) and transceivers (8286 and 8287) may be used in an iAPX 286 microsystem.

As indicated by the dashed lines in Figure 31, the ability to add processor extensions is an integral feature of iAPX 286 microsystems. The processor extension interface allows external hardware to perform special functions and transfer data concurrent with CPU execution of other instructions. Full system integrity is maintained because the 80286 supervises all data transfers and instruction execution for the processor extension.

The iAPX 286/20 numeric data processor which includes the 80287 numeric processor extension (NPX) uses this interface. The iAPX 286/20 has all the instructions and data types of an iAPX 86/20 or iAPX 88/20. The 80287 NPX can perform numeric calculations and data transfers concurrently with CPU program execution. Numerics code and data have the same integrity as all other information protected by the iAPX 286 protection mechanism.

The 80286 can overlap chip select decoding and address propagation during the data transfer for the previous bus operation. This information is latched into the 8282/3's by ALE during the middle of a T_S cycle. The latched chip select and address information remains stable during the bus operation while the next cycle's address is being decoded and propagated into the system. Decode logic can be implemented with a high speed bipolar PROM.

The optional decode logic shown in Figure 31 takes advantage of the overlap between address and data of the 80286 bus cycle to generate advanced memory and IO-select signals. This minimizes system

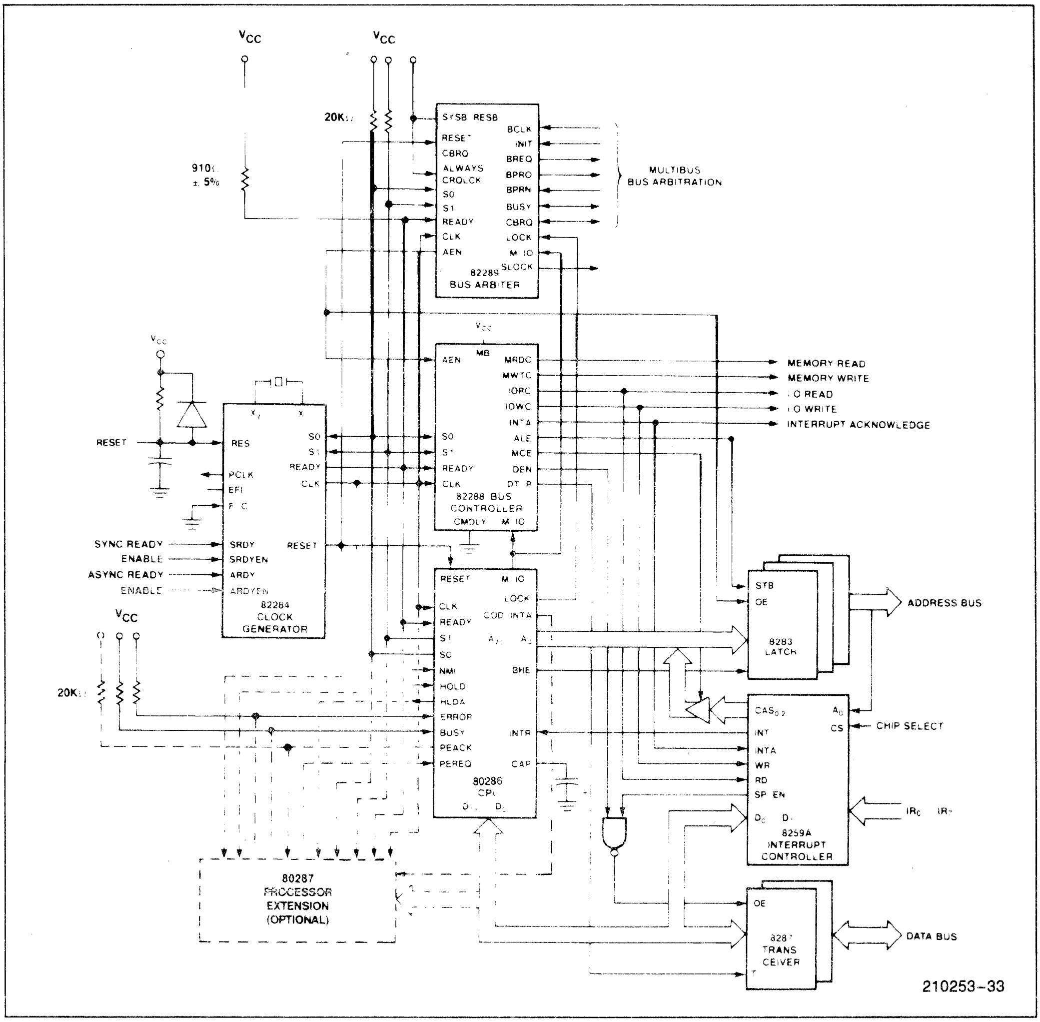
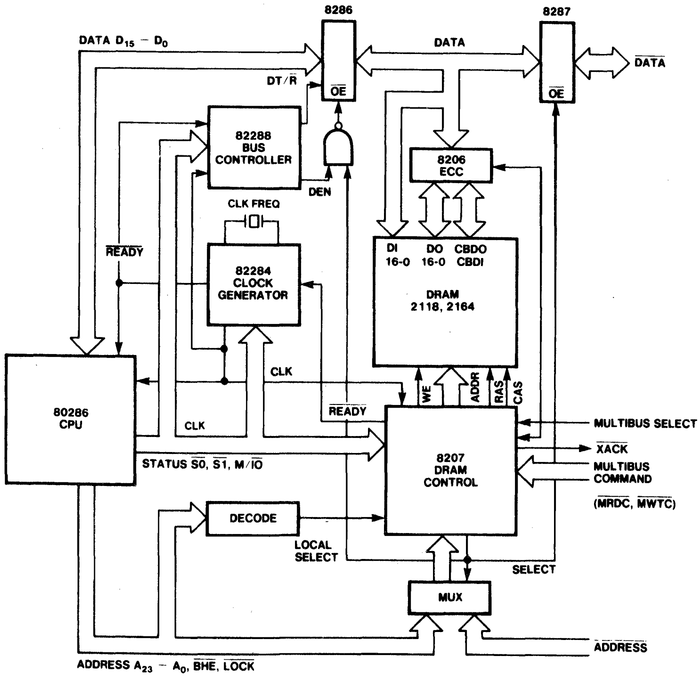


Figure 32. MULTIBUS® System Bus Interface

performance degradation caused by address propagation and decode delays. In addition to selecting memory and I/O, the advanced selects may be used with configurations supporting local and system buses to enable the appropriate bus interface for each bus cycle. The COD/$\overline{\text{INTA}}$ and M/$\overline{\text{IO}}$ signals are applied to the decode logic to distinguish between interrupt, I/O, code and data bus cycles.

By adding the 82289 bus arbiter chip the 80286 provides a MULTIBUS system bus interface as shown in Figure 32. The ALE output of the 82288 for the MULTIBUS bus is connected to its CMDLY input to delay the start of commands one system CLK as required to meet MULTIBUS address and write data setup times. This arrangement will add at least one extra T_C state to each bus operation which uses the MULTIBUS.

A second 82288 bus controller and additional latches and transceivers could be added to the local bus of Figure 32. This configuration allows the 80286 to support an on-board bus for local memory and peripherals, and the MULTIBUS for system bus interfacing.



210253-34

Figure 33. iAPX 286 System Configuration with Dual-Ported Memory

Figure 33 shows the addition of dual ported dynamic memory between the MULTIBUS system bus and the iAPX 286 local bus. The dual port interface is provided by the 8207 Dual Port DRAM Controller. The 8207 runs synchronously with the CPU to maximize throughput for local memory references. It also arbitrates between requests from the local and system buses and performs functions such as refresh, initialization of RAM, and read/modify/write cycles. The 8207 combined with the 8206 Error Checking and Correction memory controller provide for single bit error correction. The dual-ported memory can be combined with a standard MULTIBUS system bus interface to maximize performance and protection in multiprocessor system configurations.

Table 16. 80286 Systems Recommended Pull Up Resistor Values

80286 Pin and Name	Pullup Value	Purpose
4—$\overline{S1}$	20 KΩ ±10%	Pull $\overline{S0}$, $\overline{S1}$, and $\overline{PEACK}$ inactive during 80286 hold periods
5—$\overline{S0}$		
6—$\overline{PEACK}$		
53—$\overline{ERROR}$	20 KΩ ±10%	Pull $\overline{ERROR}$ and $\overline{BUSY}$ inactive when 80287 not present (or temporarily removed from socket)
54—$\overline{BUSY}$		
63—$\overline{READY}$	910Ω ±5%	Pull $\overline{READY}$ inactive within required minimum time (C_L = 150 pF, $I_R \leq$ 7 mA)

I²ICE™-286 System Design Considerations

One of the advantages of using the 80286 is that full in-circuit emulation debugging support is provided through the I²ICE system 80286 probe. To utilize this powerful tool it is necessary that the system designer be aware of a few minor parametric and functional differences between the 80286 and I²ICE system 80286 probe. The I²ICE data sheet (I²ICE Integrated Instrumentation and In-Circuit Emulation System, order #210469) contains a detailed description of these design considerations. It is recommended that this document be reviewed by the 80286 system designer to determine whether or not these differences affect his design.

PACKAGE

The 80286 is packaged in a 68-pin, leadless JEDEC type A hermetic leadless chip carrier (LCC) and 68-pin pin grid array (PGA). Figure 34 illustrates the packages, and Figure 2 shows the pinout.

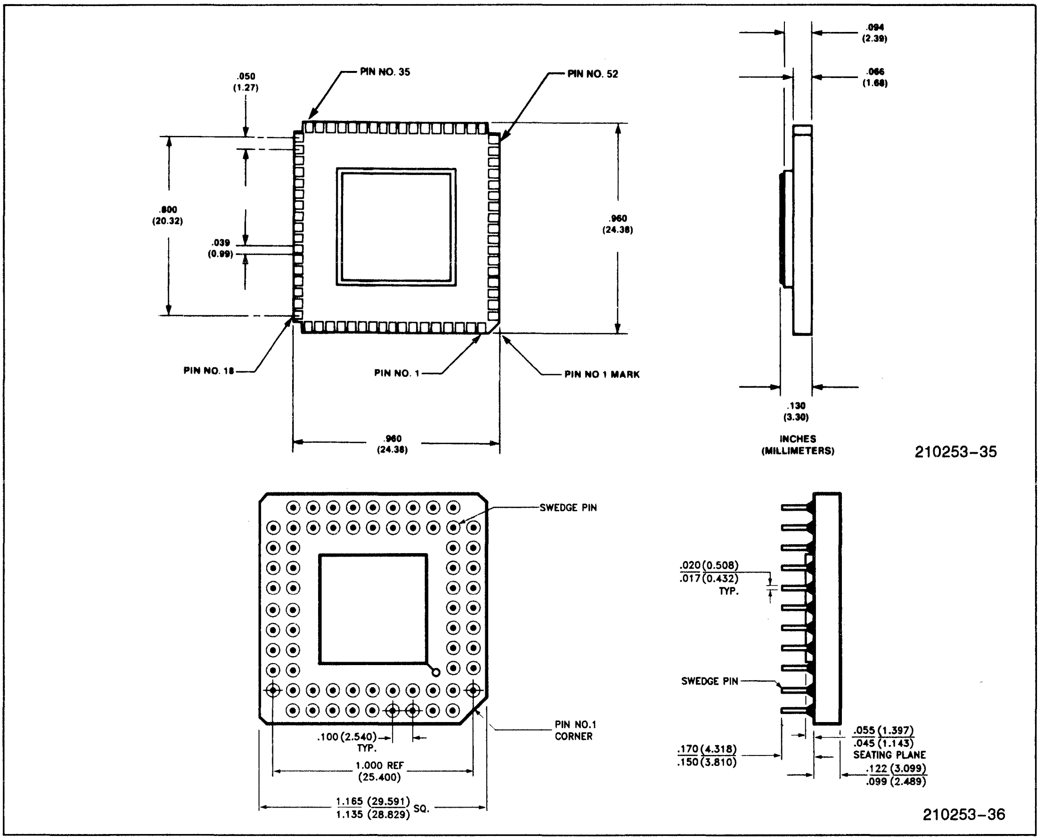


Figure 34. JEDEC Type A Package (Top) and Pin Grid Array Package

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ....0°C to +70°C
Storage Temperature .... −65°C to +150°C
Voltage on Any Pin with
Respect to Ground .... −1.0V to +7V
Power Dissipation 3.3W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (V_{CC} = 5V ±5%, T_A = 0°C to +55°C, or T_{CASE} = 0°C to +85°C)

Symbol	Parameter	6 MHz		8 MHz		10 MHz		Unit	Test Condition
		-6 Min	-6 Max	-8 Min	-8 Max	-10 Min	-10 Max		
V_{IL}	Input LOW Voltage	−.5	.8	−.5	.8	−.5	.8	V	
V_{IH}	Input HIGH Voltage	2.0	V_{CC} + .5	2.0	V_{CC} + .5	2.0	V_{CC} + .5	V	
V_{ILC}	CLK Input LOW Voltage	−.5	.6	−.5	.6	−.5	.6	V	
V_{IHC}	CLK Input HIGH Voltage	3.8	V_{CC} + .5	3.8	V_{CC} + .5	3.8	V_{CC} + .5	V	

D.C. CHARACTERISTICS (V_{CC} = 5V ±5%, T_A = 0°C to +55°C, or T_{CASE} = 0°C to +85°C)

Symbol	Parameter	6 MHz		8 MHz		10 MHz		Unit	Test Condition
		-6 Min	-6 Max	-8 Min	-8 Max	-10 Min	-10 Max		
V_{OL}	Output LOW Voltage		0.45		0.45		0.45	V	I_{OL} = 2.0 mA
V_{OH}	Output HIGH Voltage	2.4		2.4		2.4		V	I_{OH} = −400 μA
I_{LI}	Input Leakage Current		±10		±10		±10	μA	0V ≤ V_{IN} ≤ V_{CC}
I_{LCR}	Input Leakage Current		±10		±10		±10	μA	0.45 ≤ V_{IN} ≤ V_{CC}
I_{LCR}	Input Leakage Current		±1		±1		±1	mA	0V ≤ V_{IN} ≤ 0.45V
I_{IL}	Input Sustaining Current on $\overline{BUSY}$ and $\overline{ERROR}$ Pins	30	500	30	500	30	500	μA	V_{IN} = 0V
I_{LO}	Output Leakage Current		±10		±10		±10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}
I_{LO}	Output Leakage Current		±1		±1		±1	mA	0V ≤ V_{OUT} < 0.45V
I_{CC}	Supply Current (turn on, 0°C)		600		600		600	mA	Note 1
C_{CLK}	CLK Input Capacitance		20		20		20	pF	F_C = 1 MHz
C_{IN}	Other Input Capacitance		10		10		10	pF	F_C = 1 MHz
C_O	Input/Output Capacitance		20		20		20	pF	F_C = 1 MHz

A.C. CHARACTERISTICS (V_{CC} = 5V ±5%, T_A = 0°C to +55°C, or T_{CASE} = 0°C to +85°C)

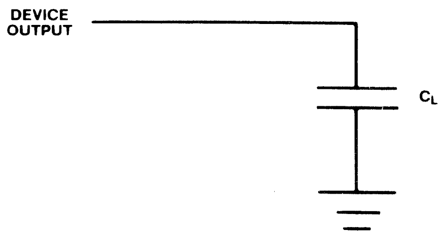
AC timings are referenced to 0.8V and 2.0V points of signals as illustrated in datasheet waveforms, unless otherwise noted.

Symbol	Parameter	6 MHz		8 MHz		10 MHz (Preliminary)		Unit	Test Condition
		-6 Min	-6 Max	-8 Min	-8 Max	-10 Min	-10 Max		
1	System Clock (CLK) Period	83	250	62	250	50	250	ns	
2	System Clock (CLK) LOW Time	20	225	15	225	12	234	ns	at 1.0V
3	System Clock (CLK) HIGH Time	25	230	25	235	16	238	ns	at 3.6V
17	System Clock (CLK) Rise Time		10		10		8	ns	1.0V to 3.6V
18	System Clock (CLK) Fall Time		10		10		8	ns	3.6V to 1.0V
4	Asynch. Inputs Setup Time	30		20		20		ns	Note 1
5	Asynch. Inputs Hold Time	30		20		20		ns	Note 1
6	RESET Setup Time	33		28		23		ns	
7	RESET Hold Time	5		5		5		ns	
8	Read Data Setup Time	20		10		8		ns	
9	Read Data Hold Time	8		8		8		ns	
10	$\overline{READY}$ Setup Time	50		38		26		ns	
11	$\overline{READY}$ Hold Time	35		25		25		ns	
12	Status/$\overline{PEACK}$ Valid Delay	1	55	1	40	—	—	ns	Note 2 Note 3
12a	Status/$\overline{PEACK}$ Active Delay	—	—	—	—	1	28	ns	Note 2 Note 3
12b	Status/$\overline{PEACK}$ Inactive Delay	—	—	—	—	1	30	ns	Note 2 Note 3
13	Address Valid Delay	1	80	1	60	1	47	ns	Note 2 Note 3
14	Write Data Valid Delay	0	65	0	50	0	40	ns	Note 2 Note 3
15	Address/Status/Data Float Delay	0	80	0	50	0	47	ns	Note 2 Note 4
16	HLDA Valid Delay	0	80	0	50	0	47	ns	Note 2 Note 3
19	Address Valid To Status Valid Setup Time	—		38		27		ns	Note 3 Note 5 Note 6

NOTES:

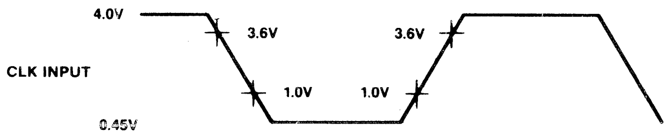
1. Asynchronous inputs are INTR, NMI, HOLD, PEREQ, $\overline{ERROR}$, and $\overline{BUSY}$. This specification is given only for testing purposes, to assure recognition at a specific CLK edge.
2. Delay from 0.8V on the CLK, to 0.8V or 2.0V or float on the output as appropriate for valid or floating condition.
3. Output load: C_L = 100 pF.
4. Float condition occurs when output current is less than I_{LO} in magnitude.
5. Delay measured from address either reaching 0.8V or 2.0V (valid) to status going active reaching 2.0V or status going inactive reaching 0.8V.
6. For load capacitance of 10 pF on STATUS/$\overline{PEACK}$ lines, subtract typically 7 ns for 8 MHz spec, and maximum 7 ns for 10 MHz spec.

A.C. CHARACTERISTICS (Continued)



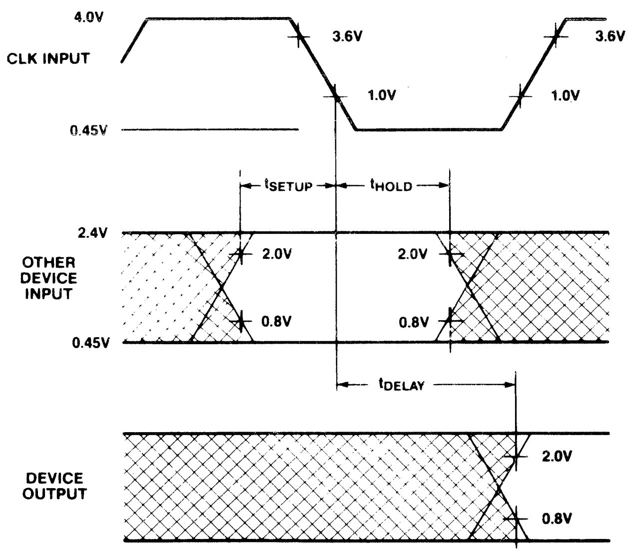
210253-37

NOTE 7:
AC Test Loading on Outputs



210253-38

NOTE 8:
AC Drive and Measurement Points—CLK Input



210253-39

NOTE 9:
AC Setup, Hold and Delay Time Measurement—General

A.C. CHARACTERISTICS (Continued)

82284 Timing Requirements

		82284-6		82284-8		82284-10 (Preliminary)		Units	Test Conditions
Symbol	Parameter	Min	Max	Min	Max	Min	Max		
11	$\overline{\text{SRDY}}$/$\overline{\text{SRDYEN}}$ Setup Time	25		17		15		ns	
12	$\overline{\text{SRDY}}$/$\overline{\text{SRDYEN}}$ Hold Time	0		0		0		ns	
13	$\overline{\text{ARDY}}$/$\overline{\text{ARDYEN}}$ Setup Time	5		0		0		ns	(Note 1)
14	$\overline{\text{ARDY}}$/$\overline{\text{ARDYEN}}$ Hold Time	30		30		30		ns	(Note 1)
19	PCLK Delay	0	45	0	45	0	35	ns	$C_L = 75$ pF $I_{OL} = 5$ mA $I_{OH} = -1$ mA

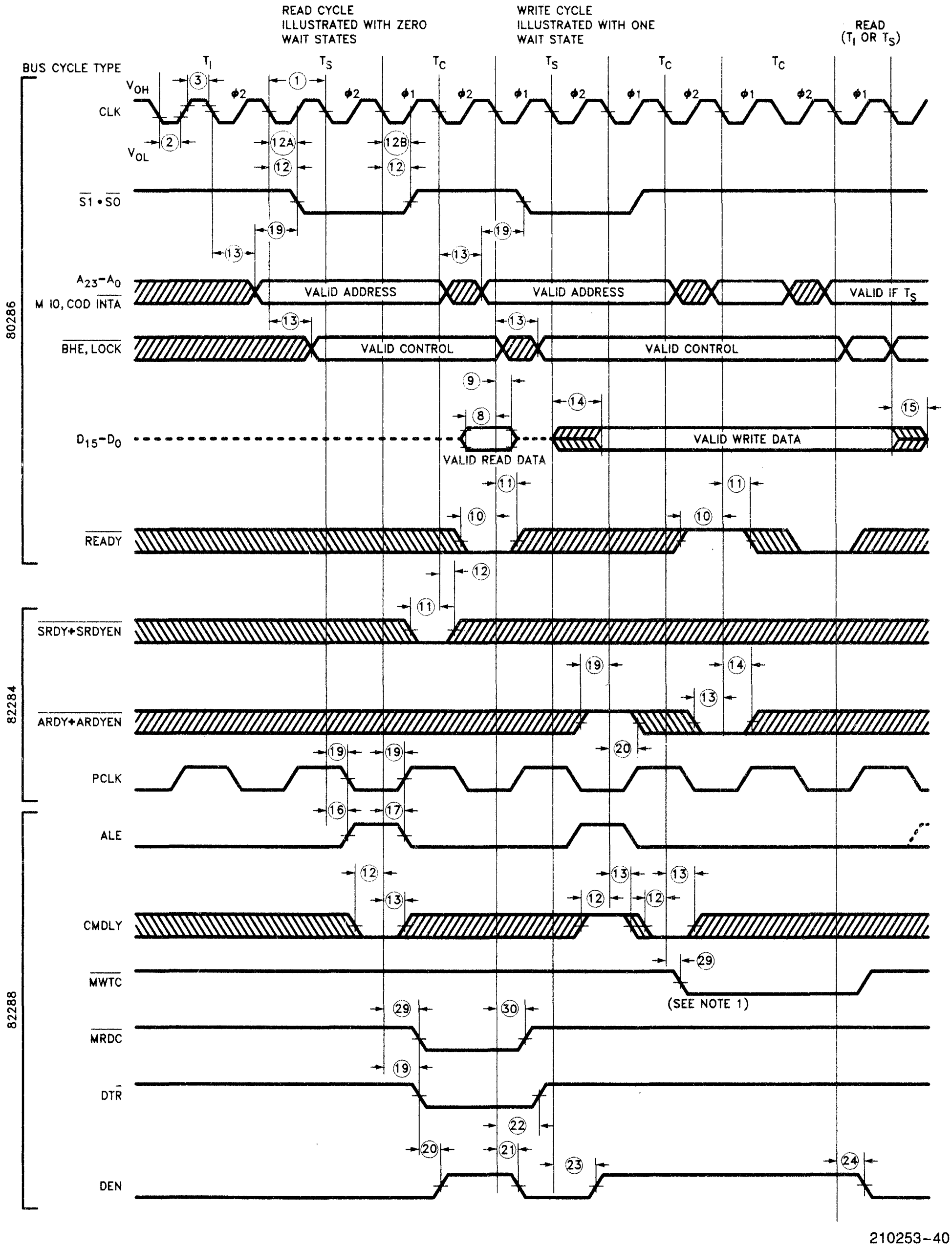
NOTE 1:
These times are given for testing purposes to assure a predetermined action.

82288 Timing Requirements

			82288-6		82288-8		82288-10 (Preliminary)		Units	Test Conditions
Symbol	Parameter		Min	Max	Min	Max	Min	Max		
12	CMDLY Setup Time		25		20		15		ns	
13	CMDLY Hold Time		1		1		1		ns	
30	Command Delay from CLK	Command Inactive	5	30	5	25	5	20	ns	$C_L = 300$ pF max $I_{OL} = 32$ mA max $I_{OH} = 5$ mA max
29		Command Active	3	40	3	25	3	21		
16	ALE Active Delay		3	25	3	20	3	16	ns	$C_L = 150$ pF $I_{OL} = 16$ mA max $I_{OH} = -1$ mA max
17	ALE Inactive Delay			35		25		19	ns	
19	DT/$\overline{\text{R}}$ Read Active Delay			40		25		23	ns	
22	DT/$\overline{\text{R}}$ Read Inactive Delay		5	45	5	35	5	20	ns	
20	DEN Read Active Delay		5	50	5	35	5	21	ns	
21	DEN Read Inactive Delay		3	40	3	35	3	21	ns	
23	DEN Write Active Delay			35		30		23	ns	
24	DEN Write Inactive Delay		3	35	3	30	3	19	ns	

WAVEFORMS

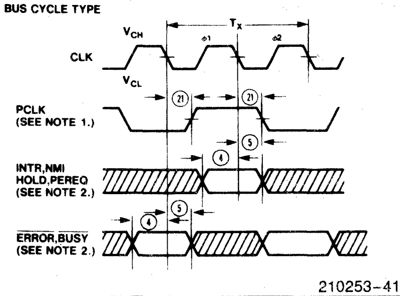
MAJOR CYCLE TIMING



NOTE:
1. The modified timing is due to the CMDLY signal being active.

WAVEFORMS (Continued)

80286 ASYNCHRONOUS INPUT SIGNAL TIMING

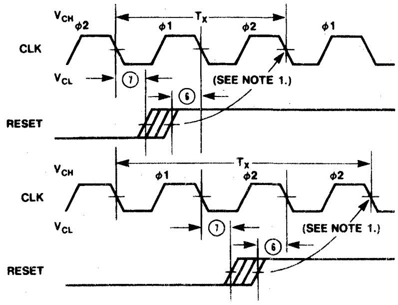


210253-41

NOTES:

1. PCLK indicates which processor cycle phase will occur on the next CLK. PCLK may not indicate the correct phase until the first bus cycle is performed.
2. These inputs are asynchronous. The setup and hold times shown assure recognition for testing purposes.

80286 RESET INPUT TIMING AND SUBSEQUENT PROCESSOR CYCLE PHASE

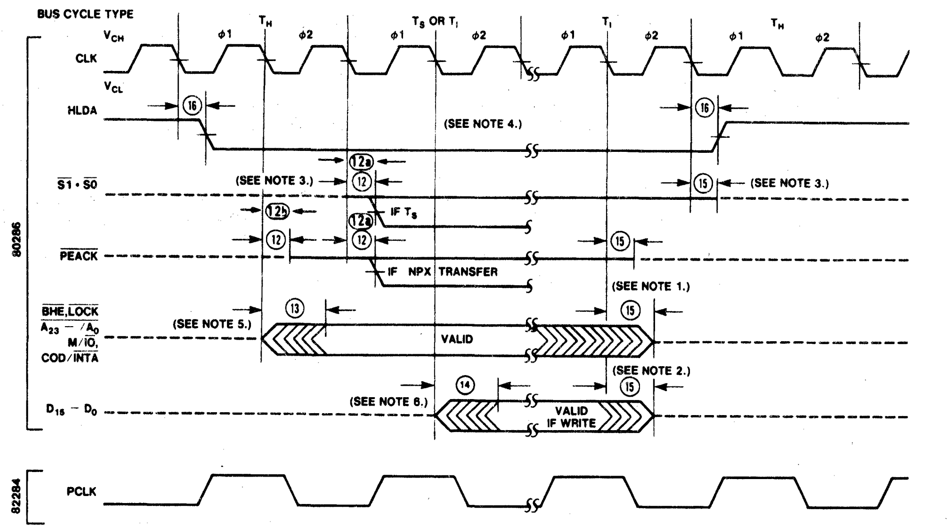


210253-42

NOTE:

When RESET meets the setup time shown, the next CLK will start or repeat $\phi 2$ of a processor cycle.

EXITING AND ENTERING HOLD



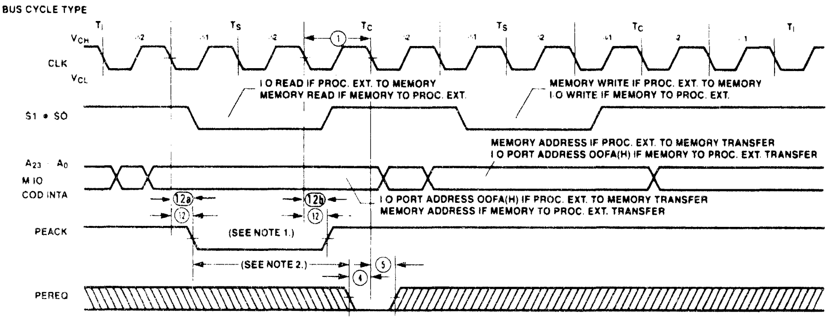
210253-43

NOTES:

1. These signals may not be driven by the 80286 during the time shown. The worst case in terms of latest float time is shown.
2. The data bus will be driven as shown if the last cycle before T_I in the diagram was a write T_C.
3. The 80286 floats its status pins during T_H. External 20 KΩ resistors keep these signals high (see Table 16).
4. For HOLD request set up to HLDA, refer to Figure 29.
5. $\overline{BHE}$ and $\overline{LOCK}$ are driven at this time but will not become valid until T_S.
6. The data bus will remain in 3-state OFF if a read cycle is performed.

WAVEFORMS (Continued)

80286 PEREQ/$\overline{\text{PEACK}}$ TIMING FOR ONE TRANSFER ONLY



ASSUMING WORD-ALIGNED MEMORY OPERAND. IF ODD ALIGNED, 80286 TRANSFERS TO/FROM MEMORY BYTE-AT-A-TIME WITH TWO MEMORY CYCLES.

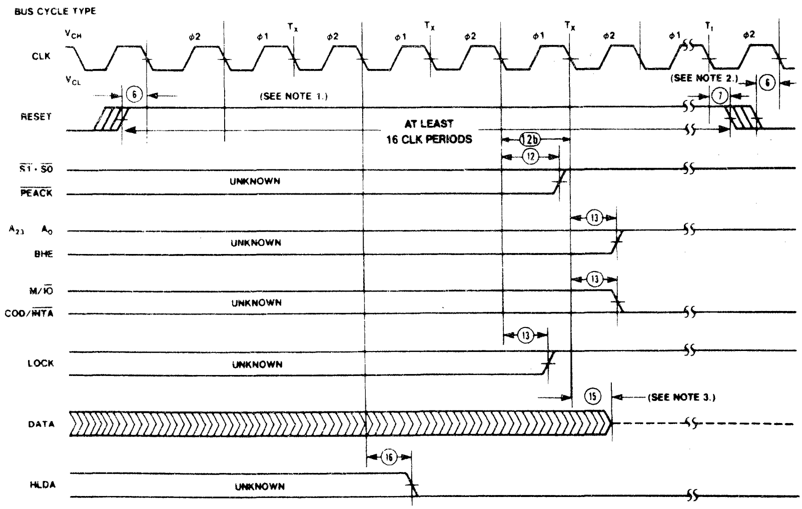
210253–44

NOTES:

1. $\overline{\text{PEACK}}$ always goes active during the first bus operation of a processor extension data operand transfer sequence. The first bus operation will be either a memory read at operand address or I/O read at port address OOFA(H).
2. To prevent a second processor extension data operand transfer, the worst case maximum time (Shown above) is: 3×a ① −12a$_{max.}$ − ④$_{min.}$. The actual, configuration dependent, maximum time is: 3× ① −12a$_{max.}$ − ④$_{min.}$ + A×2× ①.

A is the number of extra T_C states added to either the first or second bus operation of the processor extension data operand transfer sequence.

INITIAL 80286 PIN STATE DURING RESET



210253–45

NOTES:

1. Setup time for RESET ↑ may be violated with the consideration that ϕ1 of the processor clock may begin one system CLK period later.
2. Setup and hold times for RESET ↓ must be met for proper operation, but RESET ↓ may occur during ϕ1 or ϕ2.
3. The data bus is only guaranteed to be in 3-state OFF at the time shown.

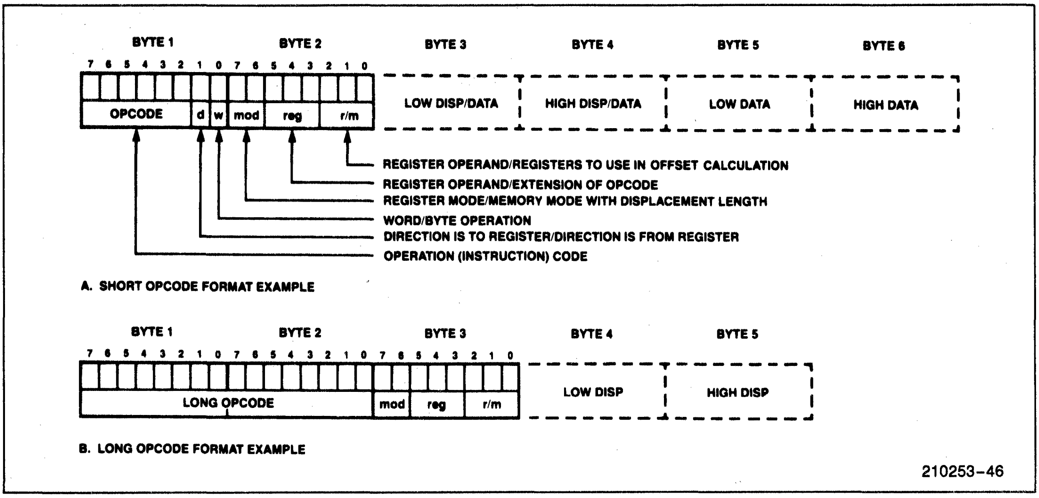


Figure 35. 80286 Instruction Format Examples

80286 INSTRUCTION SET SUMMARY

Instruction Timing Notes

The instruction clock counts listed below establish the maximum execution rate of the 80286. With no delays in bus cycles, the actual clock count of an 80286 program will average 5% more than the calculated clock count, due to instruction sequences which execute faster than they can be fetched from memory.

To calculate elapsed times for instruction sequences, multiply the sum of all instruction clock counts, as listed in the table below, by the processor clock period. An 8 MHz processor clock has a clock period of 125 nanoseconds and requires an 80286 system clock (CLK input) of 16 MHz.

Instruction Clock Count Assumptions

1. The instruction has been prefetched, decoded, and is ready for execution. Control transfer instruction clock counts include all time required to fetch, decode, and prepare the next instruction for execution.
2. Bus cycles do not require wait states.
3. There are no processor extension data transfer or local bus HOLD requests.
4. No exceptions occur during instruction execution.

Instruction Set Summary Notes

Addressing displacements selected by the MOD field are not shown. If necessary they appear after the instruction fields shown.

Above/below refers to unsigned value

Greater refers to positive signed value

Less refers to less positive (more negative) signed values

if d = 1 then to register; if d = 0 then from register

if w = 1 then word instruction; if w = 0 then byte instruction

if s = 0 then 16-bit immediate data form the operand

if s = 1 then an immediate data byte is sign-extended to form the 16-bit operand

x don't care

z used for string primitives for comparison with ZF FLAG

If two clock counts are given, the smaller refers to a register operand and the larger refers to a memory operand

* = add one clock if offset calculation requires summing 3 elements

n = number of times repeated

m = number of bytes of code in next instruction

Level (L)—Lexical nesting level of the procedure

The following comments describe possible exceptions, side effects, and allowed usage for instructions in both operating modes of the 80286.

REAL ADDRESS MODE ONLY

1. This is a protected mode instruction. Attempted execution in real address mode will result in an undefined opcode exception (6).
2. A segment overrun exception (13) will occur if a word operand reference at offset FFFF(H) is attempted.
3. This instruction may be executed in real address mode to initialize the CPU for protected mode.
4. The IOPL and NT fields will remain 0.
5. Processor extension segment overrun interrupt (9) will occur if the operand exceeds the segment limit.

EITHER MODE

6. An exception may occur, depending on the value of the operand.
7. $\overline{\text{LOCK}}$ is automatically asserted regardless of the presence or absence of the LOCK instruction prefix.
8. $\overline{\text{LOCK}}$ does not remain active between all operand transfers.

PROTECTED VIRTUAL ADDRESS MODE ONLY

9. A general protection exception (13) will occur if the memory operand cannot be used due to either a segment limit or access rights violation. If a stack segment limit is violated, a stack segment overrun exception (12) occurs.
10. For segment load operations, the CPL, RPL, and DPL must agree with privilege rules to avoid an exception. The segment must be present to avoid a not-present exception (11). If the SS register is the destination, and a segment not-present violation occurs, a stack exception (12) occurs.
11. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert $\overline{\text{LOCK}}$ to maintain descriptor integrity in multiprocessor systems.
12. JMP, CALL, INT, RET, IRET instructions referring to another code segment will cause a general protection exception (13) if any privilege rule is violated.
13. A general protection exception (13) occurs if CPL $\neq$ 0.
14. A general protection exception (13) occurs if CPL > IOPL.
15. The IF field of the flag word is not updated if CPL > IOPL. The IOPL field is updated only if CPL = 0.
16. Any violation of privilege rules as applied to the selector operand do not cause a protection exception; rather, the instruction does not return a result and the zero flag is cleared.
17. If the starting address of the memory operand violates a segment limit, or an invalid access is attempted, a general protection exception (13) will occur before the ESC instruction is executed. A stack segment overrun exception (12) will occur if the stack limit is violated by the operand's starting address. If a segment limit is violated during an attempted data transfer then a processor extension segment overrun exception (9) occurs.
18. The destination of an INT, JMP, CALL, RET or IRET instruction must be in the defined limit of a code segment or a general protection exception (13) will occur.

80286 INSTRUCTION SET SUMMARY

FUNCTION	FORMAT				CLOCK COUNT Real Address Mode	CLOCK COUNT Protected Virtual Address Mode	COMMENTS Real Address Mode	COMMENTS Protected Virtual Address Mode
DATA TRANSFER								
MOV = Move:								
Register to Register/Memory	1000100w	mod reg r/m			2,3*	2,3*	2	9
Register/memory to register	1000101w	mod reg r/m			2,5*	2,5*	2	9
Immediate to register/memory	1100011w	mod 000 r/m	data	data if w = 1	2,3*	2,3*	2	9
Immediate to register	1011w reg	data	data if w=1		2	2		
Memory to accumulator	1010000w	addr-low	addr-high		5	5	2	9
Accumulator to memory	1010001w	addr-low	addr-high		3	3	2	9
Register/memory to segment register	10001110	mod 0 reg r/m			2,5*	17,19*	2	9,10,11
Segment register to register/memory	10001100	mod 0 reg r/m			2,3*	2,3*	2	9
PUSH = Push:								
Memory	11111111	mod 110 r/m			5*	5*	2	9
Register	01010 reg				3	3	2	9
Segment register	000 reg 110				3	3	2	9
Immediate	011010s0	data	data if s=0		3	3	2	9
PUSHA = Push All	01100000				17	17	2	9
POP = Pop:								
Memory	10001111	mod 000 r/m			5*	5*	2	9
Register	01011 reg				5	5	2	9
Segment register	000 reg 111	(reg≠01)			5	20	2	9,10,11
POPA = Pop All	01100001				19	19	2	9
XCHG = Exhcange:								
Register/memory with register	1000011w	mod reg r/m			3,5*	3,5*	2,7	7,9
Register with accumulator	10010 reg				3	3		
IN = Input from:								
Fixed port	1110010w	port			5	5		14
Variable port	1110110w				5	5		14
OUT = Output to:								
Fixed port	1110011w	port			3	3		14
Variable port	1110111w				3	3		14
XLAT = Translate byte to AL	11010111				5	5		9
LEA = Load EA to register	10001101	mod reg r/m			3*	3*		
LDS = Load pointer to DS	11000101	mod reg r/m	(mod≠11)		7*	21*	2	9,10,11
LES = Load pointer to ES	11000100	mod reg r/m	(mod≠1)		7*	21*	2	9,10,11

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT				CLOCK COUNT Real Address Mode	CLOCK COUNT Protected Virtual Address Mode	COMMENTS Real Address Mode	COMMENTS Protected Virtual Address Mode
DATA TRANSFER (Continued)								
LAHF Load AH with flags	1 0 0 1 1 1 1 1				2	2		
SAHF = Store AH into flags	1 0 0 1 1 1 1 0				2	2		
PUSHF = Push flags	1 0 0 1 1 1 0 0				3	3	2	9
POPF = Pop flags	1 0 0 1 1 1 0 1				5	5	2,4	9,15
ARITHMETIC								
ADD = Add:								
Reg/memory with register to either	0 0 0 0 0 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s w = 01	3,7*	3,7*	2	9
Immediate to accumulator	0 0 0 0 0 1 0 w	data	data if w = 1		3	3		
ADC = Add with carry:								
Reg/memory with register to either	0 0 0 1 0 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s w = 01	3,7*	3,7*	2	9
Immediate to accumulator	0 0 0 1 0 1 0 w	data	data if w = 1		3	3		
INC = Increment:								
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m			2,7*	2,7*	2	9
Register	0 1 0 0 0 reg				2	2		
SUB = Subtract:								
Reg/memory and register to either	0 0 1 0 1 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate from register/memory	1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s w = 01	3,7*	3,7*	2	9
Immediate from accumulator	0 0 1 0 1 1 0 w	data	data if w = 1		3	3		
SBB = Subtract with borrow:								
Reg/memory and register to either	0 0 0 1 1 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate from register/memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s w = 01	3,7*	3,7*	2	9
Immediate from accumulator	0 0 0 1 1 1 0 w	data	data if w = 1		3	3		
DEC = Decrement								
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m			2,7*	2,7*	2	9
Register	0 1 0 0 1 reg				2	2		
CMP = Compare								
Register/memory with register	0 0 1 1 1 0 1 w	mod reg r/m			2,6*	2,6*	2	9
Register with register/memory	0 0 1 1 1 0 0 w	mod reg r/m			2,7*	2,7*	2	9
Immediate with register/memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s w = 01	3,6*	3,6*	2	9
Immediate with accumulator	0 0 1 1 1 1 0 w	data	data if w = 1		3	3		
NEG = Change sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m			2	7*	2	9
AAA = ASCII adjust for add	0 0 1 1 0 1 1 1				3	3		
DAA = Decimal adjust for add	0 0 1 0 0 1 1 1				3	3		

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT					CLOCK COUNT Real Address Mode	CLOCK COUNT Protected Virtual Address Mode	COMMENTS Real Address Mode	COMMENTS Protected Virtual Address Mode
ARITHMETIC (Continued)									
AAS = ASCII adjust for subtract	0 0 1 1 1 1 1 1					3	3		
DAS = Decimal adjust for subtract	0 0 1 0 1 1 1 1					3	3		
MUL = Multiply (unsigned):	1 1 1 1 0 1 1 w	mod 1 0 0	r/m						
Register-Byte						13	13		
Register-Word						21	21		
Memory-Byte						16*	16*	2	9
Memory-Word						24*	24*	2	9
IMUL = Integer multiply (signed):	1 1 1 1 0 1 1 w	mod 1 0 1	r/m						
Register-Byte						13	13		
Register-Word						21	21		
Memory-Byte						16*	16*	2	9
Memory-Word						24*	24*	2	9
IMUL = Integer immediate multiply (signed)	0 1 1 0 1 0 s 1	mod reg	r/m	data	data if s = 0	21,24*	21,24*	2	9
DIV = Divide (unsigned)	1 1 1 1 0 1 1 w	mod 1 1 0	r/m						
Register-Byte						14	14	6	6
Register-Word						22	22	6	6
Memory-Byte						17*	17*	2,6	6,9
Memory-Word						25*	25*	2,6	6,9
IDIV = Integer divide (signed)	1 1 1 1 0 1 1 w	mod 1 1 1	r/m						
Register-Byte						17	17	6	6
Register-Word						25	25	6	6
Memory-Byte						20*	20*	2,6	6,9
Memory-Word						28*	28*	2,6	6,9
AAM = ASCII adjust for multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0				16	16		
AAD = ASCII adjust for divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0				14	14		
CBW = Convert byte to word	1 0 0 1 1 0 0 0					2	2		
CWD = Convert word to double word	1 0 0 1 1 0 0 1					2	2		
LOGIC									
Shift/Rotate Instructions:									
Register/Memory by 1	1 1 0 1 0 0 0 w	mod TTT	r/m			2,7*	2,7*	2	9
Register/Memory by CL	1 1 0 1 0 0 1 w	mod TTT	r/m			5+n,8+n*	5+n,8+n*	2	9
Register/Memory by Count	1 1 0 0 0 0 0 w	mod TTT	r/m	count		5+n,8+n*	5+n,8+n*	2	9

TTT	Instruction
0 0 0	ROL
0 0 1	ROR
0 1 0	RCL
0 1 1	RCR
1 0 0	SHL/SAL
1 0 1	SHR
1 1 1	SAR

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT				CLOCK COUNT Real Address Mode	CLOCK COUNT Protected Virtual Address Mode	COMMENTS Real Address Mode	COMMENTS Protected Virtual Address Mode
ARITHMETIC (Continued)								
AND = **And:**								
Reg/memory and register to either	0 0 1 0 0 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w = 1	3,7*	3,7*	2	9
Immediate to accumulator	0 0 1 0 0 1 0 w	data	data if w = 1		3	3		
TEST = **And function to flags, no result:**								
Register/memory and register	1 0 0 0 0 1 0 w	mod reg r/m			2,6*	2,6*	2	9
Immediate data and register/memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1	3,6*	3,6*	2	9
Immediate data and accumulator	1 0 1 0 1 0 0 w	data	data if w = 1		3	3		
OR = **Or:**								
Reg/memory and register to either	0 0 0 0 1 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w = 1	3,7*	3,7*	2	9
Immediate to accumulator	0 0 0 0 1 1 0 w	data	data if w = 1		3	3		
XOR = **Exclusive or:**								
Reg/memory and register to either	0 0 1 1 0 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w = 1	3,7*	3,7*	2	9
Immediate to accumulator	0 0 1 1 0 1 0 w	data	data if w = 1		3	3		
NOT = Invert register/memory	1 1 1 1 0 1 1 w	mod 0 1 0 r/m			2,7*	2,7*	2	9
STRING MANIPULATION:								
MOVS = Move byte/word	1 0 1 0 0 1 0 w				5	5	2	9
CMPS = Compare byte/word	1 0 1 0 0 1 1 w				8	8	2	9
SCAS = Scan byte/word	1 0 1 0 1 1 1 w				7	7	2	9
LODS = Load byte/wd to AL/AX	1 0 1 0 1 1 0 w				5	5	2	9
STOS = Stor byte/wd from AL/A	1 0 1 0 1 0 1 w				3	3	2	9
INS = Input byte/wd from DX port	0 1 1 0 1 1 0 w				5	5	2	9,14
OUTS = Output byte/wd to DX port	0 1 1 0 1 1 1 w				5	5	2	9,14
Repeated by count in CX								
MOVS = Move string	1 1 1 1 0 0 1 1	1 0 1 0 0 1 0 w			5+4n	5+4n	2	9
CMPS = Compare string	1 1 1 1 0 0 1 z	1 0 1 0 0 1 1 w			5+9n	5+9n	2,8	8,9
SCAS = Scan string	1 1 1 1 0 0 1 z	1 0 1 0 1 1 1 w			5+8n	5+8n	2,8	8,9
LODS = Load string	1 1 1 1 0 0 1 1	1 0 1 0 1 1 0 w			5+4n	5+4n	2,8	8,9
STOS = Store string	1 1 1 1 0 0 1 1	1 0 1 0 1 0 1 w			4+3n	4+3n	2,8	8,9
INS = Input string	1 1 1 1 0 0 1 1	0 1 1 0 1 1 0 w			5+4n	5+4n	2	9,14
OUTS = Output string	1 1 1 1 0 0 1 1	0 1 1 0 1 1 1 w			5+4n	5+4n	2	9,14

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT				Clock Count: Real Address Mode	Clock Count: Protected Virtual Address Mode	Comments: Real Address Mode	Comments: Protected Virtual Address Mode
CONTROL TRANSFER								
CALL = Call:								
Direct within segment	1 1 1 0 1 0 0 0	disp-low	disp-high		7+m	7+m	2	18
Register/memory indirect within segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m			7 +m, 11+m*	7+m, 11+m*	2,8	8,9,18
Direct intersegment	1 0 0 1 1 0 1 0	segment offset			13+m	26+m	2	11,12,18
Protected Mode Only (Direct intersegment):		segment selector						
Via call gate to same privilege level						41+m		8,11,12,18
Via call gate to different privilege level, no parameters						82+m		8,11,12,18
Via call gate to different privilege level, x parameters						86 +4x+m		8,11,12,18
Via TSS						177+m		8,11,12,18
Via task gate						182+m		8,11,12,18
Indirect intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m		(mod≠11)	16+m	29+m*	2	8,9,11,12,18
Protected Mode Only (Indirect intersegment):								
Via call gate to same privilege level						44+m*		8,9,11,12,18
Via call gate to different privilege level, no parameters						83 +m*		8,9,11,12,18
Via call gate to different privilege level, x parameters						90+4x +m*		8,9,11,12,18
Via TSS						180+m*		8,9,11,12,18
Via task gate						185+m*		8,9,11,12,18
JMP = Unconditional jump:								
Short/long	1 1 1 0 1 0 1 1	disp-low			7+m	7+m		18
Direct within segment	1 1 1 0 1 0 0 1	disp-low	disp-high		7+m	7+ m		18
Register/memory indirect within segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m			7 +m, 11+m*	7+m, 11+m*	2	9,18
Direct intersegment	1 1 1 0 1 0 1 0	segment offset			11+m	23+m		11,12,18
Protected Mode Only (Direct intersegment):		segment selector						
Via call gate to same privilege level						38+m		8,11,12,18
Via TSS						175+m		8,11,12,18
Via task gate						180+m		8,11,12,18
Indirect intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m		(mod≠11)	15+m*	26+m*	2	8,9,11,12,18
Protected Mode Only (Indirect intersegment):								
Via call gate to same privilege level						41+m*		8,9,11,12,18
Via TSS						178+m*		8,9,11,12,18
Via task gate						183+m*		8,9,11,12,18
RET = Return from CALL:								
Within segment	1 1 0 0 0 0 1 1				11+m	11+m	2	8,9,18
Within seg adding immed to SP	1 1 0 0 0 0 1 0	data-low	data-high		11+m	11+m	2	8,9,18
Intersegment	1 1 0 0 1 0 1 1				15+m	25+m	2	8,9,11,12,18
Intersegment adding immediate to SP	1 1 0 0 1 0 1 0	data-low	data-high		15+m		2	8,9,11,12,18
Protected Mode Only (RET):								
To different privilege level						55+m		9,11,12,18

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT Real Address Mode	CLOCK COUNT Protected Virtual Address Mode	COMMENTS Real Address Mode	COMMENTS Protected Virtual Address Mode
CONTROL TRANSFER (Continued)					
JE/JZ = Jump on equal zero	01110100 disp	7+m or 3	7+m or 3		18
JL/JNGE = Jump on less/not greater or equal	01111100 disp	7+m or 3	7+m or 3		18
JLE/JNG = Jump on less or equal/not greater	01111110 disp	7+m or 3	7+m or 3		18
JB/JNAE = Jump on below/not above or equal	01110010 disp	7+m or 3	7+m or 3		18
JBE/JNA = Jump on below or equal/not above	01110110 disp	7+m or 3	7+m or 3		18
JP/JPE = Jump on parity/parity even	01111010 disp	7+m or 3	7+m or 3		18
JO = Jump on overflow	01110000 disp	7 +m or 3	7+m or 3		18
JS = Jump on sign	01111000 disp	7+ m or 3	7+m or 3		18
JNE/JNZ = Jump on not equal/not zero	01110101 disp	7+m or 3	7+m or 3		18
JNL/JGE = Jump on not less/greater or equal	01111101 disp	7+m or 3	7+m or 3		18
JNLE/JG = Jump on not less or equal/greater	01111111 disp	7+m or 3	7+m or 3		18
JNB/JAE = Jump on not below/above or equal	01110011 disp	7+m or 3	7+m or 3		18
JNBE/JA = Jump on not below or equal/above	01110111 disp	7+m or 3	7+m or 3		18
JNP/JPO = Jump on not par/par odd	01111011 disp	7+m or 3	7+m or 3		18
JNO = Jump on not overflow	01110001 disp	7+m or 3	7+m or 3		18
JNS = Jump on not sign	01111001 disp	7 +m or 3	7+m or 3		18
LOOP = Loop CX times	11100010 disp	8 +m or 4	8+m or 4		18
LOOPZ/LOOPE = Loop while zero/equal	11100001 disp	8+m or 4	8+m or 4		18
[illegible]	[illegible]	[illegible]	[illegible]		[illegible]
JCXZ = Jump on CX zero	11100011 disp	8 +m or 4	8+m or 4		18
ENTER = Enter Procedure	11001000 data-low data-high L			2,8	8,9
L=0		11	11	2,8	8,9
L=1		15	15	2,8	8,9
L > 1		16+4(L − 1)	16+4(L − 1)	2,8	8,9
LEAVE = Leave Procedure	11001001	5	5		
INT = Interrupt:					
Type specified	11001101 type	23+m		2,7,8	
Type 3	11001100	23+m		2,7,8	
INTO = Interrupt on overflow	11001110	24 +m or 3 (3 if no interrupt)	(3 if no interrupt)	2,6,8	

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT: Real Address Mode	CLOCK COUNT: Protected Virtual Address Mode	COMMENTS: Real Address Mode	COMMENTS: Protected Virtual Address Mode
CONTROL TRANSFER (Continued)					
Protected Mode Only:					
Via interrupt or trap gate to same privilege level			40+ m		7,8,11,12,18
Via interrupt or trap gate to fit different privilege level			78+ m		7,8,11,12,18
Via Task Gate			167+m		7,8,11,12,18
IRET = Interrupt return	11001111	17+m	31+ m	2,4	8,9,11,12,15,18
Protected Mode Only:					
To different privilege level			55+m		8,9,11,12,15,18
To different task (NT=1)			169+m		8,9,11,12,18
BOUND = Detect value out of range	01100010 | mod reg r/m	13*	13* (Use INT clock count if exception 5)	2,6	6,8,9,11,12,18
PROCESSOR CONTROL					
CLC = Clear carry	11111000	2	2		
CMC = Complement carry	11110101	2	2		
STC = Set carry	11111001	2	2		
CLD = Clear direction	11111100	2	2		
STD = Set direction	11111101	2	2		
CLI = Clear interrupt	11111010	3	3		14
STI = Set interrupt	11111011	2	2		14
HLT = Halt	11110100	2	2		13
WAIT = Wait	10011011	3	3		
LOCK = Bus lock prefix	11110000	0	0		14
CTS = Clear task switched flag	00001111 | 00000110	2	2	3	13
ESC = Processor Extension Escape	11011TTT | mod LLL r/m (TTT LLL are opcode to processor extension)	9-20*	9-20*	5,8	8,17
SEG = Segment Override Prefix	001 reg 110	0	0		
PROTECTION CONTROL					
LGDT = Load global descriptor table register	00001111 | 00000001 | mod 010 r/m	11*	11*	2,3	9,13
SGDT = Store global descriptor table register	00001111 | 00000001 | mod 000 r/m	11*	11*	2,3	9
LIDT = Load interrupt descriptor table register	00001111 | 00000001 | mod 011 r/m	12*	12*	2,3	9,13
SIDT = Store interrupt descriptor table register	00001111 | 00000001 | mod 001 r/m	12*	12*	2,3	9
LLDT = Load local descriptor table register from register memory	00001111 | 00000000 | mod 010 r/m		17,19*	1	9,11,13
SLDT = Store local descriptor table register to register/memory	00001111 | 00000000 | mod 000 r/m		2,3*	1	9

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT			CLOCK COUNT		COMMENTS	
				Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
PROTECTION CONTROL (Continued)							
LTR = Local task register from register/memory	00001111	00000000	mod 0 1 1 r/m		17,19*	1	9,11,13
STR = Store task register to register memory	00001111	00000000	mod 0 0 1 r/m		2,3*	1	9
LMSW = Load machine status word from register/memory	00001111	00000001	mod 1 1 0 r/m	3,6*	3,6*	2,3	9,13
SMSW = Store machine status word	00001111	00000001	mod 1 0 0 r/m	2,3*	2,3*	2,3	9
LAR = Load access rights from register/memory	00001111	00000010	mod reg r/m		14,16*	1	9,11,16
LSL = Load segment limit from register/memory	00001111	00000011	mod reg r/m		14,16*	1	9,11,16
ARPL = Adjust requested privilege level: from register/memory		01100011	mod reg r/m	10*,11*	2	8,9	
VERR = Verify read access: register/memory	00001111	00000000	mod 1 0 0 r/m	14,16*	1	9,11,16	
VERR = Verify write access:	00001111	00000000	mod 1 0 1 r/m	14,16*	1	9,11,16	

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

Footnotes

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16 bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low

if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EQ = disp-high: disp-low.

SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

reg is assigned according to the following:

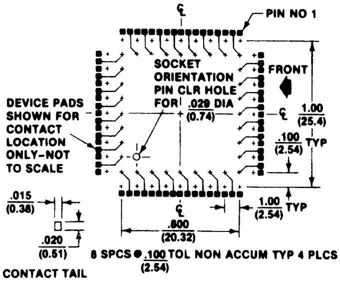
reg	Segment Register
00	ES
01	CS
10	SS
11	DC

REG is assigned according to the following table:

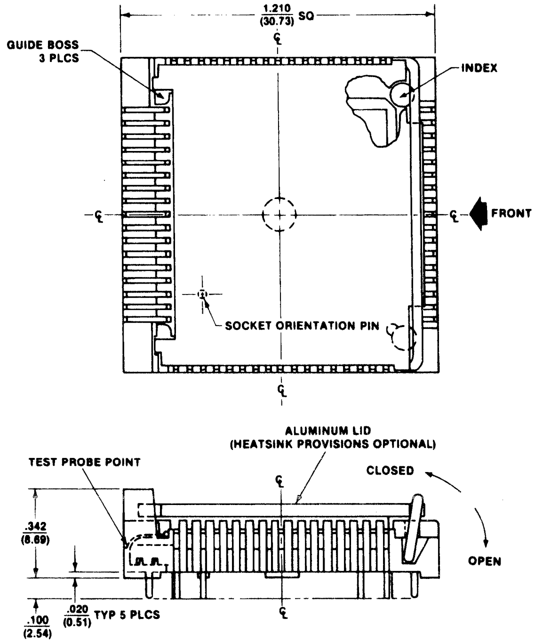
16-Bit (w = 1)		8-Bit (w = 0)	
000	AX	000	AL
001	CX	001	CL
010	DX	010	DL
011	BX	011	BL
100	SP	100	AH
101	BP	101	CH
101	SI	110	DH
111	DI	111	BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

PC BOARD PATTERN



210253-48



210253-49

Figure 36. Textool 68 Lead Chip Carrier Socket

80287
80-Bit HMOS
NUMERIC PROCESSOR EXTENSION
(80287-6, 80287-8, 80287-10)

- **High Performance 80-Bit Internal Architecture**
- **Implements Proposed IEEE Floating Point Standard 754**
- **Expands iAPX 286/10 Datatypes to Include 32-, 64-, 80-Bit Floating Point, 32-, 64-Bit Integers and 18-Digit BCD Operands**
- **Object Code Compatible with 8087**
- **Built-in Exception Handling**
- **Operates in Both Real and Protected Mode iAPX 286 Systems**
- **8x80-Bit, Individually Addressable, Numeric Register Stack**
- **Protected Mode Operation Completely Conforms to the iAPX 286 Memory Management and Protection Mechanisms**
- **Directly Extends iAPX 286/10 Instruction Set to Trigonometric, Logarithmic, Exponential and Arithmetic Instructions for All Datatypes**
- **Compatible with 80386 CPU**
- **Available in EXPRESS—Standard Temperature Range**

Available in 40 pin-Cerdip package (see Packaging Spec: Order #231369)

The Intel® 80287 is a high performance numerics processor extension that extends the iAPX 286/10 architecture with floating point, extended integer and BCD data types. The iAPX 186/20 computing system (80286 with 80287) fully conforms to the proposed IEEE Floating Point Standard. Using a numerics oriented architecture, the 80287 adds over fifty mnemonics to the iAPX 286/20 instruction set, making the iAPX 286/20 a complete solution for high performance numeric processing. The 80287 is implemented in N-channel, depletion load, silicon gate technology (HMOS) and packaged in a 40-pin cerdip package. The iAPX 286/20 is object code compatible with the iAPX 86/20 and iAPX 88/20.

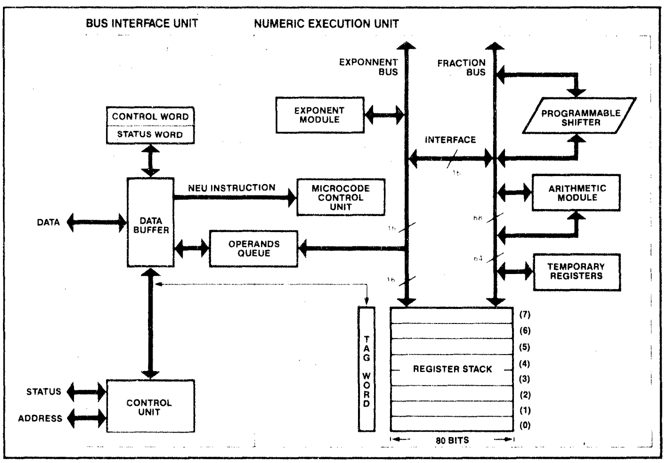
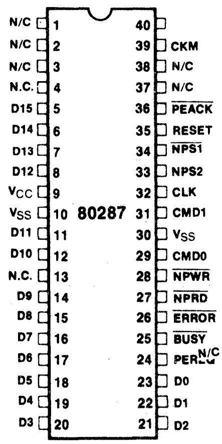


Figure 1. 80287 Block Diagram



NOTE:
N.C. PINS MUST NOT BE CONNECTED.

Figure 2. 80287 Pin Configuration

January 1986
Order Number: 210920-004

Table 1. 80287 Pin Description

Symbols	Type	Name and Function
CLK	I	Clock input: this clock provides the basic timing for internal 80287 operations. Special MOS level inputs are required. The 82284 or 8284A CLK outputs are compatible to this input.
CKM	I	Clock Mode signal: indicates whether CLK input is to be divided by 3 or used directly. A HIGH input will cause CLK to be used directly. This input may be connected to V_{CC} or V_{SS} as appropriate. This input must be either HIGH or LOW 20 CLK cycles before RESET goes LOW.
RESET	I	System Reset: causes the 80287 to immediately terminate its present activity and enter a dormant state. RESET is required to be HIGH for more than 4 80287 CLK cycles. For proper initialization the HIGH-LOW transition must occur no sooner than 50 μs after V_{CC} and CLK meet their D.C. and A.C. specifications.
D15-D0	I/O	Data: 16-bit bidirectional data bus. Inputs to these pins may be applied asynchronous to the 80287 clock.
$\overline{\text{BUSY}}$	O	Busy status: asserted by the 80287 to indicate that it is currently executing a command.
$\overline{\text{ERROR}}$	O	Error status: reflects the ES bit of the status word. This signal indicates that an unmasked error condition exists.
PEREQ	O	Processor Extension Data Channel operand transfer request: a HIGH on this output indicates that the 80287 is ready to transfer data. PEREQ will be disabled upon assertion of $\overline{\text{PEACK}}$ or upon actual data transfer, whichever occurs first, if no more transfers are required.
$\overline{\text{PEACK}}$	I	Processor Extension Data Channel operand transfer ACKnowledge: acknowledges that the request signal (PEREQ) has been recognized. Will cause the request (PEREQ) to be withdrawn in case there are no more transfers required. $\overline{\text{PEACK}}$ may be asynchronous to the 80287 clock.
$\overline{\text{NPRD}}$	I	Numeric Processor Read: Enables transfer of data from the 80287. This input may be asynchronous to the 80287 clock.
$\overline{\text{NPWR}}$	I	Numeric Processor Write: Enables transfer of data to the 80287. This input may be asynchronous to the 80287 clock.
$\overline{\text{NPS1}}$, NPS2	I	Numeric Processor Selects: indicate the CPU is performing an ESCAPE instruction. Concurrent assertion of these signals (i.e., $\overline{\text{NPS1}}$ is LOW and NPS2 is HIGH) enables the 80287 to perform floating point instructions. No data transfers involving the 80287 will occur unless the device is selected via these lines. These inputs may be asynchronous to the 80287 clock.
CMD1, CMD0	I	Command lines: These, along with select inputs, allow the CPU to direct the operation of the 80287. These inputs may be asynchronous to the 80287 clock.

Table 1. 80287 Pin Description (cont.)

Symbols	Type	Name and Function
V_{SS}	I	System ground, both pins must be connected to ground.
V_{CC}	I	+5V supply

FUNCTIONAL DESCRIPTION

The 80287 Numeric Processor Extension (NPX) provides arithmetic instructions for a variety of numeric data types in iAPX 286/20 systems. It also executes numerous built-in transcendental functions (e.g., tangent and log functions). The 80287 executes instructions in parallel with a 80286. It effectively extends the register and instruction set of an iAPX 286/10 system for existing iAPX 286 data types and adds several new data types as well. Figure 3 presents the program visible register model of the iAPX 286/20. Essentially, the 80287 can be treated as an additional resource or an extension to the iAPX 286/10 that can be used as a single unified system, the iAPX 286/20.

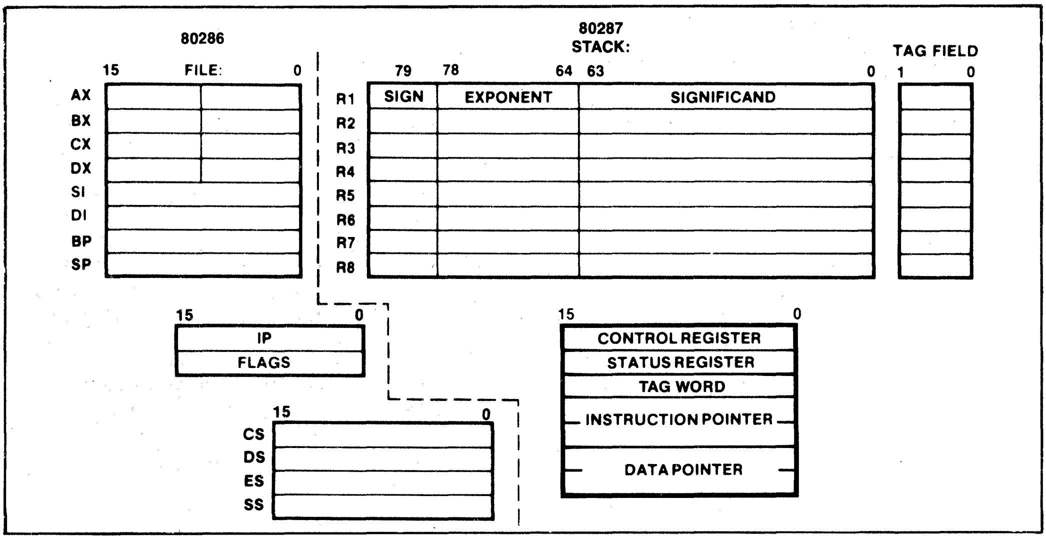


Figure 3. iAPX 286/20 Architecture

The 80287 has two operating modes similar to the two modes of the 80286. When reset, 80287 is in the real address mode. It can be placed in the protected virtual address mode by executing the SETPM ESC instruction. The 80287 cannot be switched back to the real address mode except by reset. In the real address mode, the iAPX 286/20 is completely software compatible with iAPX 86/20, 88/20.

Once in protected mode, all references to memory for numerics data or status information, obey the iAPX 286 memory management and protection rules giving a fully protected extension of the 80286 CPU. In the protected mode, iAPX 286/20 numerics software is also completely compatible with iAPX 86/20 and iAPX 88/20.

SYSTEM CONFIGURATION

As a processor extension to an 80286, the 80287 can be connected to the CPU as shown in Figure 4A. The data channel control signals (PEREQ, PEACK), the BUSY signal and the NPRD, NPWR signals, allow the NPX to receive instructions and data from the CPU. When in the protected mode, all information received by the NPX is validated by the 80286 memory management and protection unit. Once started, the 80287 can process in parallel with and independent of the host CPU. When the NPX detects an error or exception, it will indicate this to the CPU by asserting the ERROR signal.

The NPX uses the processor extension request and acknowledge pins of the 80286 CPU to implement data transfers with memory under the protection model of the CPU. The full virtual and physical address space of the 80286 is available. Data for the 80287 in memory is addressed and represented in the same manner as for an 8087.

The 80287 can operate either directly from the CPU clock or with a dedicated clock. For operation with the CPU clock (CKM=0), the 80287 works at one-third the frequency of the system clock (i.e., for an 8 MHz 80286, the 16 MHz system clock is divided down to 5.3 MHz). The 80287 provides a capability to internally divide the CPU clock by three to produce the required internal clock (33% duty cycle). To use a higher performance 80287 (8 MHz), an 8284A clock driver and appropriate crystal may be used to directly drive the 80287 with a 1/3 duty cycle clock on the CLK input (CKM=1). The following table describes the relationship between clock speed and 287 speed as a function of CKM state.

287 Speed	CLK Speed CKM = 0	CLK Speed CKM = 1
6 MHz	16 MHz	6 MHz
8 MHz	20 MHz	8 MHz
10 MHz	25 MHz	10 MHz

Figure 4B details the 80287 connected as a processor extension to 80386.

HARDWARE INTERFACE

Communication of instructions and data operands between the 80286 and 80287 is handled by the CMD0, CMD1, NPS1, NPS2, NPRD, and NPWR signals. I/O port addresses 00F8H, 00FAH, and 00FCH are used by the 80286 for this communication. When any of these addresses are used, the NPS1 input must be LOW and NPS2 input HIGH. The IORC and IOWC outputs of the 82288 identify I/O space transfers (see Figure 4). CMD0 should be connected to latched 80286 A1 and CMD1 should be connected to latched 80286 A2.

I/O ports 00F8H to 00FFH are reserved for the 80286/80287 interface. To guarantee correct operation of the 80287, programs must not perform any I/O operations to these ports.

The PEREQ, PEACK, BUSY, and ERROR signals of the 80287 are connected to the same-named 80286 input. The data pins of the 80287 should be directly connected to the 80286 data bus. Note that all bus drivers connected tothe 80286 local bus must be inhibited when the 80286 reads from the 80287. The use of M/IO in the decoder prevents INTA bus cycles from disabling the data transceivers.

PROGRAMMING INTERFACE

Table 2 lists the seven data types the 80287 supports and presents the format for each type. These values are stored in memory with the least significant digits at the lowest memory address. Programs retrieve these values by generating the lowest address. All values should start at even addresses for maximum system performance.

Internally the 80287 holds all numbers in the temporary real format. Load instructions automatically convert operands represented in memory as 16-, 32-, or 64-bit integers, 32- or 64-bit floating point number or 18-digit packed BCD numbers into temporary real format. Store instructions perform the reverse type conversion.

80287 computations use the processor's register stack. These eight 80-bit registers provide the equivalent capacity of 40 16-bit registers. The 80287 register set can be accessed as a stack, with instructions operating on the top one or two stack elements, or as a fixed register set, with instructions operating on explicitly designated registers.

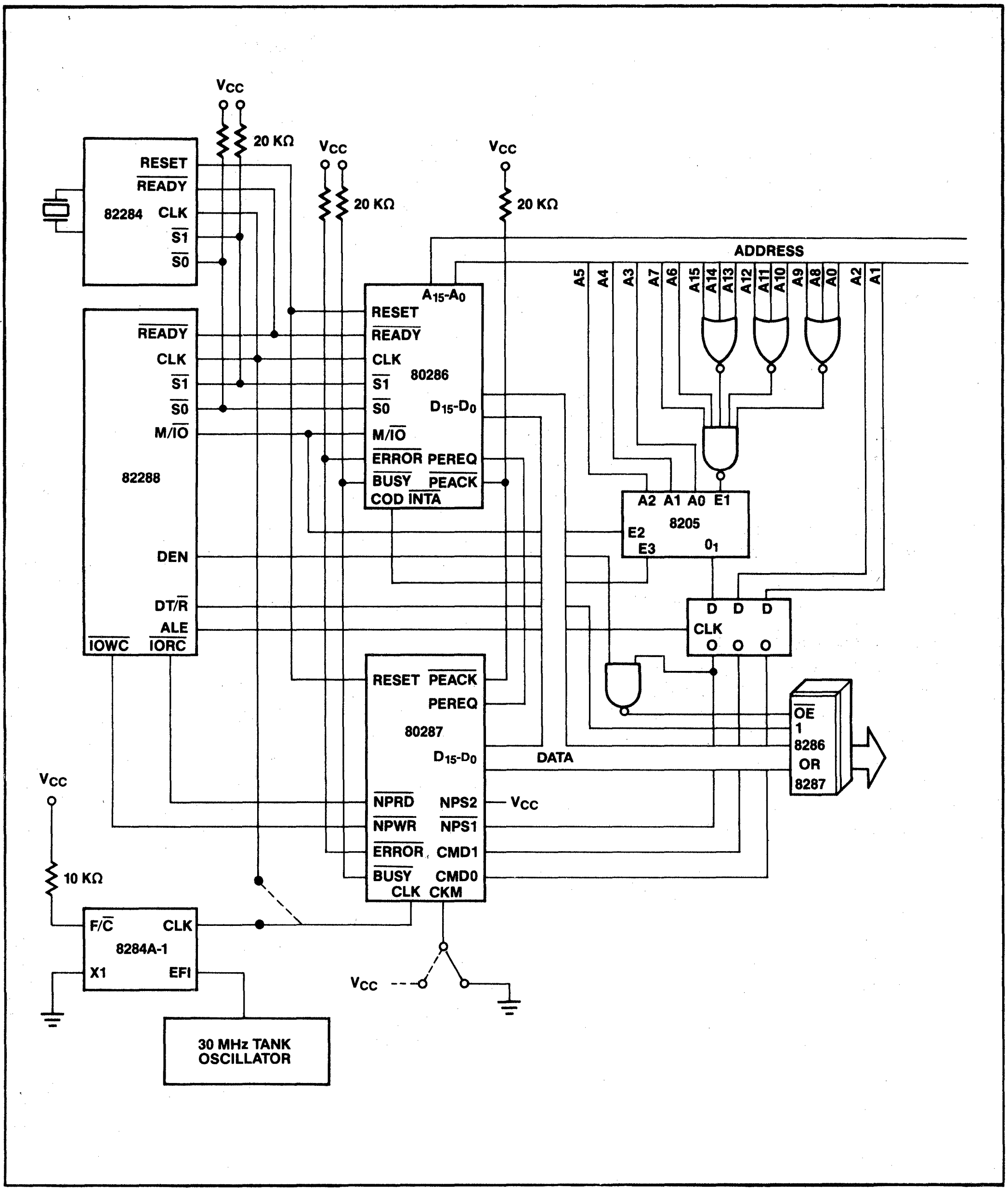


Figure 4A. 80286/80287 System Configuration

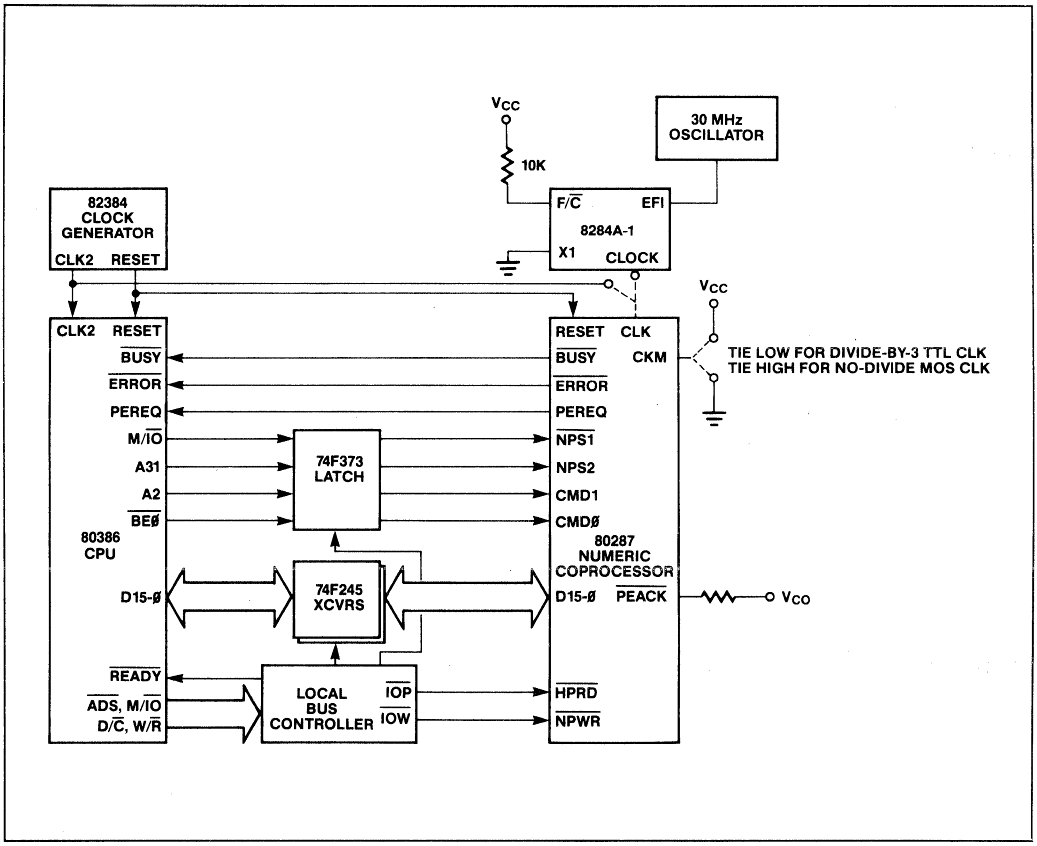

VCC
10K
30 MHz OSCILLATOR
F/C
EFI
8284A-1
X1
CLOCK
82384 CLOCK GENERATOR
CLK2 RESET
VCC
CLK2 RESET
BUSY
ERROR
PEREQ
M/IO
A31
A2
BEØ
80386 CPU
D15-Ø
READY
ADS, M/IO
D/C, W/R
74F373 LATCH
74F245 XCVRS
LOCAL BUS CONTROLLER
IOP
IOW
RESET CLK
BUSY CKM
ERROR
PEREQ
NPS1
NPS2
CMD1
CMDØ
80287 NUMERIC COPROCESSOR
D15-Ø PEACK
HPRD
NPWR
TIE LOW FOR DIVIDE-BY-3 TTL CLK
TIE HIGH FOR NO-DIVIDE MOS CLK
VCO

Table 2. 80287 Datatype Representation in Memory

Data Formats	Range	Precision	Most Significant Byte — HIGHEST ADDRESSED BYTE (7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0)
Word Integer	10^4	16 Bits	[ ] (TWO'S COMPLEMENT) — 15 … 0
Short Integer	10^9	32 Bits	[ ] (TWO'S COMPLEMENT) — 31 … 0
Long Integer	10^{19}	64 Bits	[ ] (TWO'S COMPLEMENT) — 63 … 0
Packed BCD	10^{18}	18 Digits	S | X | MAGNITUDE: d_{17} d_{16} d_{15} d_{14} d_{13} d_{12} d_{11} d_{10} d_9 d_8 d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0 — 79, 72, 0
Short Real	$10^{\pm 38}$	24 Bits	S | BIASED EXPONENT | SIGNIFICAND — 31, 23, 0; I ▲
Long Real	$10^{\pm 308}$	53 Bits	S | BIASED EXPONENT | SIGNIFICAND — 63, 52, 0; I ▲
Temporary Real	$10^{\pm 4932}$	64 Bits	S | BIASED EXPONENT | I | SIGNIFICAND — 79, 64, 63 ▲, 0

NOTES:

(1) S = Sign bit (0 = positive, 1 = negative)
(2) d_n = Decimal digit (two per byte)
(3) X = Bits have no significance; 8087 ignores when loading, zeros when storing.
(4) ▲ = Position of implicit binary point
(5) I = Integer bit of significand; stored in temporary real, implicit in short and long real
(6) Exponent Bias (normalized values):
Short Real: 127 (7FH)
Long Real: 1023 (3FFH)
Temporary Real: 16383 (3FFFH)
(7) Packed BCD: $(-1)^S(D_{17} \ldots D_0)$
(8) Real: $(-1)^S(2^{E\text{-}BIAS})(F_0\ F_1 \ldots)$

Table 6 lists the 80287's instructions by class. No special programming tools are necessary to use the 80287 since all new instructions and data types are directly supported by the iAPX 286 assembler and appropriate high level languages. All iAPX 86/88 development tools which support the 8087 can also be used to develop software for the iAPX 286/20 in real address mode.

SOFTWARE INTERFACE

The iAPX 286/20 is programmed as a single processor. All communication between the 80286 and the 80287 is transparent to software. The CPU automatically controls the 80287 whenever a numeric instruction is executed. All memory addressing modes, physical memory, and virtual memory of the CPU are available for use by the NPX.

Since the NPX operates in parallel with the CPU, any errors detected by the NPX may be reported after the CPU has executed the ESCAPE instruction which caused it. To allow identification of the failing numeric instruction, the NPX contains two pointer registers which identify the address of the failing numeric instruction and the numeric memory operand if appropriate for the instruction encountering this error.

INTERRUPT DESCRIPTION

Several interrupts of the iAPX 286 are used to report exceptional conditions while executing numeric programs in either real or protected mode. The interrupts and their functions are shown in Table 3.

PROCESSOR ARCHITECTURE

As shown in Figure 1, the NPX is internally divided into two processing elements, the bus interface unit (BIU) and the numeric execution unit (NEU). The NEU executes all numeric instructions, while the BIU receives and decodes instructions, requests operand transfers to and from memory and executes processor control instructions. The two units are able to operate independently of one another allowing the BIU to maintain asynchronous communication with the CPU while the NEU is busy processing a numeric instruction.

BUS INTERFACE UNIT

The BIU decodes the ESC instruction executed by the CPU. If the ESC code defines a math instruction, the BIU transmits the formatted instruction to the NEU. If the ESC code defines an administrative instruction, the BIU executes it independently of the NEU. The parallel operation of the NPX with the CPU is normally transparant to the user. The BIU generates the $\overline{\text{BUSY}}$ and $\overline{\text{ERROR}}$ signals for 80826/80287 processor synchronization and error notification, respectively.

The 80287 executes a single numeric instruction at a time. When executing most ESC instructions, the

Table 3. 80286 Interrupt Vectors Reserved for NPX

Interrupt Number	Interrupt Function
7	An ESC instruction was encountered when EM or TS of the 80286 MSW was set. EM=1 indicates that software emulation of the instruction is required. When TS is set, either an ESC or WAIT instruction will cause interrupt 7. This indicates that the current NPX context may not belong to the current task.
9	The second or subsequent words of a numeric operand in memory exceeded a segment's limit. This interrupt occurs after executing an ESC instruction. The saved return address will not point at the numeric instruction causing this interrupt. After processing the addressing error, the iAPX 286 program can be restarted at the return address with IRET. The address of the failing numeric instruction and numeric operand are saved in the 80287. An interrupt handler for this interrupt *must* execute FNINIT before *any* other ESC or WAIT instruction.
13	The starting address of a numeric operand is not in the segment's limit. The return address will point at the ESC instruction, including prefixes, causing this error. The 80287 has not executed this instruction. The instruction and data address in 80287 refer to a previous, correctly executed, instruction.
16	The previous numeric instruction caused an unmasked numeric error. The address of the faulty numeric instruction or numeric data operand is stored in the 80287. Only ESC or WAIT instructions can cause this interrupt. The 80286 return address will point at a WAIT or ESC instruction, including prefixes, which may be restarted after clearing the error condition in the NPX.

80286 tests the $\overline{\text{BUSY}}$ pin and waits until the 80287 indicates that it is not busy before initiating the command. Once initiated, the 80286 continues program execution while the 80287 executes the ESC instruction. In iAPX 86/20 systems, this synchronization is achieved by placing a WAIT instruction before an ESC instruction. For most ESC instructions, the iAPX 286/20 does not require a WAIT instruction before the ESC opcode. However, the iAPX 286/20 will operate correctly with these WAIT instructions. In all cases, a WAIT or ESC instruction should be inserted after any 80287 store to memory (except FSTSW and FSTCW) or load from memory (except FLDENV or FRSTOR) before the 80286 reads or changes the value to be sure the numeric value has already been written or read by the NPX.

Data transfers between memory and the 80287, when needed, are controlled by the PEREQ $\overline{\text{PEACK}}$, $\overline{\text{NPRD}}$, $\overline{\text{NPWR}}$, $\overline{\text{NPS1}}$, NPS2 signals. The 80286 does the actual data transfer with memory through its processor extension data channel. Numeric data transfers with memory performed by the 80286 use the same timing as any other bus cycle. Control signals for the 80287 are generated by the 80826 as shown in Figure 4, and meet the timing requirements shown in the AC requirements section.

NUMERIC EXECUTION UNIT

The NEU executes all instructions that involve the register stack; these include arithmetic, logical, transcendental, constant and data transfer instructions. The data path in the NEU is 84 bits wide (68 significand bits, 15 exponent bits and a sign bit) which allows internal operand transfers to be performed at very high speeds.

When the NEU begins executing an instruction, it activates the BIU $\overline{\text{BUSY}}$ signal. This signal is used in conjunction with the CPU WAIT instruction or automatically with most of the ESC instructions to synchronize both processors.

REGISTER SET

The 80287 register set is shown in Figure 5. Each of the eight data registers in the 8087's register stack

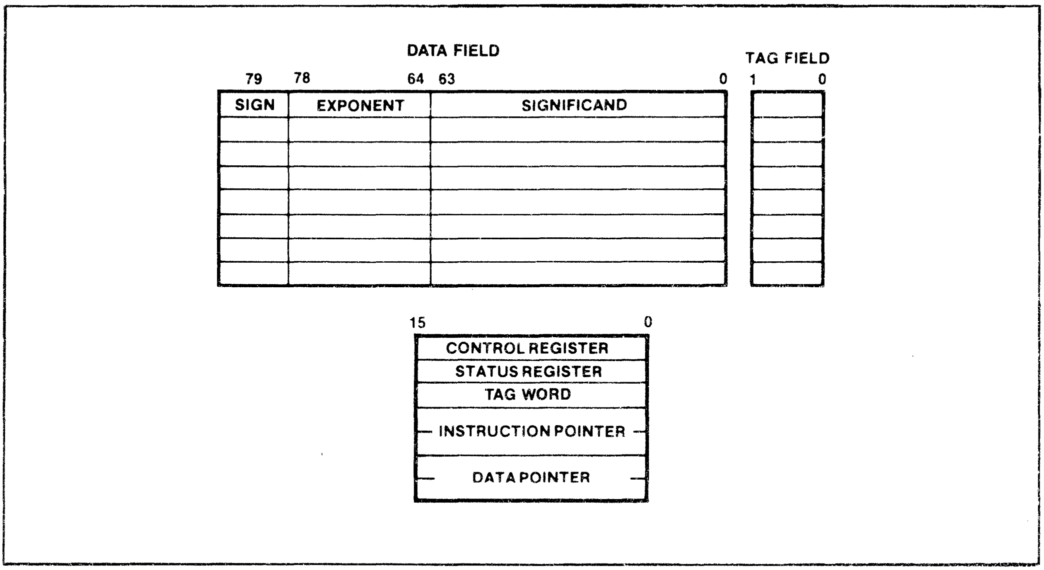


Figure 5. 80287 Register Set

is 80 bits wide and is divided into "fields" corresponding to the NPX's temporary real data type.

At a given point in time the TOP field in the status word identifies the current top-of-stack register. A "push" operation decrements TOP by 1 and loads a

stores the value from the current top register and then increments TOP by 1. Like 80286 stacks in memory, the 80287 register stack grows "down" toward lower-addressed registers.

Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the TOP of the stack. These instructions implicitly address the register pointed by the TOP. Other instructions allow the programmer to explicitly specify the register which is to be used. This explicit register addressing is also "top-relative."

STATUS WORD

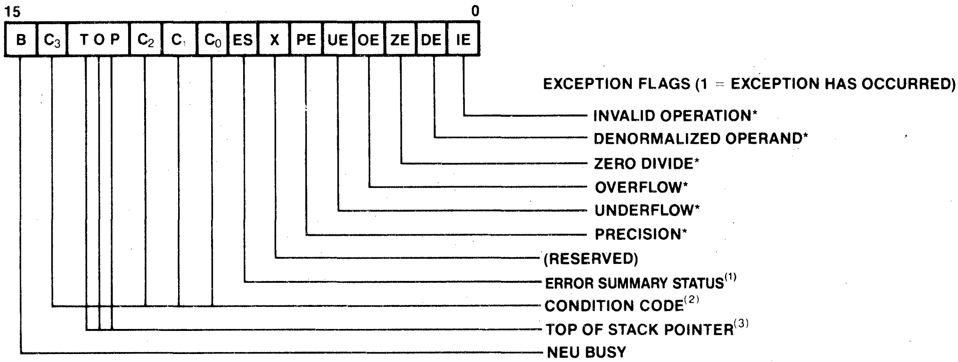
The 16-bit status word (in the status register) shown in Figure 6 reflects the overall state of the 80287. It may be read and inspected by CPU code. The busy bit (bit 15) indicates whether the NEU is executing an instruction (B = 1) or is idle (B = 0).

The instructions FSTSW, FSTSW AX, FSTENV, and FSAVE which store the status word are executed exclusively by the BIU and do not set the busy bit themselves or require the Busy bit be cleared in order to be executed.

similar to the flags in a CPU: instructions that perform arithmetic operations update these bits to reflect the outcome of NPX operations. The effect of these instructions on the condition code is summarized in Tables 4a and 4b.

Bits 14-12 of the status word point to the 80287 register that is the current top-of-stack (TOP) as described above. Figure 6 shows the six error flags in bits 5-0 of the status word. Bits 5-0 are set to indicate that the NEU has detected an exception while executing an instruction. The section on exception handling explains how they are set and used.

Bit 7 is the error summary status bit. This bit is set if any unmasked exception bit is set and cleared otherwise. If this bit is set, the ERROR signal is asserted.



(1) ES IS SET IF ANY UNMASKED EXCEPTION BIT IS SET, CLEARED OTHERWISE.
(2) SEE TABLE 5 FOR CONDITION CODE INTERPRETATION.
(3) TOP VALUES
000 = Register 0 is Top of Stack
001 = Register 1 is Top of Stack
•
•
•
111 = Register 7 is Top of Stack

*For definitions, see the section on exception handling

Figure 6. 80287 Status Word

TAG WORD

The tag word marks the content of each register as shown in Figure 7. The principal function of the tag word is to optimize the NPX's performance. The eight two-bit tags in the tag word can be used, however, to interpret the contents of 80287 registers.

INSTRUCTION AND DATA POINTERS

The instruction and data pointers (See Figures 8a and 8b) are provided for user-written error handlers. Whenever the 80287 executes a new instruction, the BIU saves the instruction address, the operand address (if present) and the instruction opcode. 80287 instructions can store this data into memory.

The instruction and data pointers appear in one of two formats depending on the operating mode of the 80287. In real mode, these values are the 20-bit physical address and 11-bit opcode formatted like the 8087. In protected mode, these values are the 32-bit virtual addresses used by the program which executed an ESC instruction. The same FLDENV/FSTENV/FSAVE/FRSTOR instructions as those of the 8087 are used to transfer these values between the 80287 registers and memory.

The saved instruction address in the 80287 will point at any prefixes which preceded the instruction. This is different than in the 8087 which only pointed at the ESCAPE instruction opcode.

CONTROL WORD

The NPX provides several processing options which are selected by loading a word from memory into the control word. Figure 9 shows the format and encoding of fields in the control word.

The low order byte of this control word configures the 80287 error and exception masking. Bits 5–0 of the control word contain individual masks for each of the six exceptions that the 80287 recognizes. The high order byte of the control word configures the 80287 operating mode including precision,

Table 4a. Condition Code Interpretation

Instruction Type	C_3	C_2	C_1	C_0	Interpretation
Compare, Test	0	0	X	0	ST > Source or 0 (FTST)
	0	0	X	1	ST < Source or 0 (FTST)
	1	0	X	0	ST = Source or 0 (FTST)
	1	1	X	1	ST is not comparable
Remainder	Q_1	0	Q_0	Q_2	Complete reduction with three low bits of quotient (See Table 5b)
	U	1	U	U	Incomplete Reduction
Examine	0	0	0	0	Valid, positive unnormalized
	0	0	0	1	Invalid, positive, exponent =0
	0	0	1	0	Valid, negative, unnormalized
	0	0	1	1	Invalid, negative, exponent =0
	0	1	0	0	Valid, positive, normalized
	0	1	0	1	Infinity, positive
	0	1	1	0	Valid, negative, normalized
	0	1	1	1	Infinity, negative
	1	0	0	0	Zero, positive
	1	0	0	1	Empty
	1	0	1	0	Zero, negative
	1	0	1	1	Empty
	1	1	0	0	Invalid, positive, exponent = 0
	1	1	0	1	Empty
	1	1	1	0	Invalid, negative, exponent = 0
	1	1	1	1	Empty

NOTES:
1. ST = Top of stack
2. X = value is not affected by instruction
3. U = value is undefined following instruction
4. Q_n = Quotient bit n

Table 4b. Condition Code Interpretation after FPREM Instruction As a Function of Dividend Value

Dividend Range	Q_2	Q_1	Q_0
Dividend < 2 * Modulus	C_3	C_1	Q_0
Dividend < 4 * Modulus	C_3	Q_1	Q_0
Dividend ≥ 4 * Modulus	Q_2	Q_1	Q_0

NOTE:
1. Previous value of indicated bit, not affected by FPREM instruction execution.

rounding, and infinity control. The precision control bits (bits 9–8) can be used to set the 80287 internal operating precision at less than the default of temporary real (80-bit) precision. This can be useful in providing compatibility with the early generation arithmetic processors of smaller precision than the 80287. The rounding control bits (bits 11–10) provide for directed rounding and true chop as well as the unbiased round to nearest even mode specified in the IEEE standard. Control over closure of the number space at infinity is also provided (either affine closure: ± ∞, or projective closure: ∞, is treated as unsigned, may be specified).

15							0
TAG (7)	TAG (6)	TAG (5)	TAG (4)	TAG (3)	TAG (2)	TAG (1)	TAG (0)

NOTE: The index i of tag(i) is not top-relative. A program typically uses the "top" field of Status Word to determine which tag(i) field refers to logical top of stack.

TAG VALUES:
00 = VALID
01 = ZERO
10 = INVALID or INFINITY
11 = EMPTY

Figure 7. 80287 Tag Word

15 — 0	MEMORY OFFSET
CONTROL WORD	+0
STATUS WORD	+2
TAG WORD	+4
IP OFFSET	+6
CS SELECTOR	+8
DATA OPERAND OFFSET	+10
DATA OPERAND SELECTOR	+12

Figure 8a. Protected Mode 80287 Instruction and Data Pointer Image in Memory

EXCEPTION HANDLING

The 80287 detects six different exception conditions that can occur during instruction execution. Any or all exceptions will cause the assertion of external $\overline{\text{ERROR}}$ signal and ES bit of the Status Word if the appropriate exception masks are not set.

The exceptions that the 80287 detects and the 'default' procedures that will be carried out if the exception is masked, are as follows:

Invalid Operation: Stack overflow, stack underflow, indeterminate form (0/0, ∞, $-\infty$, etc) or the use of a Non-Number (NAN) as an operand. An exponent value of all ones and non-zero significand is reserved to identify NANs. If this exception is masked, the 80287 default response is to generate a specific NAN called INDEFINITE, or to propogate already existing NANs as the calculation result.

Overflow: The result is too large in magnitude to fit the specified format. The 80287 will generate an encoding for infinity if this exception is masked.

Zero Divisor: The divisor is zero while the dividend is a non-infinite, non-zero number. Again, the 80287 will generate an encoding for infinity if this exception is masked.

Underflow: The result is non-zero but too small in magnitude to fit in the specified format. If this exception is masked the 82087 will denormalize (shift right) the fraction until the exponent is in range. The process is called gradual underflow.

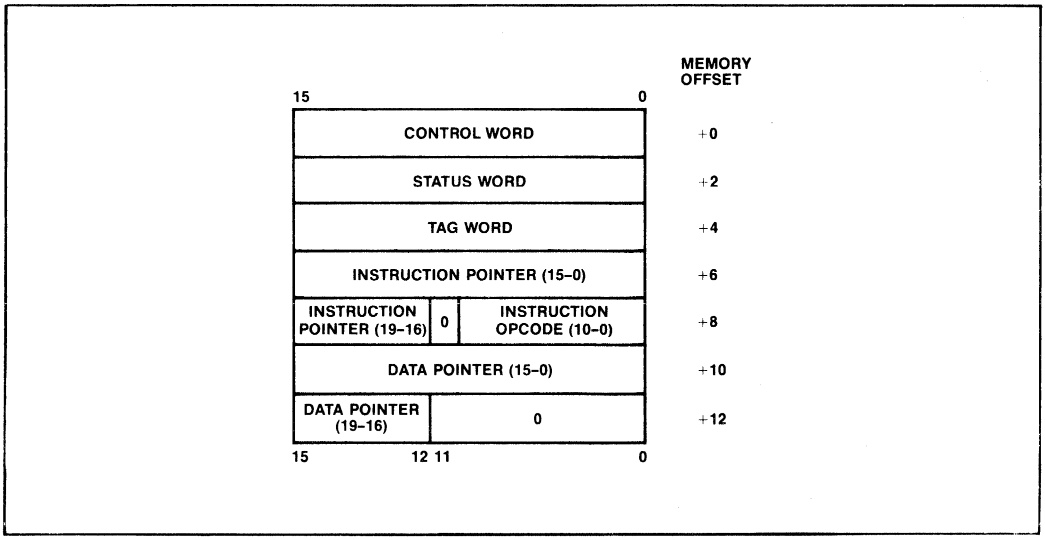


Figure 8b. Real Mode 80287 Instruction and Data Pointer Image in Memory

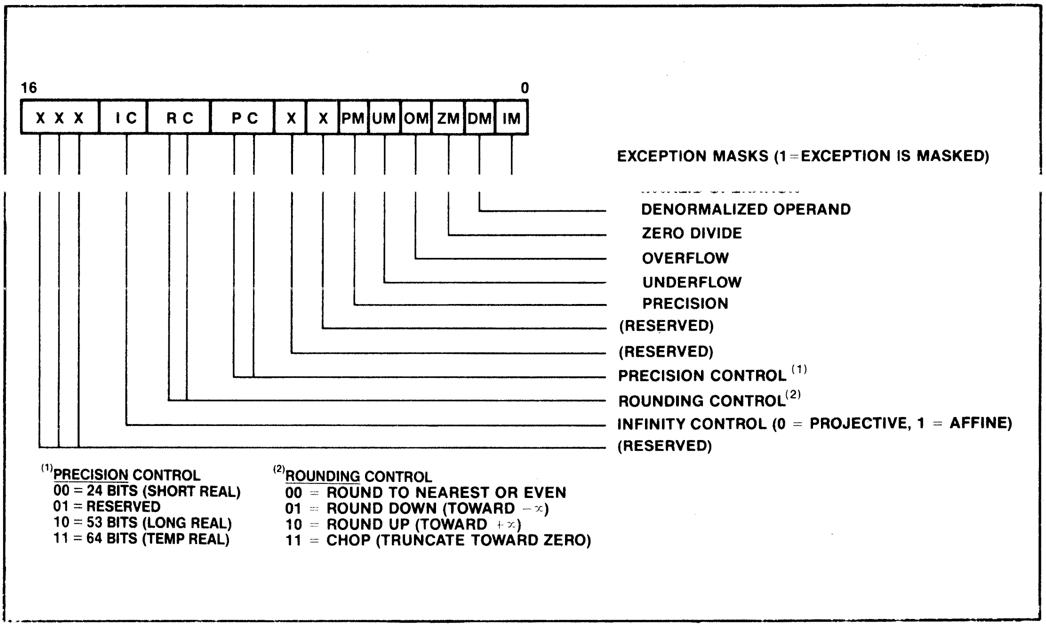


Figure 9. 80287 Control Word

Denormalized Operand: At least one of the operands is denormalized; it has the smallest exponent but a non-zero significand. Normal processing continues if this exception is masked off.

Inexact Result: The true result is not exactly representable in the specified format, the result is rounded according to the rounding mode, and this flag is set. If this exception is masked, processing will simply continue.

If the error is not masked, the corresponding error bit and the error status bit (ES) in the control word will be set, and the $\overline{\text{ERROR}}$ output signal will be asserted. If the CPU attempts to execute another ESC or WAIT instruction, exception 7 will occur.

The error condition must be resolved via an interrupt service routine. The 80287 saves the address of the floating point instruction causing the error as well as the address of the lowest memory location of any memory operand required by that instruction.

iAPX 86/20 COMPATIBILITY:

iAPX 286/20 supports portability of iAPX 86/20 programs when it is in the real address mode. However, because of differences in the numeric error handing techniques, error handling routines *may* need to be changed. The differences between an iAPX 286/20 and iAPX 86/20 are:

1. The NPX error signal does not pass through an interrupt controller (8087 INT signal does). Therefore, any interrupt controller oriented instructions for the iAPX 86/20 may have to be deleted.
2. Interrupt vector 16 must point at the numeric error handler routine.
3. The saved floating point instruction address in the 80287 includes any leading prefixes before the ESCAPE opcode. The corresponding saved address of the 8087 does not include leading prefixes.
4. In protected mode, the format of the saved instruction and operand pointers is different than for the 8087. The instruction opcode is not saved—it must be read from memory if needed.
5. Interrupt 7 will occur when executing ESC instructions with either TS or EM of MSW=1. If TS of MSW=1 then WAIT will also cause interrupt 7. An interrupt handler should be added to handle this situation.
6. Interrupt 9 will occur if the second or subsequent words of a floating point operand fall outside a segment's size. Interrupt 13 will occur if the starting address of a numeric operand falls outside a segment's size. An interrupt handler should be added to report these programming errors.

In the protected mode, iAPX 86/20 application code can be directly ported via recompilation if the 286 memory protection rules are not violated.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . 0°C to 70°C
Storage Temperature .... −65°C to +150°C
Case Temperature .... 0°C to 85°C
Voltage on Any Pin with
Respect to Ground .... −1.0 to +7V
Power Dissipation .... 3.0 Watt

**NOTICE: Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to 85°C, V_{CC} = 5V ± 5%
All Speed Selections

Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{IL}	Input LOW Voltage	−.5	.8	V	
V_{IH}	Input HIGH Voltage	2.0	V_{CC} + .5	V	
V_{ILC}	Clock Input LOW Voltage CKM = 1: CKM = 0:	 2.0 3.8	 V_{CC} + 1 V_{CC} + 1	 V V	
V_{OL}	Output LOW Voltage		.45	V	I_{OL} = 0.3 mA
V_{OH}	Output HIGH Voltage	2.4		V	I_{OH} = −400 μA
I_{LI}	Input Leakage Current	.	±10	μA	0V ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current	.	±10	μA	.45V ≤ V_{OUT} ≤ V_{CC}
I_{CC}	Power Supply Current	.	600 475 375	mA mA mA	T_A = 0°C T_A = 25°C T_A = 70°C
[illegible]	[illegible]	.	[illegible]	[illegible]	[illegible]
C_O	Input/Output Capacitance (D0-D15)	.	20	pF	F_C = 1 MHz
C_{CLK}	CLK Capacitance	.	12	pF	F_C = 1 MHz

210920-004

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, T_{CASE} = 0°C to 85°C, V_{CC} = 5V ± 5%)

TIMING REQUIREMENTS

A.C. timings are referenced to 0.8V and 2.0V points on signals unless otherwise noted.

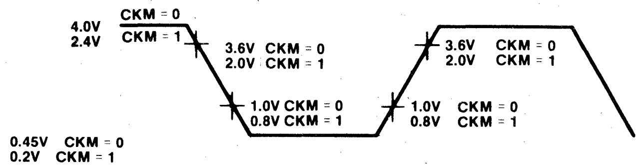
		6 MHz		8 MHz		10 MHZ Preliminary			
Symbol	**Parameter**	**-6 Min**	**-6 Max**	**-8 Min**	**-8 Max**	**-10 Min**	**-10 Max**	**Unit**	**Test Conditions**
T_{CLCL}	CLK Period								
	CKM = 1:	166	500	125	500	100	500	ns	
	CKM = 0:	62.5	166	50	166	40	166	ns	
T_{CLCH}	CLK LOW Time								
	CKM = 1:	100	343	68	343	62	343	ns	At 0.8V
	CKM = 0:	15	146	15	146	11	146	ns	At 0.6V
T_{CHCL}	CLK HIGH Time								
	CKM = 1:	50	230	43	230	28	230	ns	At 2.0V
	CKM = 0:	20	151	20	151	18	151	ns	At 3.6V
T_{CH1CH2}	CLK Rise Time		10		10		10	ns	1.0V to 3.6V if CKM = 1
T_{CL2CL1}	CLK Fall Time		10		10		10	ns	3.6V to 1.0V if CKM = 1
T_{DYWH}	Data Setup to NPWR Inactive	75		75		75		ns	
T_{WHDX}	Data Hold from NPWR Inactive	30		18		18		ns	
T_{WLWH} T_{RLRH}	NPWR NPRD Active Time	95		90		90		ns	At 0.8V
T_{AVRL} T_{AVWL}	Command Valid to NPWR or NPRD Active	0		0		0		ns	
T_{MHRL}	Minimum Delay from PEREQ Active to NPRD Active	130		130		100		ns	
T_{KLKH}	PEAK Active Time	85		85		60		ns	At 0.8V
T_{KHKL}	PEAK Inactive Time	250		250		200		ns	At 2.0V
T_{KHCH}	PEAK Inactive to NPWR, NPRD Inactive	50		40		40		ns	
T_{CHKL}	NPWR, NPRD Inactive to PEAK Active	-30		-30		-30		ns	
T_{WMAX} T_{RMAX}	Command Hold from NPWR, NPRD Inactive	30		30		22		ns	
T_{KLCL}	PEAK Active Setup to NPWR NPRD Active	50		40		40		ns	
T_{IVCL}	NPWR, NPRD to CLK Setup Time	70		70		53		ns	NOTE 1
T_{CLIH}	NPWR, NPRD from CLK Hold Time	45		45		37		ns	NOTE 1
T_{RSCL}	RESET to CLK Setup Time	20		20		20		ns	NOTE 1
T_{CLRS}	RESET from CLK Hold Time	20		20		20		ns	NOTE 1

210920-003

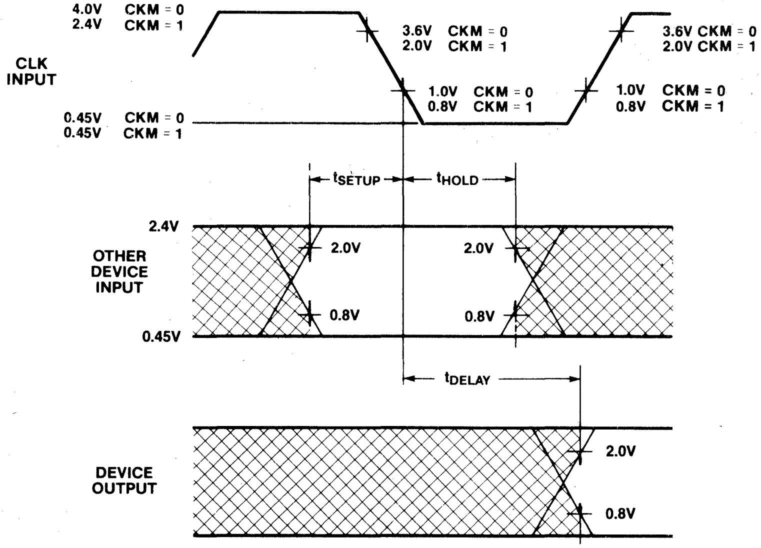
A.C. CHARACTERISTICS
TIMING RESPONSES

Symbol	Parameter	6 MHz -6 Min	6 MHz -6 Max	8 MHz -8 Min	8 MHz -8 Max	10 MHZ Preliminary -10 Min	10 MHZ Preliminary -10 Max	Unit	Test Conditions
T_{RHOZ}	NPRD Inactive to Data Float		37.5		35		25	ns	NOTE 2
T_{RLOV}	NPRD Active to Data Valid		60		60		60	ns	NOTE 3
T_{ILBH}	ERROR Active to BUSY Inactive	100		100		100		ns	NOTE 4
T_{WLBV}	NPWR Active to BUSY Active		100		100		100	ns	NOTE 5
T_{KLML}	PEACK Active to PEREQ Inactive		127		127		127	ns	NOTE 6
T_{CMDI}	Command Inactive Time								
	Write-to-Write	95		95			75	ns	At 2.0V
	Read-to-Read	95		95			75	ns	At 2.0V
	Write-to-Read	95		95			75	ns	At 2.0V
	Read-to-Write	95		95			75	ns	At 2.0V
T_{RHOH}	Data Hold from NPRD Inactive	5		3		3		ns	NOTE 7

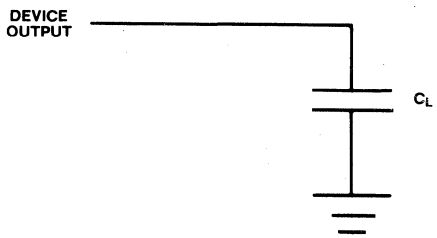
NOTES:
1. This is an asynchronous input. This specification is given for testing purposes only, to assure recognition at a specific CLK edge.
2. Float condition occurs when output current is less than I_{LO} on D0-D15.
3. D0-D15 loading: CL = 100pF
4. BUSY loading: CL = 100pF
5. BUSY loading: CL = 100pF
6. On last data transfer of numeric instruction.
7. D0-D15 loading: CL = 100pF



NOTE 8: AC Drive and Measurement Points — CLK Input



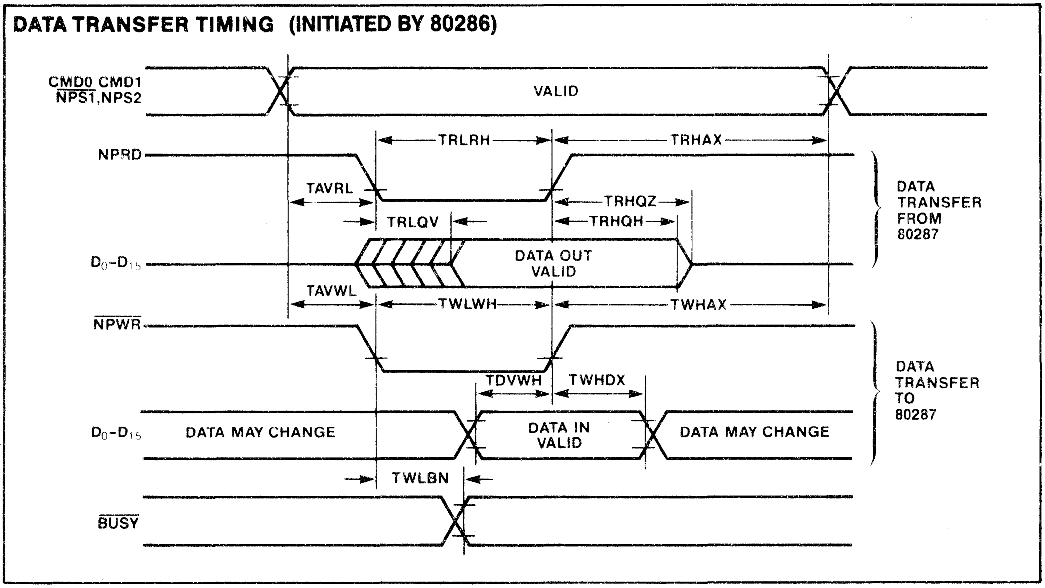
NOTE 9: AC Setup, Hold and Delay Time Measurement — General

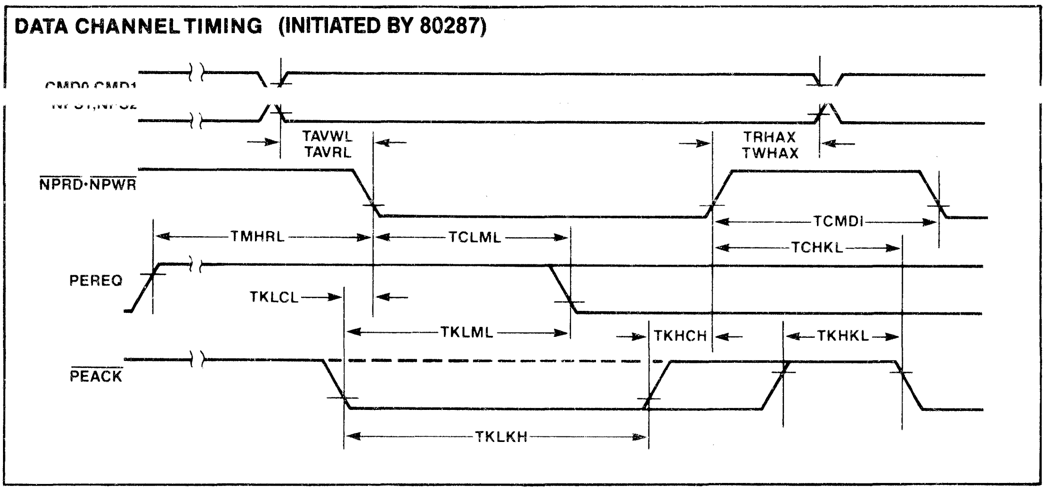


NOTE 10: AC Test Loading on Outputs

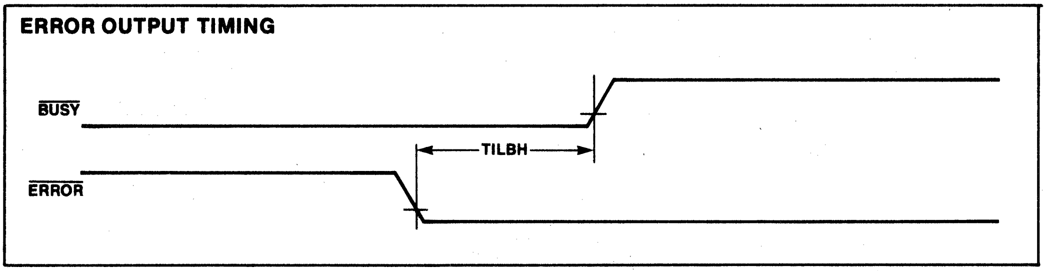
210920-004

WAVEFORMS (cont.)

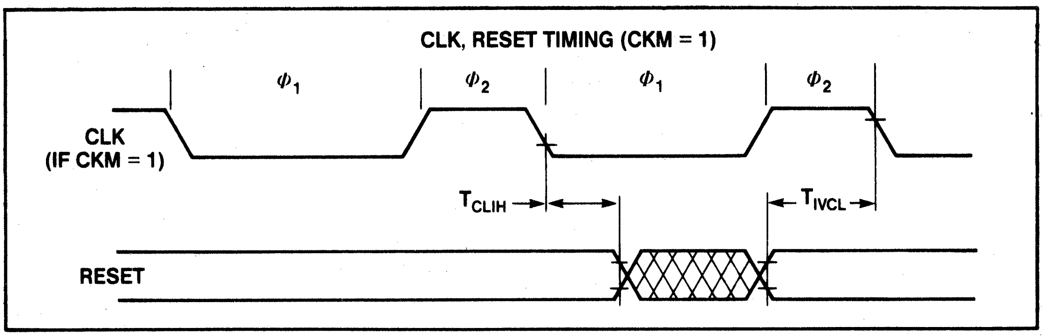




WAVEFORMS (cont.)

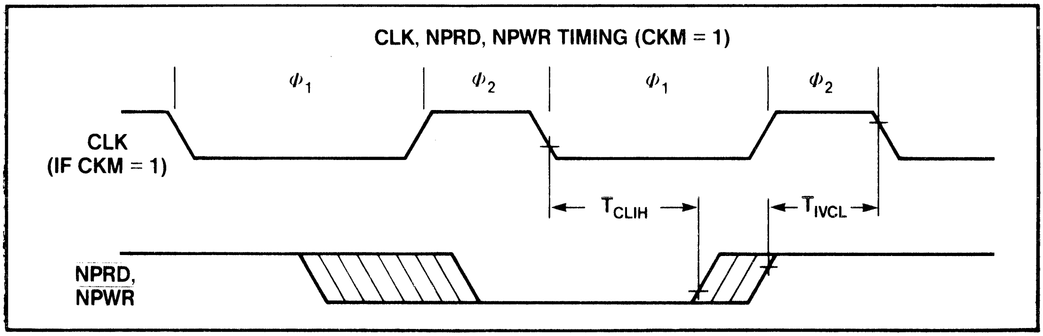


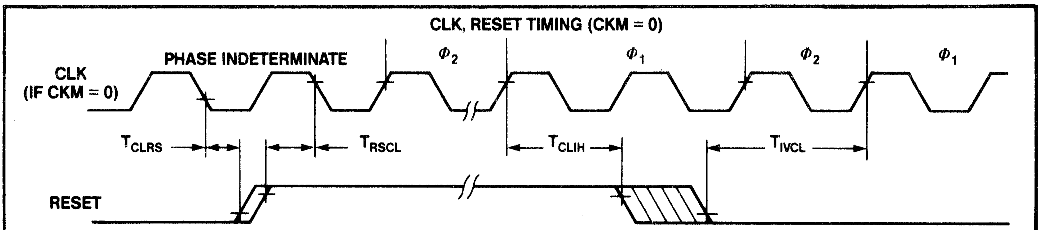
(Reset, NPWR, NPRD are inputs asynchronous to CLK. Timing requirements on this page are given for testing purposes only, to assure recognition at a specific CLK edge.)



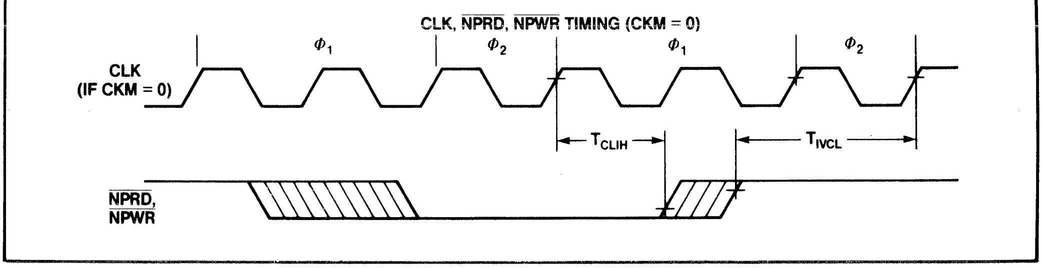
210920-004

WAVEFORMS (cont.)





NOTE: Reset must meet timing shown to guarantee known phase of internal ÷ 3 circuit.



210920-004

Table 6. 80287 Extensions to the 80286 Instruction Set

Data Transfer	Instruction Encoding		Optional 8,16 Bit Displacement	Clock Count Range 32 Bit Real	32 Bit Integer	64 Bit Real	16 Bit Integer
FLD = LOAD	MF			00	01	10	11
Integer/Real Memory to ST(0)	ESCAPE MF 1	MOD 0 0 0 R/M	DISP	38-56	52-60	40-60	46-54
Long Integer Memory to ST(0)	ESCAPE 1 1 1	MOD 1 0 1 R/M	DISP	60-68			
Temporary Real Memory to ST(0)	ESCAPE 0 1 1	MOD 1 0 1 R/M	DISP	53-65			
BCD Memory to ST(0)	ESCAPE 1 1 1	MOD 1 0 0 R/M	DISP	290-310			
ST(i) to ST(0)	ESCAPE 0 0 1	1 1 0 0 0 ST(i)		17-22			
FST = STORE							
ST(0) to Integer/Real Memory	ESCAPE MF 1	MOD 0 1 0 R/M	DISP	84-90	82-92	96-104	80-90
ST(0) to ST(i)	ESCAPE 1 0 1	1 1 0 1 0 ST(i)		15-22			
FSTP = STORE AND POP							
ST(0) to Integer/Real Memory	ESCAPE MF 1	MOD 0 1 1 R/M	DISP	86-92	84-94	98-106	82-92
ST(0) to Long Integer Memory	ESCAPE 1 1 1	MOD 1 1 1 R/M	DISP	94-105			
ST(0) to Temporary Real Memory	ESCAPE 0 1 1	MOD 1 1 1 R/M	DISP	52-58			
ST(0) to BCD Memory	ESCAPE 1 1 1	MOD 1 1 0 R/M	DISP	520-540			
ST(0) to ST(i)	ESCAPE 1 0 1	1 1 0 1 1 ST(i)		17-24			
FXCH = Exchange ST(i) and ST(0)	ESCAPE 0 0 1	1 1 0 0 1 ST(i)		10-15			
Comparison							
FCOM = Compare							
Integer/Real Memory to ST(0)	ESCAPE MF 0	MOD 0 1 0 R/M	DISP	60-70	78-91	65-75	72-86
ST(i) to ST (0)	ESCAPE 0 0 0	1 1 0 1 0 ST(i)		40-50			
FCOMP = Compare and Pop							
Integer/Real Memory to ST(0)	ESCAPE MF 0	MOD 0 1 1 R/M	DISP	63-73	80-93	67-77	74-88
ST(i) to ST(0)	ESCAPE 0 0 0	1 1 0 1 1 ST(i)		45-52			
FCOMPP = Compare ST(1) to ST(0) and Pop Twice	ESCAPE 1 1 0	1 1 0 1 1 0 0 1		45-55			
FTST = Test ST(0)	ESCAPE 0 0 1	1 1 1 0 0 1 0 0		38-48			
FXAM = Examine ST(0)	ESCAPE 0 0 1	1 1 1 0 0 1 0 1		12-23			

Table 6. 80287 Extensions to the 80286 Instruction Set (cont.)

Constants			Optional 8,16 Bit Displacement	Clock Count Range 32 Bit Real	32 Bit Integer	64 Bit Real	16 Bit Integer
	MF	=		00	01	10	11
FLDZ = LOAD + 0.0 into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 1 1 0		11–17			
FLD1 = LOAD + 1.0 into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 0 0 0		15–21			
FLDPI = LOAD π into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 0 1 1		16–22			
FLDL2T = LOAD $\log_2 10$ into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 0 0 1		16–22			
FLDL2E = LOAD $\log_2 e$ into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 0 1 0		15–21			
FLDLG2 = LOAD $\log_{10} 2$ into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 1 0 0		18–24			
FLDLN2 = LOAD $\log_e 2$ into ST(0)	ESCAPE 0 0 1	1 1 1 0 1 1 0 1		17–23			
Arithmetic							
FADD = Addition							
Integer/Real Memory with ST(0)	ESCAPE MF 0	MOD 0 0 0 R/M	DISP	90–120	108–143	95–125	102–137
ST(i) and ST(0)	ESCAPE d P 0	1 1 0 0 0 ST(i)		70–100 (Note 1)			
FSUB = Subtraction							
Integer/Real Memory with ST(0)	ESCAPE MF 0	MOD 1 0 R R/M	DISP	90–120	108–143	95–125	102–137
ST(i) and ST(0)	ESCAPE d P 0	1 1 1 0 R R/M		70–100 (Note 1)			
FMUL = Multiplication							
Integer/Real Memory with ST(0)	ESCAPE MF 0	MOD 0 0 1 R/M	DISP	110–125	130–144	112–168	124–138
ST(i) and ST(0)	ESCAPE d P 0	1 1 0 0 1 R/M		90–145 (Note 1)			
FDIV = Division							
Integer/Real Memory with ST(0)	ESCAPE MF 0	MOD 1 1 R R/M	DISP	215–225	230–243	220–230	224–238
ST(i) and ST(0)	ESCAPE d P 0	1 1 1 1 R R/M		193–203 (Note 1)			
FSQRT = Square Root of ST(0)	ESCAPE 0 0 1	1 1 1 1 1 0 1 0		180–186			
FSCALE = Scale ST(0) by ST(1)	ESCAPE 0 0 1	1 1 1 1 1 1 0 1		32–38			
FPREM = Partial Remainder of ST(0) ÷ST(1)	ESCAPE 0 0 1	1 1 1 1 1 0 0 0		15–190			
FRNDINT = Round ST(0) to Integer	ESCAPE 0 0 1	1 1 1 1 1 1 0 0		16–50			

NOTE:
1. If P=1 then add 5 clocks.

Table 6. 80287 Extensions to the 80286 Instruction Set (cont.)

			Optional 8,16 Bit Displacement	Clock Count Range
FXTRACT = Extract Components of St(0)	ESCAPE 0 0 1	1 1 1 1 0 1 0 0		27–55
FABS = Absolute Value of ST(0)	ESCAPE 0 0 1	1 1 1 0 0 0 0 1		10–17
FCHS = Change Sign of ST(0)	ESCAPE 0 0 1	1 1 1 0 0 0 0 0		10–17
Transcendental				
FPTAN = Partial Tangent of ST(0)	ESCAPE 0 0 1	1 1 1 1 0 0 1 0		30–540
FPATAN = Partial Arctangent of ST(0) ÷ ST(1)	ESCAPE 0 0 1	1 1 1 1 0 0 1 1		250–800
F2XM1 = $2^{ST(0)} - 1$	ESCAPE 0 0 1	1 1 1 1 0 0 0 0		310–630
FYL2X = ST(1) • Log_2 [ST(0)]	ESCAPE 0 0 1	1 1 1 1 0 0 0 1		900–1100
FYL2XP1 = ST(1) • Log_2 [ST(0) + 1]	ESCAPE 0 0 1	1 1 1 1 1 0 0 1		700–1000
Processor Control				
FINIT = Initialize NPX	ESCAPE 0 1 1	1 1 1 0 0 0 1 1		2–8
FSETPM = Enter Protected Mode	ESCAPE 0 1 1	1 1 1 0 0 1 0 0		2–8
FSTSW AX = Store Control Word	ESCAPE 1 1 1	1 1 1 0 0 0 0 0		10–16
FLDCW = Load Control Word	ESCAPE 0 0 1	MOD 1 0 1 R/M	DISP	7–14
FSTCW = Store Control Word	ESCAPE 0 0 1	MOD 1 1 1 R/M	DISP	12–18
FSTSW = Store Status Word	ESCAPE 1 0 1	MOD 1 1 1 R/M	DISP	12–18
FCLEX = Clear Exceptions	ESCAPE 0 1 1	1 1 1 0 0 0 1 0		2–8
FSTENV = Store Environment	ESCAPE 0 0 1	MOD 1 1 0 R/M	DISP	40–50
FLDENV = Load Environment	ESCAPE 0 0 1	MOD 1 0 0 R/M	DISP	35–45
FSAVE = Save State	ESCAPE 1 0 1	MOD 1 1 0 R/M	DISP	205–215
FRSTOR = Restore State	ESCAPE 1 0 1	MOD 1 0 0 R/M	DISP	205–215
FINCSTP = Increment Stack Pointer	ESCAPE 0 0 1	1 1 1 1 0 1 1 1		6–12
FDECSTP = Decrement Stack Pointer	ESCAPE 0 0 1	1 1 1 1 0 1 1 0		6–12

Table 6. 80287 Extensions to the 80286 Instruction Set (cont.)

			Clock Count Range
FFREE = Free ST(i)	ESCAPE 1 0 1	1 1 0 0 0 ST(i)	9–16
FNOP = No Operation	ESCAPE 0 0 1	1 1 0 1 0 0 0 0	10–16

NOTES:

1. if mod=00 then DISP=0*, disp-low and disp-high are absent
   if mod=01 then DISP=disp-low sign-extended to 16-bits, disp-high is absent
   if mod=10 then DISP=disp-high; disp-low
   if mod=11 then r/m is treated as an ST(i) field
2. if r/m=000 then EA=(BX) + (SI) +DISP
   if r/m=001 then EA=(BX) + (DI) +DISP
   if r/m=010 then EA=(BP) + (SI) +DISP
   if r/m=011 then EA=(BP) + (DI) +DISP
   if r/m=100 then EA=(SI) + DISP
   if r/m=101 then EA=(DI) + DISP
   if r/m=110 then EA=(BP) + DISP
   if r/m=111 then EA=(BX) + DISP

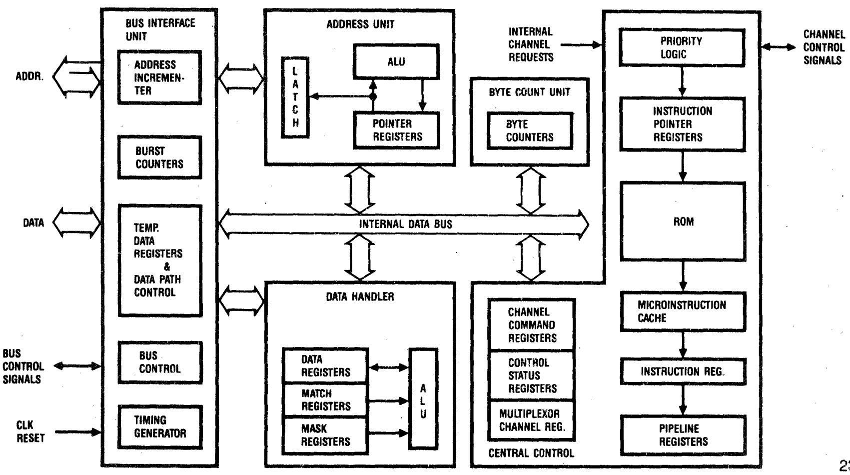
   *except if mod=000 and r/m=110 then EA =disp-high; disp-low.
3. MF= Memory Format
   00—32-bit Real
   01—32-bit Integer
   10—64-bit Real
   11—16-bit Integer
4. ST(0)= Current stack top
   ST(i) i^{th} register below stack top
5. d= Destination
   0—Destination is ST(0)
   1—Destination is ST(i)
6. P= Pop
   0—No pop
   1—Pop ST(0)
7. R= Reverse: When d=1 reverse the sense of R
   0—Destination (op) Source
8. For **FSQRT:** $-0 \leq ST(0) \leq +\infty$
   For **FSCALE:** $-2^{15} \leq ST(1) < +2^{15}$ and ST(1) integer
   For **F2XM1:** $0 \leq ST(0) \leq 2^{-1}$
   For **FYL2X:** $0 < ST(0) < \infty$
   $-\infty < ST(1) < +\infty$
   For **FYL2XP1:** $0 \leq |ST(0)| < (2 - \sqrt{2})/2$
   $-\infty < ST(1) < \infty$
   For **FPTAN:** $0 \leq ST(0) \leq \pi/4$
   For **FPATAN:** $0 \leq ST(0) < ST(1) < +\infty$
9. ESCAPE bit pattern is 11011.

82258
ADVANCED DIRECT MEMORY ACCESS COPROCESSOR (ADMA)

- **High Performance 16 Bit DMA Controller for the iAPX 286 Family**
  — 8 MByte/sec Maximum Transfer Rate in 8 MHz iAPX 286 Systems
- **Four Independently Programmable Channels**
- **Multiplexor Channel Capability to Support Up to 32 Subchannels**
- **On Chip Bus Interface for the Whole iAPX 86 Architecture**
  — 80286
  — 80186/188
  — 8086/88
- **Command Chaining for CPU Independent Processing**
- **Automatic Data Chaining for Gathering and Scattering of Data Blocks**
- **16 MByte Addressing Range**
- **16 MByte Block Transfer Capability**
- **"On the Fly" Compare, Translate and Verify Operations**
- **Automatic Assembly/Disassembly of Data**
- **Programmable Bus Loading**
- **6 and 8 MHz Speed Selections**
- **Available in 68-Pin LCC Package**
  (See Packaging Spec. Order #231369)

INTRODUCTION

Intel's 82258, Advanced Direct Memory Access Coprocessor is a high performance, 16 bit DMA processor optimized for the iAPX 286 and the iAPX 86 family of CPUs. It has on-chip bus interface for the whole iAPX 86 family architecture. Four high speed, independently programmable DMA channels can achieve a maximum cumulative transfer rate of 8 MByte/sec in an 8 MHz iAPX 286 system and 4 MByte/sec in 8 MHz iAPX 8086/80186 systems. Channel 3 can be used as a Multiplexor channel, whereby, it supports 32 subchannels. This flexibility allows one to use a single DMA channel to handle a large number of slow and medium speed I/O devices. Advanced capabilities like Command and Data chaining and "On the fly" operations allow the 82258 to remove the I/O management load from the processor. The 82258 addresses the full CPU memory (16 MB for 80286), thus simplifying the system design. Automatic assembly/disassembly of data allows 16 bit processors to interface with common 8 bit peripherals and vice-versa. Remote mode of operation, where the 82258 has its own resident bus, allows modular system design. The 82258 complements the high performance, multitasking capabilities of the 80286.



231263-1

Figure 1. 82258 Internal Block Diagram

Order Number: 231263-002

FABRICATION

The 82258 is a 68 pin device, fabricated in Intel HMOS II technology. It is packaged in JEDEC type A hermetic leadless chip carrier.

PIN DEFINITIONS AND FUNCTIONS

The 82258 has four operational modes

- — 286
- — 186—for the 80186/88 and the 8086/88 (Min. mode) CPUs
- — 8086—for the 8086/88 (Max. mode) CPUs
- — Remote

Pins of the 82258 have different definitions for different modes. 286 and remote modes have the same non-multiplexed bus structure and similar pin descriptions. Similarly, the 186 and the 8086 modes have multiplexed bus and similar pin description.

PINNING IN THE 286 MODE

In the 286 mode, the bus signals and the bus timings of the 82258 are the same as those of the 80286 processor. The processor can access the internal registers of the 82258 and these accesses must be supported by the bus signals. Therfore, some of the bus control signals are bidirectional and some additional bus control signals are necessary.

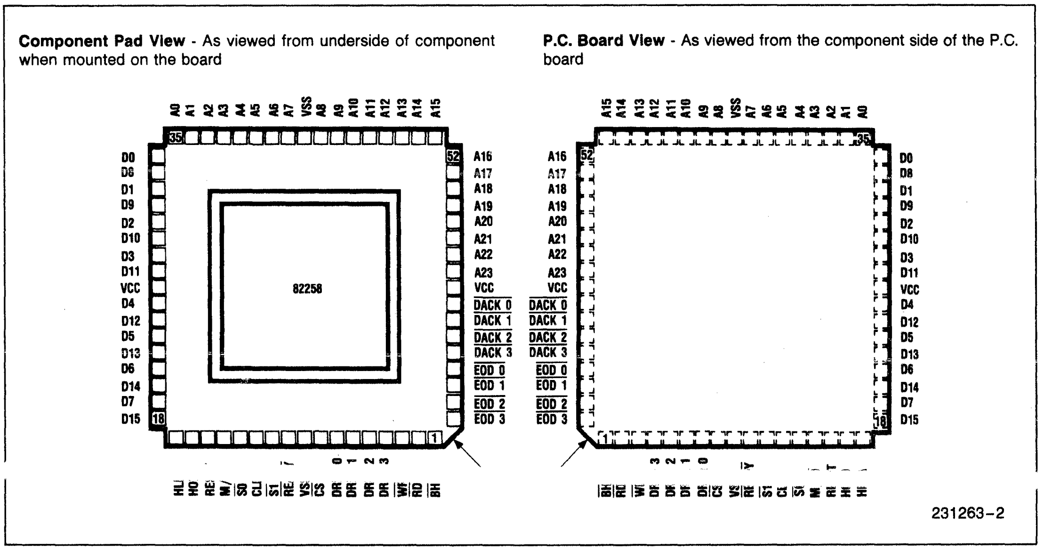


Figure 2. Pin Configuration in 286 Mode

Table 1. Pin Description for the 286 Mode (Also Contains Pins Identical in Other Modes)

<table>
<tr><th rowspan="2">Symbol</th><th colspan="2">Pin</th><th rowspan="2">Identical In All Modes</th><th rowspan="2">Functions</th></tr>
<tr><th>Type Input (I) Output (O)</th><th>Number</th></tr>
<tr><td>$\overline{BHE}$</td><td>I/O</td><td>1</td><td>YES</td><td>BUS HIGH ENABLE indicates transfer of data on the upper byte of the data bus, D15-8. Eight bit devices assigned to the upper byte of the data bus would normally use $\overline{BHE}$ to condition chip select function. $\overline{BHE}$ is active LOW and floats to Tri-State OFF when the 82258 does not own the bus.
$\overline{BHE}$ and A0 Encoding
<table>
<tr><th>$\overline{BHE}$ Value</th><th>A0 Value</th><th>Function</th></tr>
<tr><td>0</td><td>0</td><td>Word Transfer (D15-0)</td></tr>
<tr><td>0</td><td>1</td><td>Byte Transfer on upper half of data bus (D15-8)</td></tr>
<tr><td>1</td><td>0</td><td>Byte Transfer on lower half of data bus (D7-0)</td></tr>
<tr><td>1</td><td>1</td><td>Odd addressed byte on 8 bit bus (D7-0)</td></tr>
</table></td></tr>
<tr><td>$\overline{RD}$</td><td>I</td><td>2</td><td>NO</td><td>READ command in conjunction with chip select ($\overline{CS}$) enables reading out of the 82258 register, addressed by the address lines A7-A0. $\overline{RD}$ is an active LOW signal and is asynchronous to the 82258 clock.</td></tr>
<tr><td>$\overline{WR}$</td><td>I</td><td>3</td><td>NO</td><td>WRITE command along with $\overline{CS}$ is used for writing into the 82258 registers. $\overline{WR}$ is an active LOW signal and is asynchronous to the 82258 clock.</td></tr>
<tr><td>DREQ3-DREQ0</td><td>I</td><td>4—7</td><td>YES</td><td>DMA REQUEST input signals are used for externally synchronized DMA transfers. If channel 3 is used as a Multiplexor channel, DREQ3 is defined as I/O Request (IOREQ) signal. These signals are active HIGH signals and are asynchronous to the 82258 clock.</td></tr>
<tr><td>$\overline{CS}$</td><td>I</td><td>8</td><td>NO</td><td>CHIP SELECT is used to enable a processor to access the 82258 registers. This access is additionally controlled either by bus status signals or by the Read or Write command signals. $\overline{CS}$ is an active LOW signal, asynchronous to the 82258 clock.</td></tr>
<tr><td>$\overline{READY}$</td><td>I</td><td>10</td><td>NO</td><td>BUS READY terminates a bus cycle. Bus cycles are extended without limit until terminated by an active $\overline{READY}$. $\overline{READY}$ is an active LOW, synchronous input, requiring set up and hold times relative to system clock to be met for correct operation.</td></tr>
<tr><td>$\overline{S1}$, $\overline{S0}$</td><td>I/O</td><td>11,13</td><td>YES</td><td>BUS CYCLE STATUS signals control the support circuitry. The beginning of a bus cycle is indicated by $\overline{S1}$, or $\overline{S0}$, or both going active. The termination of a bus cycle is indicated by all the status signals going inactive in the 186 mode or the bus ready ($\overline{READY}$) going active in the 286 mode. Both $\overline{S0}$ & $\overline{S1}$ are active LOW signals. $\overline{S0}$, $\overline{S1}$ along with $\overline{S2}$ (in the 186 mode) or M/$\overline{IO}$ (in the 286 mode) define the type of bus cycle. $\overline{S2}$ and M/$\overline{IO}$ have the same meaning but, in the 186 mode $\overline{S2}$ signal can be active only when at least one of $\overline{S1}$ and $\overline{S0}$ is active, whereas in the 286 mode the M/$\overline{IO}$ signal is valid with the address on address lines.</td></tr>
</table>

Table 1. Pin Description for the 286 Mode (Also Contains Pins Identical in Other Modes) (Continued)

Symbol	Pin Type Input (I) Output (O)	Pin Number	Identical In All Modes	Functions
				The 82258 Bus Cycle Status Definitions (82258 Local Bus Master, All Signals (O)) — see tables below

The 82258 Bus Cycle Status Definitions
(82258 Local Bus Master, All Signals (O))

M/$\overline{IO}$ or $\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Bus Cycle Initiated
0	0	0	Read I/O-Vector (For Multiplexor channel)
0	0	1	Read from I/O space
0	1	0	Write into I/O space
0	1	1	None. (Does not occur in the 186 mode).
1	0	0	None. (Does not occur)
1	0	1	Read from memory space
1	1	0	Write into memory space
1	1	1	None; not a bus cycle

When the 82258 is not a bus master of the local bus, the status signals are used as inputs for detection of synchronous accesses to the 82258.

Interpretation of the Status and $\overline{CS}$ Signals by the 82258
(82258 Slave, All Signals (I))

$\overline{CS}$	$\overline{S1}$	$\overline{S0}$	Interpretation
1	X	X	82258 not selected (No action)
0	0	0	No 82258 access (No action)
0	0	1	Read from an 82258 register
0	1	0	Write into an 82258 register
0	1	1	Not a bus cycle*

*: The 82258 is selected but no synchronous access is activated. The 82258 monitors $\overline{RD}$ and $\overline{WR}$ signals for detection of an asynchronous access.

Symbol	Pin Type Input (I) Output (O)	Pin Number	Identical In All Modes	Functions
CLK	I	12	NO	**SYSTEM CLOCK** provides the fundamental system timing. It is divided by two to generate the 82258 internal clock. CLK is an active HIGH signal which can be connected directly to the 82284 CLK output. The internal divide-by-two circuitry is synchronized to the external clock generator by a LOW to HIGH transition on the RESET input, or by first HIGH to LOW transition on the Status Input $\overline{S0}$ or $\overline{S1}$ after RESET.
M/$\overline{IO}$	O	14	NO	**MEMORY/$\overline{IO}$ SELECT** distinguishes between memory and I/O space addresses.
RESET	I	15	YES	**SYSTEM RESET** forces the 82258 to the initial state. RESET is an active HIGH signal and must be synchronous to the system clock. Reset must be activated for at least 16 CLK cycles.

Table 1. Pin Description for the 286 Mode (Also Contains Pins Identical in Other Modes) (Continued)

Symbol	Pin Type Input (I) Output (O)	Pin Number	Identical In All Modes	Functions
HOLD HLDA	O I	16 17	NO	**BUS HOLD REQUEST AND HOLD ACKNOWLEDGE** control ownership of the local 82258 bus. When active, HOLD indicates a request for the control of the local bus. HOLD goes inactive when the 82258 relinquishes the bus. HLDA, when active, indicates that the 82258 can acquire the control of the bus. When HLDA goes inactive, the 82258 must relinquish the bus at the end of its current cycle. HLDA may be synchronous to the system clock. Both HOLD and HLDA are active HIGH signals.
D15-D0	I/O	18-25, 27-34	NO	**DATA BUS** is the bidirectional 16 bit bus. For use with an 8 bit bus, only the lower 8 data lines D7-D0 are relevant. The data bus is active HIGH.
A7-A0	I/O	35-42	NO	**ADDRESS LINES** A7-A0 are the lower 8 address lines for DMA transfers. They are also used to input the register address when the processor accesses an 82258 register. All lines are active HIGH.
A23-A8	O	44-59	NO	**ADDRESS LINES** A23-A8 form the remainder of the 82258 address bus. Address bus is active HIGH. ***Pin A21 must have a pullup resistor (n 10k Ω) connected to it to ensure that it is high during reset.***
$\overline{DACK3}$-$\overline{DACK0}$	O	61-64	YES	**DMA ACKNOWLEDGE** signal acknowledges the requests of the corresponding DREQ signal. $\overline{DACKi}$ goes active when the requested transfers are performed on the channel i in response to a DREQi. If channel 3 is in the multiplexor mode, $\overline{DACK3}$ is defined as I/O acknowledge (IOACK). These signals are active LOW.
$\overline{EOD3}$-$\overline{EOD0}$	I/O	65-68	YES	**END OF DMA** signals are open drain drivers with internal high impedance pull up resistors (no external pull up resistor is required) and can be used as quasi-bi-directional lines. These signals are active LOW. As OUTPUTs the signals are activated (if enabled) for two T-STATES cycles at the end of the DMA transfer of the corresponding channel or they are activated under program control (End of DMA output or interrupt output). $\overline{EODs}$ acts as "End of DMA" level triggered INPUTs if the signals are held high internally but forced low by the external circuitry for at least 250 ms. The current transfer is aborted and the 82258 continues with the next command. $\overline{EOD2}$ can also be used as a common active high interrupt signal (INTOUT) for all four channels. In this mode, this signal is a push-pull output and not an open drain output. Other EODi pins may still be used in their regular I/O mode.
V_{SS}	I	9, 43	YES	**SYSTEM GROUND:** 0 Volt.
V_{CC}	I	26, 60	YES	**SYSTEM POWER:** +5V Power Supply Pin.

PINNING IN THE 186 MODE

The 80186 has a multiplexed bus structure. Therefore, many 82258 pins have different meaning in the 186 mode than in the 286 mode. Since the 80186 has 20 address lines compared to 24 for the 80286, the 4 extra lines are used to generate additional bus control signals. The following table gives the details of pins having different meaning in the 186 mode compared to the 286 mode:

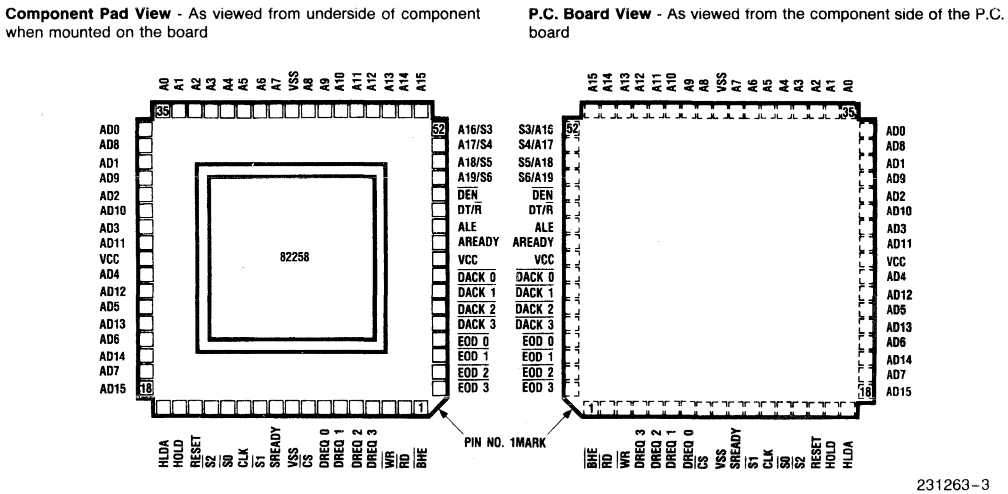


Figure 3. Pin Configuration in the 186 Mode

Table 2. Changes in Pin Description in the 186 Mode: (Compared to the 286 Mode)

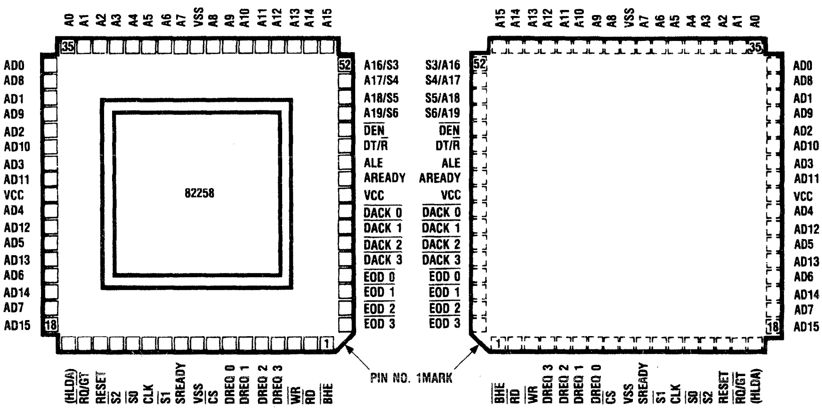
Symbol	Pin Type Input (I) Output (O)	Pin Number	Functions
$\overline{RD}$, $\overline{WR}$	I/O	2, 3	**READ, WRITE** In the 186 mode, the $\overline{RD}$ & $\overline{WR}$ pins are used additionally as output pins to support the 80186 or the 8086 minimum systems. These signals are active LOW.
ALE	O	58	**ADDRESS LATCH ENABLE** signal provides a strobe to separate the address information on the multiplexed address-data lines. ALE is an active HIGH signal.
$\overline{DEN}$	O	56	**DATA ENABLE** signal is used for enabling the data transceiver, 8286/8287. $\overline{DEN}$ is an active LOW signal.
DT/$\overline{R}$	O	57	**DATA TRANSMIT/RECEIVE** signal controls the direction of data flow through the external 8286/8287 data bus transceiver, depending on whether a read, or a write bus cycle is performed. ***This pin must have a pullup resistor connected to it to ensure that it is high during reset.***
SREADY	I	10	**SYNCHRONOUS READY** input signal must be synchronized externally. Use of this pin permits a relaxed system and timing specification by eliminating the clock phase, required for resolving the signal level, when using AREADY input. SREADY is an active HIGH signal.
CLK	I	12	**SYSTEM CLOCK** input gets a prescaled signal from the 186 clock (CLKOUT) or the 8086 clock (50% duty cycle for 186 and 33% duty cycle for 8086). No internal prescaling is done. CLK is an active HIGH signal.
$\overline{S2}$	O	14	**STATUS SIGNAL** along with $\overline{S0}$ and $\overline{S1}$ provides the bus cycle description (for details see 286 mode pin description of $\overline{S0}$ and $\overline{S1}$).
AD15–AD0 A7–A0 A15–A8	I/O I/O O	18–25 27–34 35–42 44–51	**ADDRESS/DATA BUS** signals AD15–AD0 contain multiplexed lower address and data information. Also, the demultiplexed address information is available on address pins A15–A0.
A19/S6 A16/S3	O	52–55	**ADDRESS PINS** A19–A16 are multiplexed with additional status information on the bus cycle. These pins are active HIGH. Signals S5 and S6 provide information on the status of the bus cycle. During an active bus cycle, S6 is always high and S5 always low. Low S6 implies a processor bus cycle. Signals S4 and S3 give the channel number for the running bus cycle as follows: S4 / S3 / Channel Number 0 / 0 / 0 0 / 1 / 1 1 / 0 / 2 1 / 1 / 3
AREADY	I	59	**ASYNCHRONOUS READY** is an asynchronous bus ready signal. The rising edge is internally synchronised. During reset, AREADY must be low to enter the 82258 into the 186 mode. AREADY is an active HIGH signal.

PINNING FOR THE 8086 MODE

For the 8086 MIN configuration the pinning is identical to the 186 mode. For the 8086 MAX configuration, the bus arbitration is done via the $\overline{RQ}/\overline{GT}$ protocol. Otherwise, the function of pins is identical to the 186 mode.

Component Pad View - As viewed from underside of component when mounted on the board

P.C. Board View - As viewed from the component side of the P.C. board



231263-4

Figure 4. Pin Configuration in the 8086 (Max) Mode

Table 3. Changes in Pin Description in the 8086 (Max) Mode
(Compared to the 186 Mode)

Symbol	Pin		Functions
	Type Input (I) Output (O)	Number	
$\overline{RQ}/\overline{GT}$	I/O	16	**REQUEST/GRANT** implements a one line communication protocol to arbitrate the use of the system bus; normally done via HOLD/HLDA. $\overline{RQ}/\overline{GT}$ is an active LOW signal having internal pull-up resistor.
HLDA	I	17	**HOLD ACKNOWLEDGE** has no meaning in the 8086 (Max) mode. It should be tied high, for mode recognition, during reset.

PINNING IN THE REMOTE MODE

In the remote mode, most of the signals have the same function as in the 286 mode. Exceptions are noted in the following table:

Component Pad View - As viewed from underside of component when mounted on the board

P.C. Board View - As viewed from the component side of the P. C. board

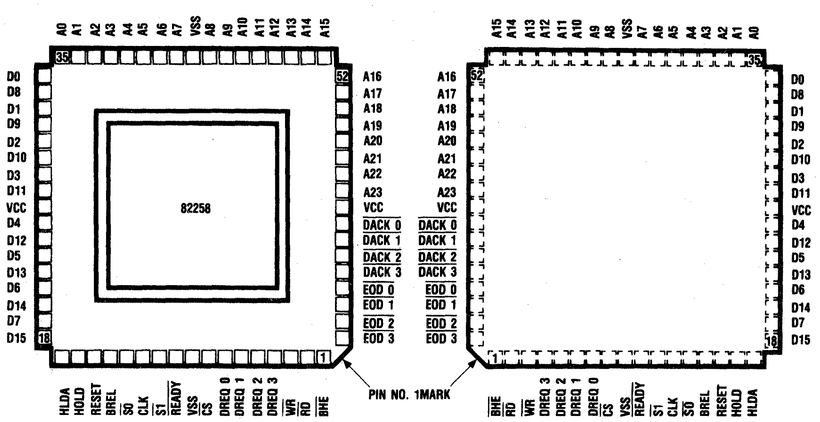


Figure 5. Pin Configuration in Remote Mode

Table 4. Changes in Pin Description in the Remote Mode (Compared to the 286 Mode)

Symbol	Pin: Type Input (I) Output (O)	Pin: Number	Functions
$\overline{CS}$	I	8	**CHIP SELECT** has two functions in the remote mode. As in the 286 mode, $\overline{CS}$ enables access to the 82258 internal registers. In addition $\overline{CS}$ works as an Access Request Input. When forced LOW, it signals to the 82258 that another bus master needs access to the local bus of the 82258. The 82258 releases the bus as soon as possible and signals it to the CPU by activating BREL (Bus Release) output. $\overline{CS}$ is an active LOW signal.
BREL	O	14	**BUS RELEASE** signal is used to indicate when the 82258 releases control of the resident bus.
HOLD HLDA	O I	16 17	**HOLD & HOLD ACKNOWLEDGE** signals are used only for access to the system bus. They are connected to the bus arbiter (82289). Resident bus accesses are directly executed without the HOLD/HLDA sequence.

FUNCTIONAL DESCRIPTION

The 82258 is an advanced DMA coprocessor for the iAPX 86 family architecture. In addition to providing high speed DMA transfers (8 MByte/sec in an 8 MHz iAPX 286 and 4 MByte/sec in 8 MHz iAPX 186/86 systems), the 82258 takes I/O processing load off the CPU, thus improving overall system performance. The 82258 has advanced featues not found in the previous generation DMA controllers: multiplexor channel, command & data chaining and 'on the fly' data manipulation operations.

MODES OF OPERATION

The 82258 has a number of different modes of operation based upon its coupling with the CPU (tight or loose) and its adaptive on-chip bus interface (the 286 bus or the 186 bus).

Figure 6 shows the different operating modes of the 82258 and the CPUs it can interface with in those modes. Figure 7 shows how to configure the 82258 into these different modes.

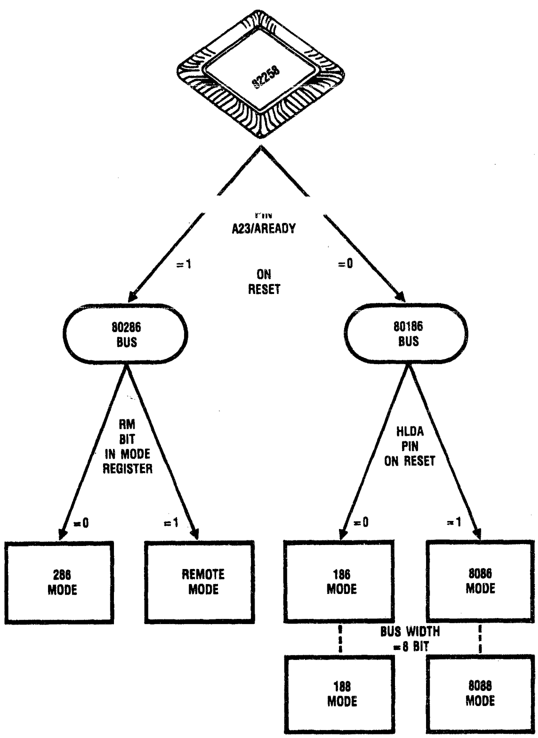
LOCAL MODE

In this mode the 82258 shares the local bus and all the support/control devices with the CPU. Because of its on-chip bus interface, the 82258 can be directly coupled to the whole iAPX 86 family of microprocessors.

CPU COUPLING	BUS INTERFACE: NON-MULTIPLEXED BUS	BUS INTERFACE: MULTIPLEXED BUS
LOOSE (REMOTE MODE)	80286 80186 80188 8086 8088	DOES NOT EXIST
TIGHT (LOCAL MODE)	80286 (286 MODE)	80186 80188 8086 8088 (186/86 MODE)

231263-52

Figure 6. Operating Modes for the 82258



231263-6

Figure 7. Selecting Modes of Operation

286 System

The configuration in Figure 8 shows the 82258 in the local mode (286 mode) in an iAPX 286 system which includes the Numeric Processor Extension 80287. The 286 mode is selected during reset (Figure 7). In this mode the 82258 supports the non-multiplexed, pipelined 286 bus. The DMA coprocessor resides on the processor's local bus (physical pins of the 80286) and shares all the support circuits: latches, tranceivers, bus controller and arbiter, clock generator etc. By residing on the 286 bus, the 82258 achieves maximum data transfer rate; up to 8 MByte/sec at 8 MHz for single cycle transfer. HOLD/ HLDA protocol is used for bus exchange between the 80286 and the 82258. The 82258 can be programmed to handle both internal and external terminate conditions. Internal termination is programmed in the command block (in type 2 command as explained later). External termination is handled by the $\overline{EOD}$ (end of DMA) pins if they are enabled. Interrupts for the CPU are handled by an interrupt controller (e.g. 8259A) which receives the end of DMA pins ($\overline{EOD}$ 0-3) as interrupts. The multiplexor channel uses external 8259As to prioritize and arbitrate service requests between peripherals (Figure 13).

To link this system to the MULTIBUS® bus architecture another set of latches, tranceivers, bus controllers and a bus arbiter (82289) as shown in Figure 11 (for remote mode configuration) are needed.

186/188 (8086/8088 Min) Systems

The 82258 can be configured into the 186 mode during reset (Figure 7). In this mode it supports the 80186 and the 8086 (Min) processors. It can be programmed to support the 80188 and the 8088 (Min) by programming the bus width in General Mode Register (GMR). Figure 9 shows the 82258 used in an iAPX 186 system containing the 8087 numeric coprocessor. This system uses the iAPX 86 bipolar support components: latches, transceivers and the bus controller (8288). The Integrated Bus Controller (82188) links the 80186 to the 8087. The 82188 is also used to support the 82258, since the 80186 has only one set of bus exchange signals (HOLD/ HLDA). An interrupt controller (8259A) processes the $\overline{EOD}$ signals for the CPU.

In the 186 mode, the 82258 directly supports the 80186/ 8086 bus with 16 address bits internally multiplexed into the data lines (AD15-AD0). The address pins A19-A16 are multiplexed with the status lines S6-S3. The address pins A22-A20 (in the 286 mode) are used to generate the control signals ALE, $\overline{DEN}$ and DT/$\overline{R}$ (in the 186 mode). The A23 pin (in the 286 mode) serves as an asynchronous ready input AREADY (in the 186 mode). As a master in the 186 mode, the 82258 offers address lines A15-A0 as latched outputs and shares all the 186/8086 support components with the processor.

8086/88 Systems

The 82258 is configured into the 8086 mode during reset (Figure 7). In this mode the 82258 supports 8086/88 in the maximum mode and uses the $\overline{RQ}/\overline{GT}$ protocol for the processor - DMA coprocessor bus exchange. The 8087 can be supported in the system without requiring the integrated bus controller, 82188. To support the 8088 system in the maximum mode, the General Mode Register is programmed for 8 bit bus width. Figure 10 shows the 82258 in an iAPX 86 system containing the 8087. The system configuration is very similar to the iAPX 186 system in Figure 9.

REMOTE MODE

The 82258 is configured to be in the Remote Mode (Figure 7) by programming the General Mode Register (RM bit), after putting the 82258 in the 286 mode during the reset. The 82258 has the bus timings and signals compatible to the 286 bus.

In the remote mode, the 82258 is the sole local bus (resident bus) master and interfaces to the processor through the system bus (using the bus arbiter 82289). Therefore, the 82258 can work in parallel with the processor. The remote mode is useful for a modular I/O subsystem.

Figure 11 shows the 82258 configured in the remote mode of operation. The peripherals interface to the 82258 on the resident bus. The resident bus components are similar to the ones used for the 286 system. Additional support components are used to interface the 82258 to a system bus e.g. the MULTIBUS. The 82258 communicates with the CPU (80286) over the system bus.

Since the 82258 is the only master of the local/resident bus, it can start the local bus cycles without any bus arbitration. For system bus accesses, a deadlock can arise if:

- The 82258 occupies the local bus to gain access to the system bus and
- The CPU (80286) occupies the system bus to gain access to the 82258 (through its local bus)

To prevent this deadlock, for the system bus accesses the 82258 does not occupy the local bus until it has the system bus. Therefore, in the remote mode, the 82258 initiates all system bus accesses (and only these) through the HOLD/HLDA protocol. The local bus arbitration (for the CPU) is done through the $\overline{CS}$ and the BREL lines.

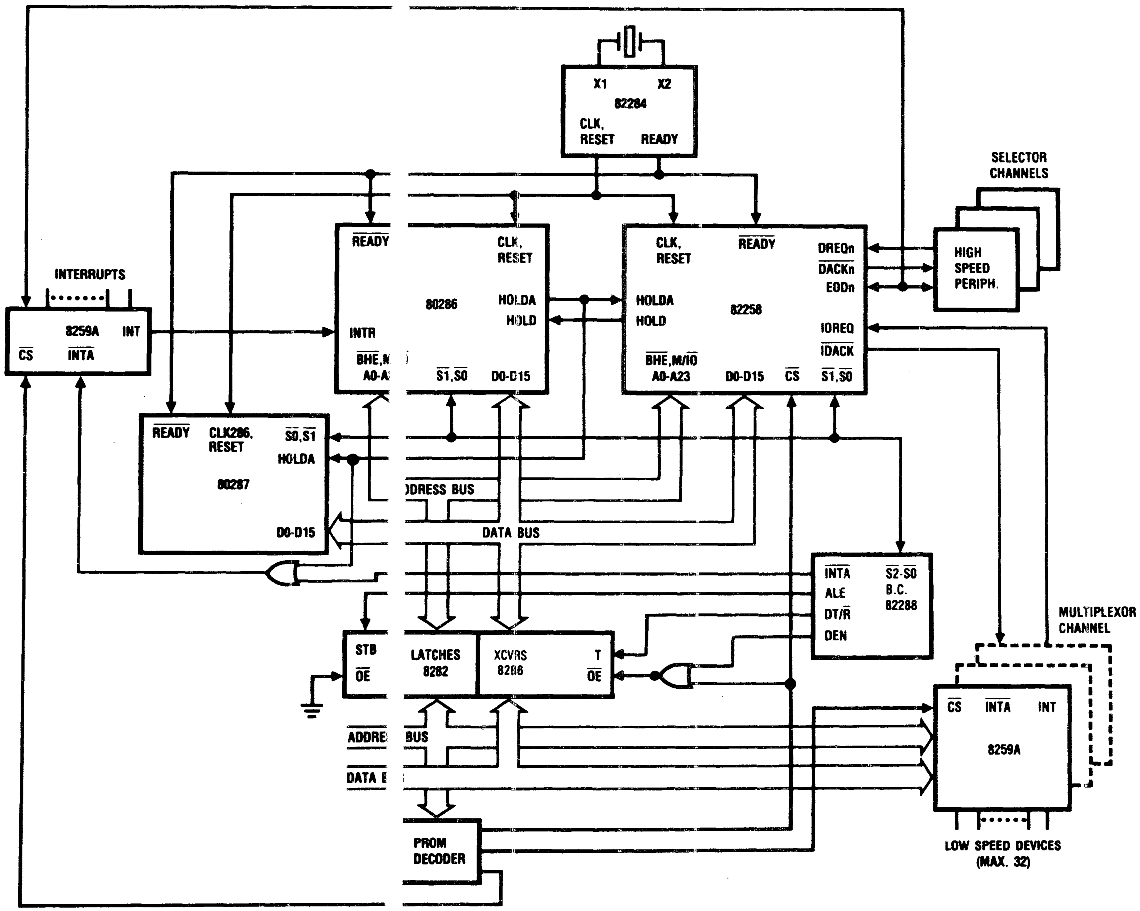


Figure 8. 82258 in an iAPX 286 System

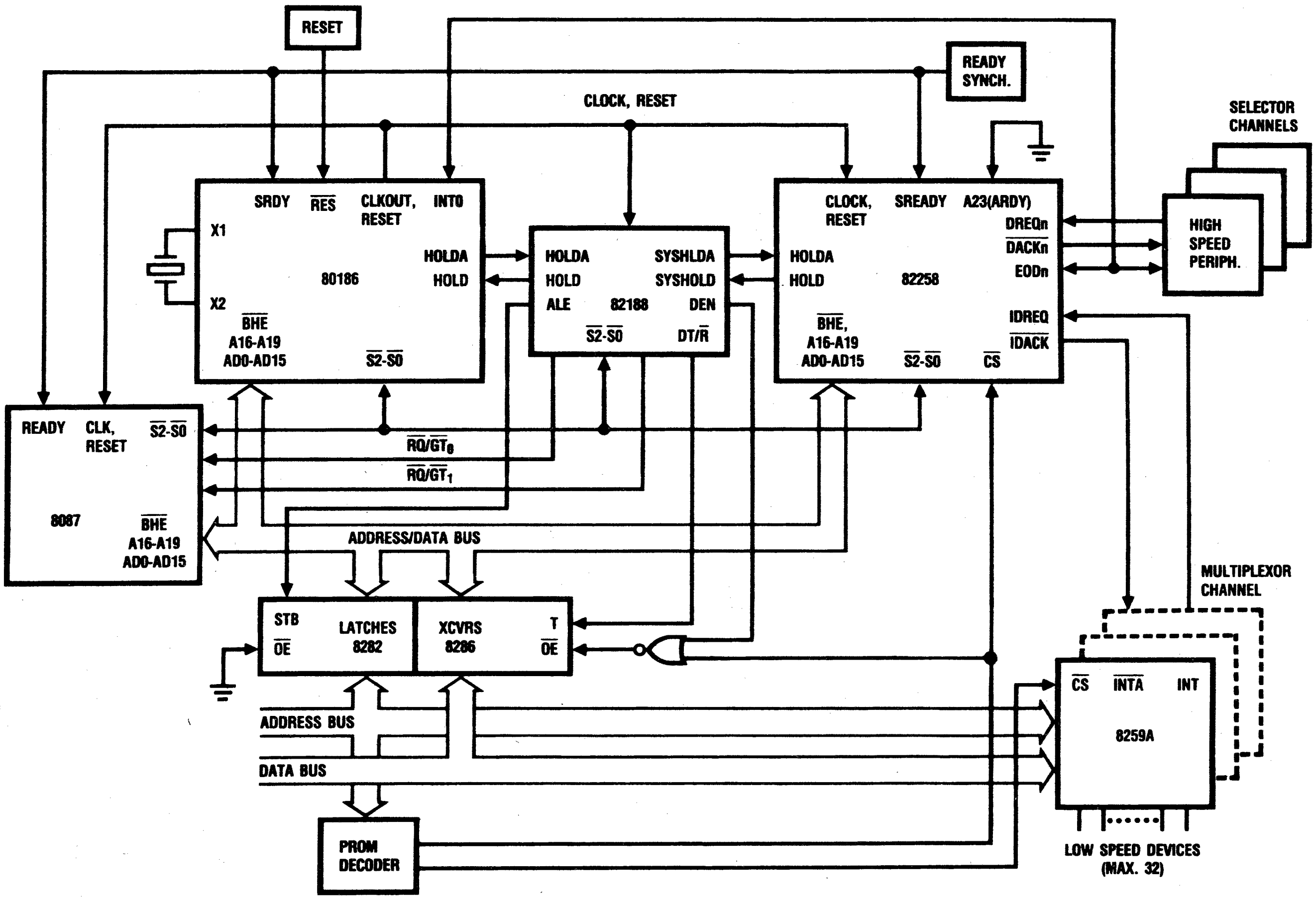


Figure 9. 82258 in an iAPX 186 System

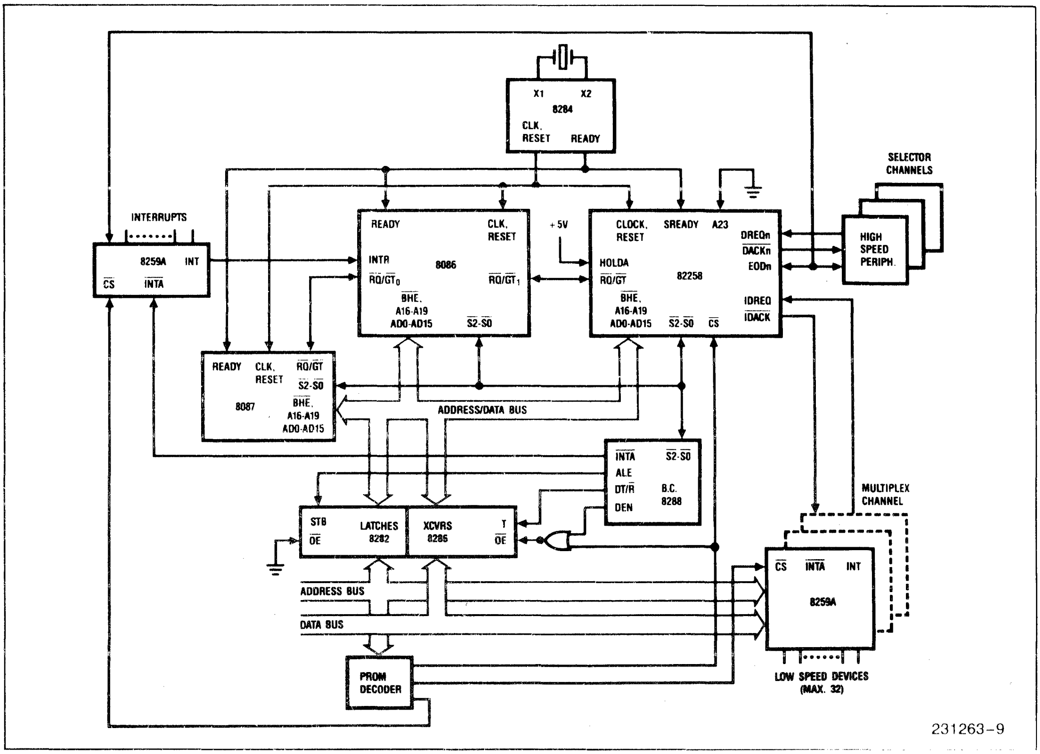


Figure 10. 82258 in an iAPX 86 System

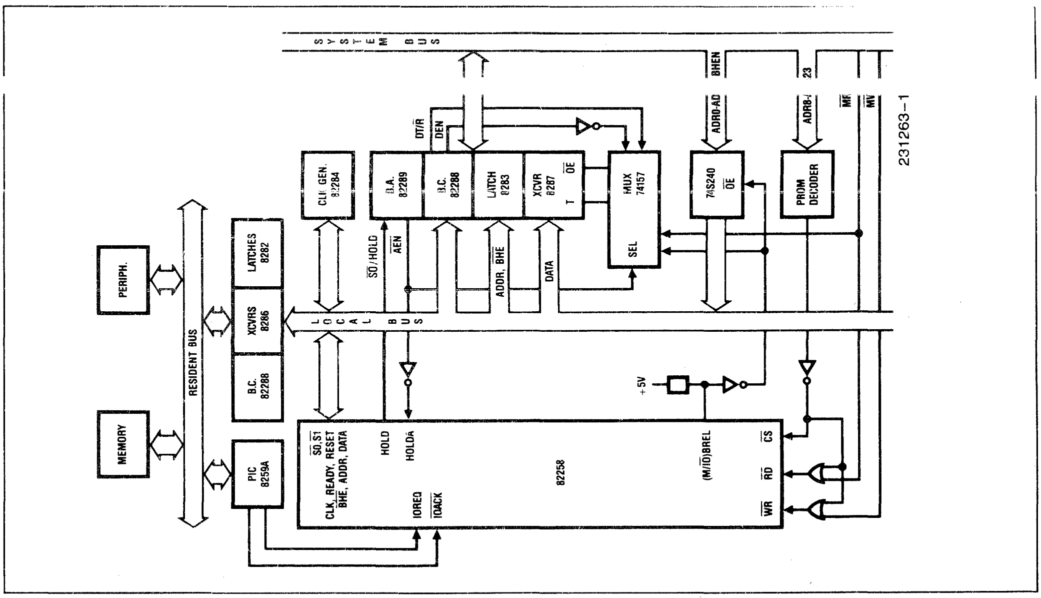


Figure 11. 82258 In Remote Mode

COMMUNICATION MECHANISMS

CPU → 82258 COMMUNICATION

Communication from the CPU to the 82258 is two-fold:

- Some 82258 registers receive the main commands from the CPU, through the slave interface of the 82258. Access to the 82258 is either synchronous (using $\overline{CS}$, $\overline{S1}$, $\overline{S0}$) or asynchronous (using $\overline{CS}$, $\overline{RD}$, $\overline{WR}$; $\overline{S1}$ = $\overline{S2}$ = 1).
- Most of the data is transferred via the control space in the memory in terms of organization blocks e.g. command blocks and multiplexor table. Control space can lie in the memory space or the memory mapped I/O space (system or resident space for the remote mode) and can be dynamically changed with every start channel command.

The CPU communicates with the 82258 by depositing data in the memory and into the on-chip registers of the 82258. The CPU can access the 82258 general registers and status registers, and can start a channel by writing the proper command to the general command register (GCR). The 82258 will then read the data from the memory command block and set itself up.

Slave Interface

The slave interface of the 82258 is used by the CPU to access the 82258 internal registers. Although most of the CPU to 82258 communication is done through memory based data blocks, some direct accesses to the 82258 registers are necessary. For example, during the initialization phase the general mode register (GMR) must be written to set up the 82258 or, to start a channel the command pointer registor (CPR) and the general command register (GCR) must be loaded. During the system debugging phase, access to the 82258 internal registers is very important.

The slave interface is enabled by the $\overline{CS}$ input and consists of the following lines:

$\overline{S1}$, $\overline{SO}$	—Status Lines (inputs)
$\overline{RD}$, $\overline{WR}$	—Control Lines (inputs)
A7–A0	—Register Address (inputs)
D15–D0	—Data Lines (inputs/outputs)-(for the 286 and the remote modes)
AD15–AD0	—Address/Data Lines (inputs/outputs)-(for the 186 and 8086 modes)

In the 286 mode and the 186/86 mode, two types of accesses are possible:

- synchronous access through the status lines $\overline{SI}$ and $\overline{S0}$
- Asynchronous access using $\overline{RD}$ and $\overline{WR}$

The register address must be supplied on the address pins A7–A0, except for the synchronous access in the 186/86 mode. Address data lines AD7–AD0 are used for the register address information in case of a synchronous access in the 186/86 mode.

In the remote mode, a synchronous access is not possible as the 82258 has to release its local bus to enable the CPU to access its registers. On receiving an access request ($\overline{CS}$ input asserted), the 82258 releases the local bus as soon as possible and signals it by asserting the BREL line. Only then, can the CPU access the 82258 registers.

82258 → CPU COMMUNICATION

The 82258 to the CPU communication is also two-fold:

- Hardware based communication, using one or more $\overline{EOD}$ lines as interrupt request lines to the CPU. The CPU can then read the status registers (and the interrupt vector register for the multiplexor channel) and service the interrupt.

— Control space based communication: At the end of a DMA transfer, the 82258 writes the contents of the appropriate channel status register into the channel command block. Additionally, it may transfer some other information (e.g. the updated source pointer) into the command status block.

The 82258 updates its internal registers (e.g. the channel command pointer, the general status register etc.) for any CPU access.

82258 — PERIPHERAL COMMUNICATION

The DMA interface of the 82258 is used for its communication with the peripherals. It consists of three signal lines:

$\overline{\text{DREQ}}$ —DMA Request

$\overline{\text{DACK}}$ —DMA Acknowledge

$\overline{\text{EOD}}$ —End of DMA

DREQ and $\overline{\text{DACK}}$ control the externally synchronized DMA transfers. A burst of data is transferred for a continuous DMA request, as long as the request signal is active.

$\overline{\text{EOD}}$ lines, which are quasi-bidirectional, enhance the 82258—Peripheral communication link. First these can be used as inputs to the 82258 to receive an asynchronous external terminate signal to terminate a running DMA. As outputs, they can be used to interrupt the CPU and/or to signal a specific status to the peripheral (e.g. transfer aborted or, end of a block or, send/receive next block..). In addition, the $\overline{\text{EOD}}$ output of channel 2 can be used as a collective interrupt output (INTOUT) for all the DMA channels while the other three $\overline{\text{EOD}}$ lines retain their normal function.

An $\overline{\text{EOD}}$ output signal can be generated synchronous to a synchronising device at the last data transfer or, synchronous to the internal clock at the last destination cycle. An $\overline{\text{EOD}}$ can also be generated asynchronously through a Type 2 command.

BUS ARBITRATION

HOLD/HLDA Sequence

These signals are used for the bus arbitration in the 286 mode and the 186/88 (8086/88 Min.) mode. Whenever the 82258 needs the bus, it activates the HOLD signal and the processor surrenders the local bus as soon as possible by asserting HLDA. The 82258 performs the transfer and switches the HOLD to low. The processor takes the bus and switches the HLDA to low. To force the 82258 to surrender the bus, the HLDA must be set to low. The 82258 will release the bus after the currently running bus cycle or the unseparable bus cycles. Unseparable bus cycles are:

— The two IO acknowledge bus cycles for the 8259A PIC.

— Word transfers on odd boundary addresses, realised by two bus cycles where each transfer is a byte.

— Fetch of 24 bit address pointers out of the memory or restore of the pointers.

— Read- modify- write the 8259A mask registers.

The 82258 signals the surrendering of the bus by floating the bus and removing the HOLD signal. If requests for bus cycles are present, the HOLD will go active after a delay of two T-states.

$\overline{\text{RQ}}/\overline{\text{GT}}$ Sequence

$\overline{\text{RQ}}/\overline{\text{GT}}$ protocol is used for the 8086/88 (Max.) Mode. The 82258 requests the bus by sending a request pulse of one CLK period length, via the $\overline{\text{RQ}}/\overline{\text{GT}}$ signal, to the processor. The processor acknowledges it with a pulse on the same line. Then the 82258 controls the bus. When surrendering the bus, it sends a release pulse on the $\overline{\text{RQ}}/\overline{\text{GT}}$ line.

$\overline{\text{CS}}$/BREL Sequence

This is used in the remote mode along with the HOLD/HLDA signals. HOLD/HLDA are used for system bus arbitration and $\overline{\text{CS}}$/BREL for local bus

registers or the resident bus). The CPU asserts the $\overline{\text{CS}}$ signal to ask for the local bus and the 82258 releases the bus as soon as possible by activating BREL. After the CPU has completed its access, it should set $\overline{\text{CS}}$ high. The 82258 deactivates BREL and proceeds with its own bus cycles on the local bus.

NOTE:
When the 82258 is not in possession of the bus, all output signals are tristated except the following:
HOLD (except in the $\overline{\text{RQ}}/\overline{\text{GT}}$ protocol), $\overline{\text{DACK0}}$–$\overline{\text{DACK3}}$, $\overline{\text{EOD0}}$–$\overline{\text{EOD3}}$,
BREL (remote mode) and ALE (186 mode)

CHANNEL CONFIGURATION

The 82258 has four independently programmable DMA channels with their own register sets. All channels can be used as high speed selector channels for achieving maximum transfer rate or channel 3 can be used as a multiplexor channel to allow the 82258 to interface to a large number of I/O devices.

The selector channels support synchronised and non synchronised transfers as well as advanced features like single cycle transfer, command and data chaining. Channel switching imposes no performance penalty on the 82258. Programmable priority schemes allow flexible multiple channel processing.

MULTIPLEXOR CHANNEL

Channel 3 of the 82258 can also be operated as a multiplexor channel supporting up to 32 subchannels. External 8259As are used to arbitrate and prioritize channel requests (Figure 13). Multiplexor channel allows command chaining but data chaining is not supported.

As a multiplexor channel, channel 3 uses an external multiplexor table (MT) in the memory to store separate command pointers and, the PIC (8259A) mask register locations for each device in that channel. Each entry in the MT consists of 8 bytes; the first 4 give the command pointer for the subchannel and the second 4 the address of the mask register of the 8259A for that subchannel (Figure 14).

After an I/O request from the 8259A, the 82258 fetches an 8 bit vector (device number) from the interrupt controller (by the INT/INTA mechanism), left shifts it by three and, uses that as an offset into the multiplexor table with that entry pointing to the current subchannel command block. The 8259A should be programmed for AEOI mode.

Each subchannel can have a subchannel program or a command chain. The command chain must be terminated by a stop and mask command (as opposed to a stop command for a selector channel). Three kinds of data transfers are possible:

Byte/Word Multiplex: One byte/word is transferred per request. The source/destination pointer and the byte count fields of the command block are updated. The command pointer is not advanced until the block transfer is terminated. Maximum cumulative data transfer rate of 275K Bytes/sec can be achieved for the channel.

Single Transfer: Similar to the byte/word multplex. But, the command pointer is advanced after each transfer, thus, executing command chaining.

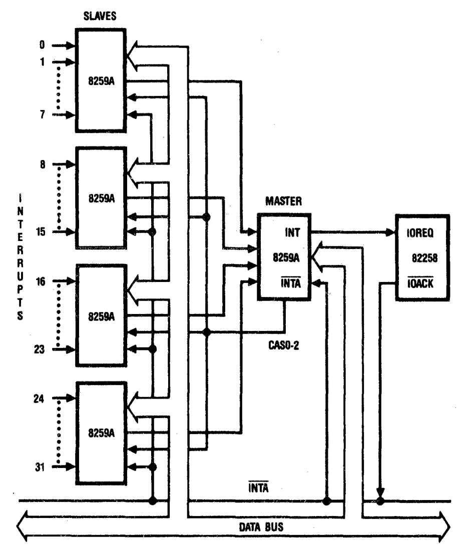


Figure 13. Multiplexor Configuration

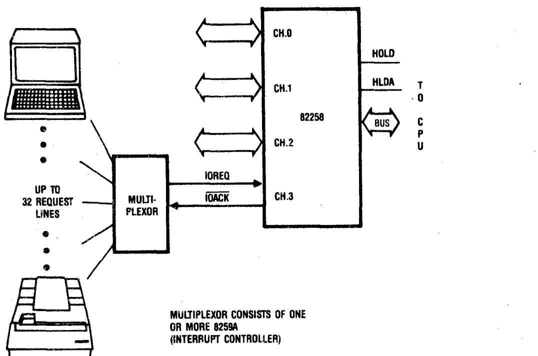


Figure 12. 82258 Channel Configuration

Block Multiplex Transfer: The whole command block is executed and a block transfer made upon receiving a request. Such transfer is necessarily free running or non-synchronised and is carried out at a maximum speed of 4 MByte/sec in an 8 MHz iAPX286 system. After termination, the command pointer is advanced (command chaining).

The type 2 commands have the same function as for the selector channels (Table 6). A subchannel is stopped with a stop and mask command which must occur at the end of a command block chain. The 82258 generates the interrupt (INTOUT) or $\overline{EOD}$, if programmed. The 82258 automatically masks the request line on the 8259A by setting its mask bit. Thus no further requests can come from this subchannel until it is enabled by the CPU. The 82258 indicates the interrupted subchannel (vector) in the Multiplexor Channel Interrupt Vector Register (MIVR). The MIVR can be accessed by the CPU and, after reading the MIVR, the stop bit of the indicated subchannel is reset. If no channel 3 interrupt (EOD or programmed INTOUT) is enabled, the internal interrupt flag is set by the stop and mask command. Then the CPU checks the MIVR by polling, i.e., with each reference of this register, the CPU can read off the stopped subchannel vector that has the highest priority in queue until the NV (vector is not valid) bit in MIVR is set.

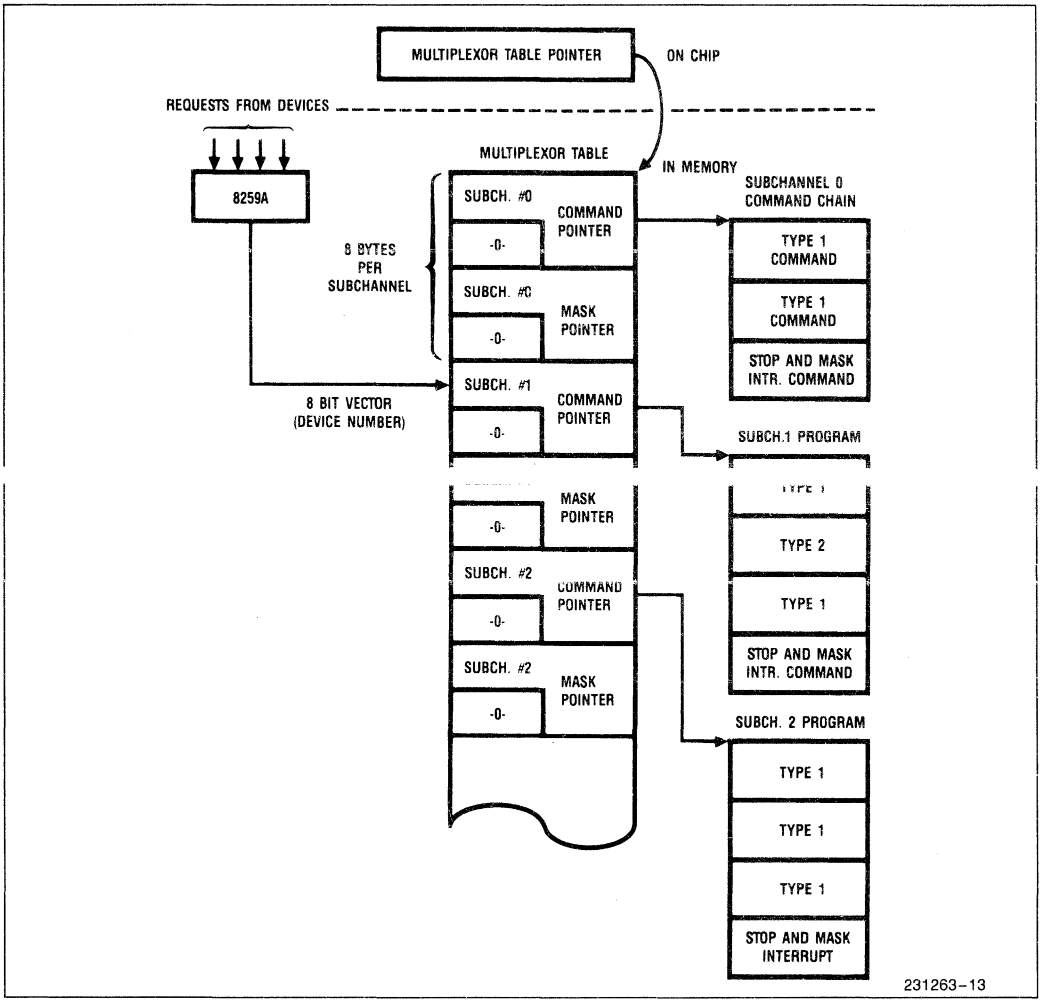


Figure 14. Multiplexor Table

DATA TRANSFER AND MANIPULATION CONTROL

SINGLE CYCLE AND TWO CYCLE TRANSFERS

The 82258 provides the flexibility to optimize the system design by allowing:

— Highest speed DMA transfers in the single cycle transfer mode. In this mode bytes or words (16 bits) are transferred directly from the source to the destination without storing the data in the 82258 registers (Figure 15). The single cycle transfer mode does not, necessarily, mean one bus cycle for transfer (though most of the transfers require either a source or a destination data cycle only). Maximum single channel or multiple channel transfer rate of 8 MByte/sec. in an 8 MHz iAPX 286 system (4 MByte/sec in 8 MHz iAPX 186/86 systems) is achieved in this mode.

In the single cycle transfer mode, while the requesting device is serviced (and addressed) using $\overline{\text{DACK}}$ signal, the pointer to the other location (memory or I/O) is issued and its bus cycle executed by the 82258. It is the duty of the I/O device to know whether the cycle is a read cycle or a write cycle and, to generate its command signal out of the bus command signals.

Single cycle transfers mode is not allowed for the multiplexor channel. All single cycle transfer are externally synchronised and "On the fly" operations are restricted (see Table 5).

— Maximum data manipulation operations in the two cycle transfer mode. The two cycle transfer mode does not, necessarily, imply two bus cycles, though most of the transfers consist of a fetch cycle from the source and a store cycle to the destination location. In this mode the source data is always stored in the 82258 registers before being sent out to the destination. Although half as

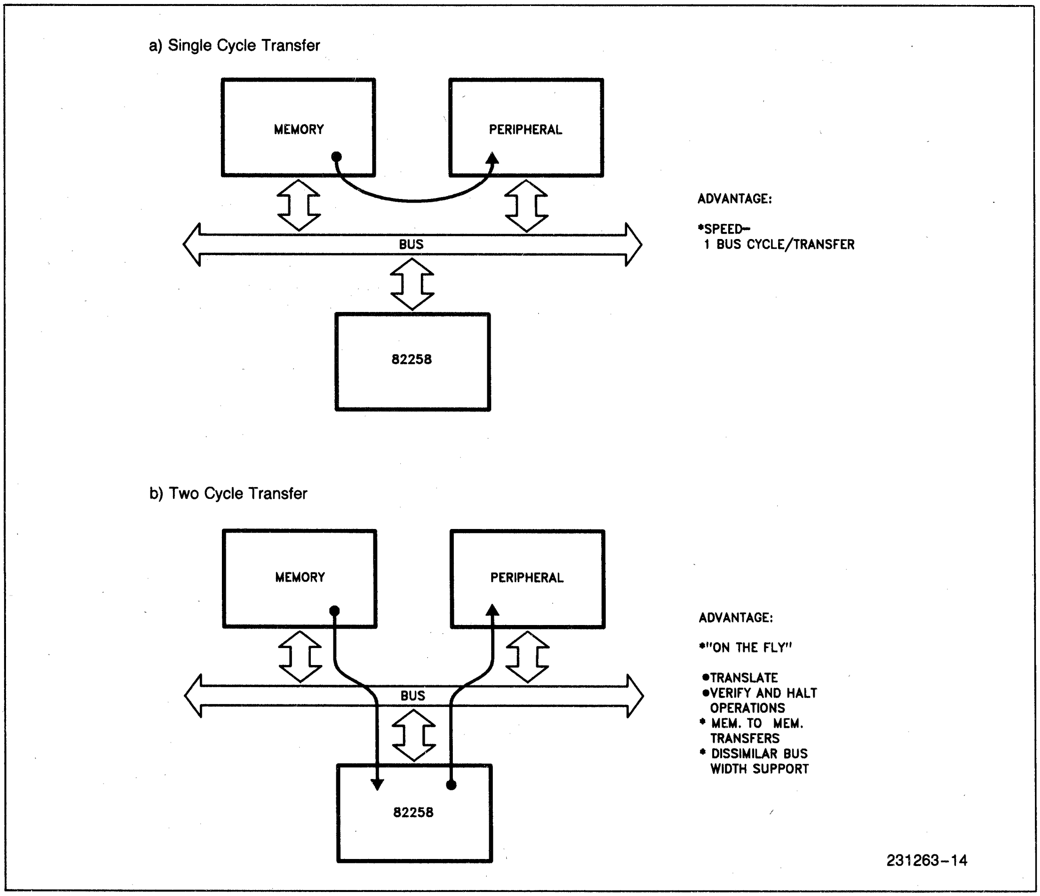


Figure 15. Single/Two Cycle Transfer

fast as the single cycle mode, a number of "On the fly" operations e.g., translation, make this mode extremely versatile. The two cycle transfer mode also allows automatic assembly and disassembly of the data, i.e., the data can be read as one 16 bit word and written as 2 bytes or vice-versa. It is useful for linking the 8 bit peripherals to a 16 bit system and vice-versa.

The two cycle transfer mode allows multiplexor channel operation and memory to memory transfers. Two special cases of two cycle data transfer are:

Read Operation or, data transfer without a destination address (the data assembly register of the 82258 itself is the destination of the source data). Compare operations on the source data are possible (e.g. to test the status of a disk controller).

Write Operation or, data transfer with no source address i.e., the source data is a byte or word constant (literal) in the data assembly register of the 82258 (loaded during the setup routine with a low word out of the source pointer field). The write operation can be used to erase a memory/peripheral data block (or peripheral register) or to load it with a certain constant.

Table 5. Data Manipulation Operations

Operation	Single Cycle	Two Cycle	Multiplexor Channel*
	Bus Cycles Required**		
Masked Compare (Byte/Word)	2	2	2
Verify	N/A	2	1
Verify and Halt	N/A	2	N/A***
Verify and Save	2	F	F
Translate	F	3	3
Transfer w/o Source or Destination	F	1	1
	Operation Allowed		
Command Chaining	Yes	Yes	Yes
List Data Chaining	Yes	Yes	No
Linked List Data Chaining	Yes	Yes	No
Assembly/Disassembly	No	Yes	Yes
Source Synchronization	Yes	Yes	Yes
Destination Synchronization	Yes	Yes	Yes
Free Running	No	Yes	Yes

* : The multiplexor channel can only run in the two cycle transfer mode.
** : Actual number of bus cycles may vary depending upon address boundary, hardware wait state number, pointer modification direction etc.
*** : Verify and Halt is executed properly if the subchannel is not byte/word multiplex.
F : Fatal error is generated.
N/A : Not Allowed

CHANNEL COMMANDS AND COMMAND BLOCKS

The 82258 controls the data transfer, with all its modifications, through the channel command blocks. These contain the channel command word and all the initial parameters for the data transfer execution. The channel start command from the CPU causes the 82258 to read the channel command block, with all its parameters from the memory and, to load them into the internal channel registers. The channel registers that are loaded via the command blocks are: CCR, SPR, DPR, BCR, TTPR, LPR/MTPR, MASKR and COMPR (see the register description for details on these registers). After examining the channel command for programming errors, the data block transfer is executed if no errors are detected. After the transfer termination, the reason for the termination is displayed in a word in the channel command block (channel status). Optionally, the last values of the source and the destination pointers and the byte count register may also be written out to the command block (constituting a status block if enabled). The CPU should not access the channel's control space while the channel is active (not stopped).

There are two basic types of channel commands:

Type 1 Channel Command - Data transfer Operation (Transfer Channel Command).

Type 2 Channel Command - Control Operation (Organizational Channel Command).

A complete channel program consists of at least one channel command block with a type one command and one type 2 command (stop).

Type 1 Channel Commands And Command Blocks

A command block always specifies a data transfer operation. The type 1 channel command defines the task to be performed by the channel (see the channel command register for details). Simple block transfer is specified by the short channel command block (Figure 16), which also allows data chaining. For more complex operations, the standard block is expanded by a command and a block extension, forming a long channel command block (Figure 16). The command block is always pointed at by the command pointer. Each channel has its own command pointer. Enabling of the status block (a bit in the channel command extension) extends the long channel command block by a status field of 12 byte length. This status field is loaded by the 82258 after the termination of the block transfer (Figure 16).

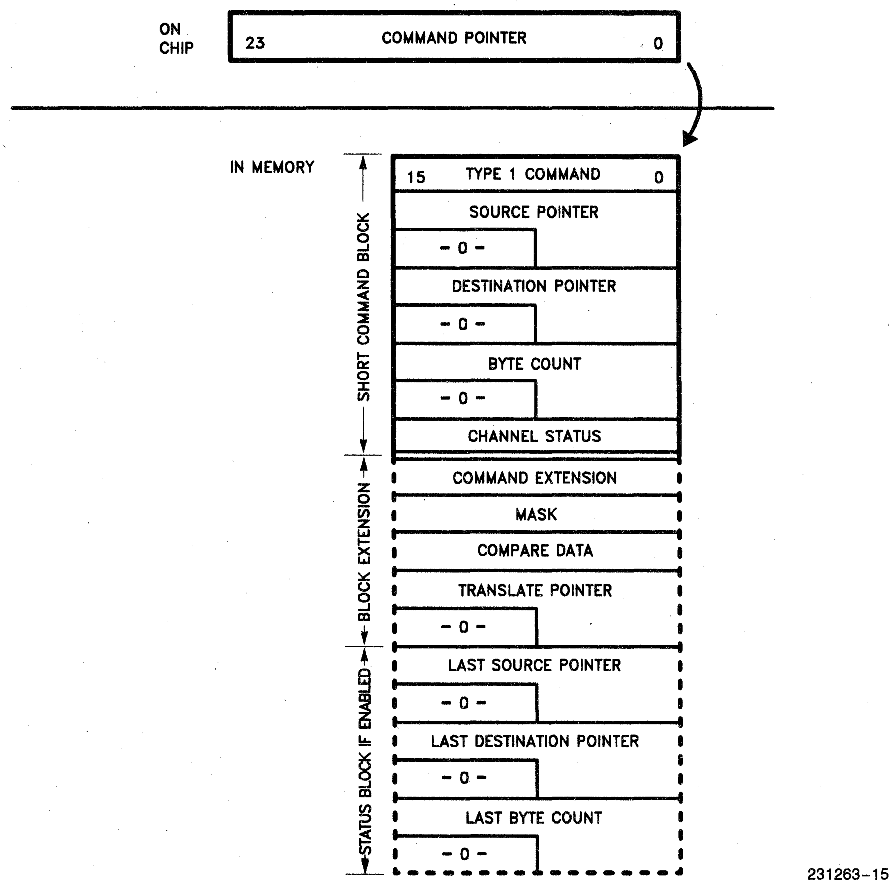


Figure 16. Type 1 Command Block

Type 2 Channel Commands and Command Blocks

The type 2 channel commands support the construction of channel programs by allowing operations such as auto-initialization, conditional chaining or program controlled interrupts. Figure 17 shows the structure of the type 2 channel command blocks.

The first word of the type 2 command block is the command and the second and the third may be an address.

Most of the type 2 commands can be executed conditionally; only exception being the unconditional stop which on the multiplexor channel functions as the Stop and Mask command. The 4 termination conditions are given in the CSR. If more than one condition is specified, the conditions are ORed. A special flag in the command word (I flag) allows to invert the channel status register bits before they are compared with the termination conditions. Table 6 gives the list of the different type 2 channel commands.

The type 2 commands can also activate a program controlled interrupt (INTOUT) and/or an $\overline{\text{EOD}}$ signal during the execution of a command (controlled by the ED and the IT flags). In the type 2 command the $\overline{\text{EOD}}$ is an asynchronous $\overline{\text{EOD}}$ (compared to the type 1 $\overline{\text{EOD}}$ which is synchronous to the last data transfer). If the ED or the IT flag is set, the signal generation is unconditional, independent of the condition code.

Table 6. Type 2 Channel Commands

Command
Relative Jump*
Absolute Jump*
Unconditional Stop
(Stop and Mask Subchannel for multiplexor channel)
Conditional Stop**

* : Both conditional or unconditional
** : The 82258 does not check if a selector channel only type 2 command is used on the multiplexor channel, but its execution will lead to erroneous channel processing.

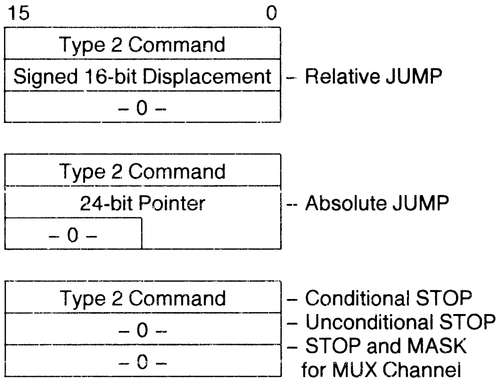


Figure 17. Type 2 Command Block

COMMAND AND DATA CHAINING

Command Chaining

The 82258 allows chaining of the command blocks in the memory, for any channel, for sequential execution. Figures 16 and 17 show channel command blocks and Figure 18 shows the examples of command chaining. The 82258 gets the address of the command block from its on-chip command pointer (initialized by the CPU) and starts executing. When it comes to the end of one command, it automatically starts to fetch and execute the next command block until a stop command is found. Conditional and unconditional STOP and JUMP commands allow complex sequences of DMAs to be performed.

Command chaining allows the 82258 to do CPU independent I/O processing, thus, saving valuable CPU time.

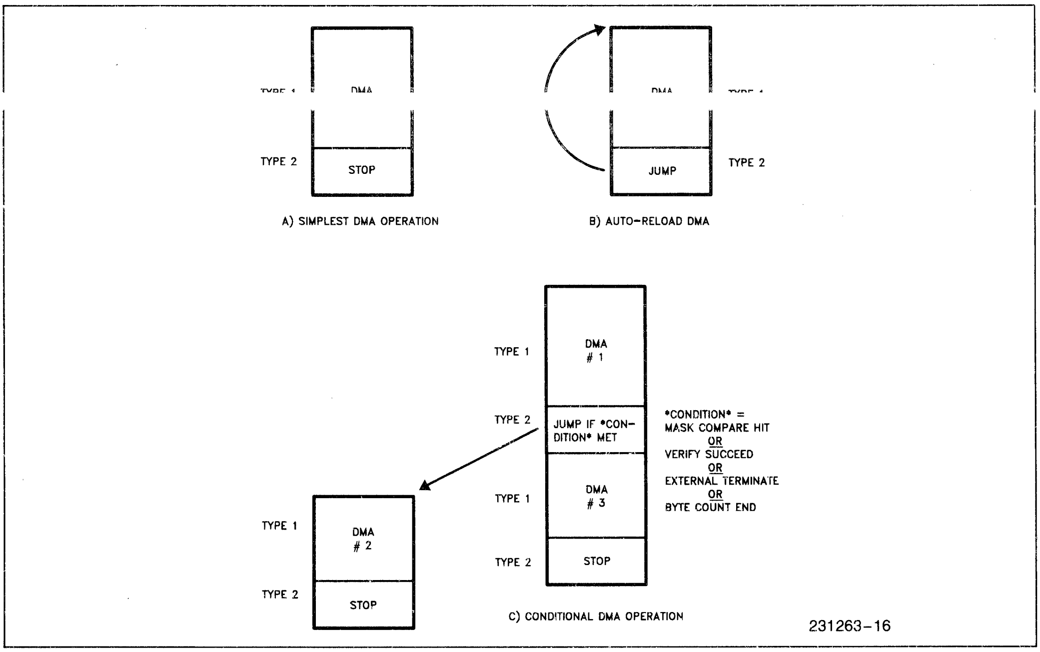


Figure 18. Command Chaining

Data Chaining

Data chaining allows gathering and scattering of data blocks. The 82258 permits automatic, dynamic linking of the data blocks scattered in the memory. Each data block in a chain can be up to 64K bytes. Two types of data chaining are allowed:

List Chaining: The chained data block descriptors are contiguous in a block which forms the data chain list (Figure 19). End of the chain is indicated by making the byte count field zero in the data chain list. List chaining is fast (1 microsecond between completion of one block transfer and going to the next element in the list, in an 8 MHz iAPX 286 system) but not very flexible.

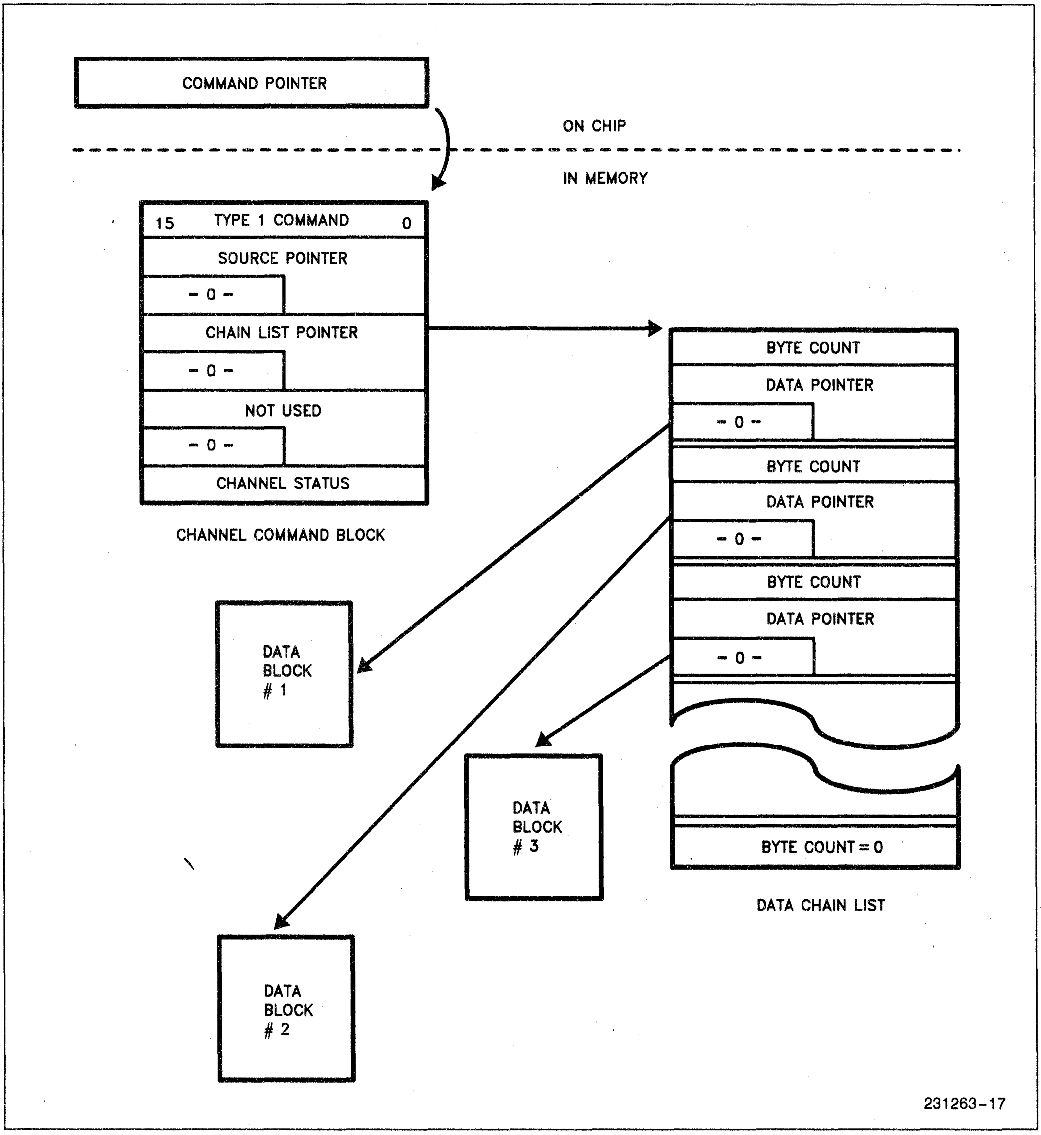


Figure 19. Destination List Chaining of Data

Linked List Chaining: Each list element which describes a particular data block (location and length) also holds a pointer to the next list element to be processed (Figure 20). End of the chain is indicated by making the byte count field zero in the linked list.

Linked list chaining is slower than the list chaining but the data blocks can be included, removed or, their sequence altered dynamically, through the link pointer manipulation by the CPU.

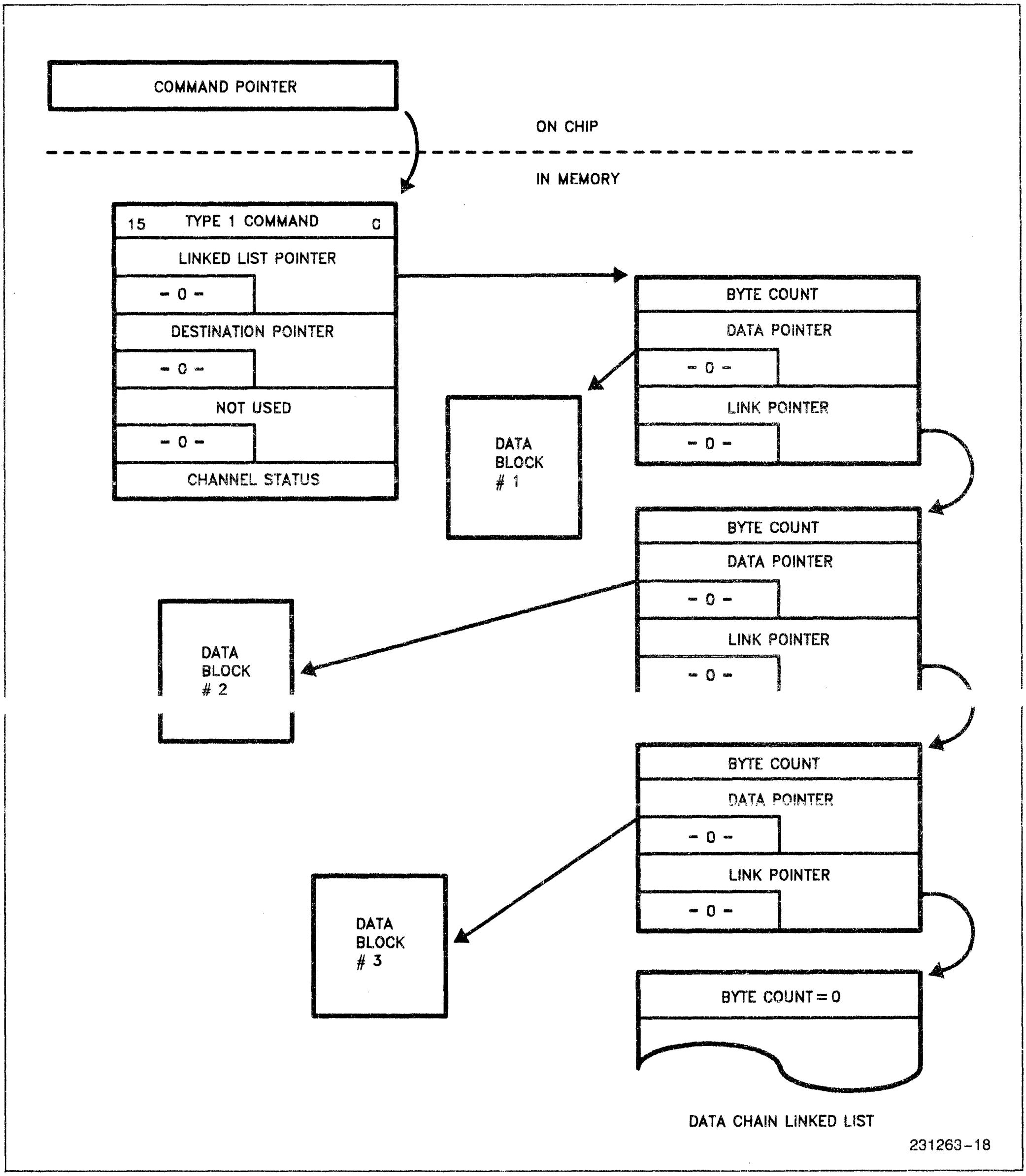


Figure 20. Source Linked List Chaining of Data

"ON THE FLY" OPERATIONS

The 82258 allows various data manipulation operations during the transfer:

Mask and Compare

Allows comparison of each byte, word or bit field (masking) in source data with some given pattern. Data transfer can be terminated on a match or a mismatch depending upon the program. This is possible both for the single and the two cycle transfer modes but, the transfer rate is halved in the single cycle mode.

Verify

Complete source data block is compared with a given data block. The data transfer can be terminated on mismatch (Verify and Halt). Supported only for the two cycle transfer mode.

Verify and Save

The data block is transferred from source to destination and in parallel compared with a given data block. The data transfer is not stopped on a mismatch. This operation is supported only for the single cycle transfer.

Translate

The source data (bytes) is translated with the aid of a translation table (Figure 21) before being sent to the destination. Translation is supported for the two cycle transfer mode only. If the destination is 16 bits, the two translated source bytes are assembled in the DAR before the destination cycle is executed.

Various 'on the fly' operations can be combined to allow the 82258 to perform versatile DMA operations.

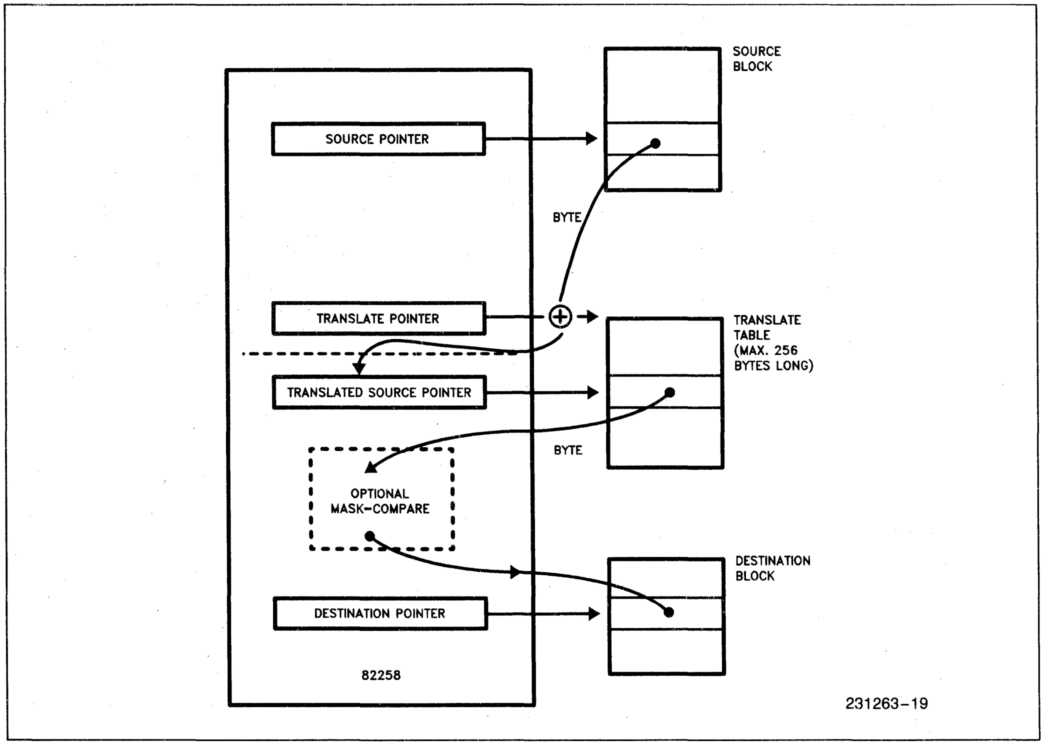


Figure 21. Translate Operation

PRIORITY CONTROL

The 82258 controls concurrent processing of its different channels (and subchannels) and, the internal and the external requests through a flexible priority scheme.

The PRI bits in the GMR are used to select the priority scheme which can be fixed or variable or a combination of the two (see the GMR description for the details). The unseparable bus cycles (e.g., 24 bit pointers) are not affected by the priority rotation. External 8259As determine the priorities for the multiplexed subchannels.

The processing of the internal or the external requests is controlled by a fully nested priority system including all four channels. Since more than one request can compete for the same channel, the requests are also prioritised in relation to their types as follows (in descending order of priority).

- Channel Stop (Command from the CPU out of the GCR)
- External asynchronous termination request (through $\overline{EOD}$)
- Internal continue request on previously interrupted sequence
- Start or stop subchannel or multiplexor channel
- Internal (without synchronization) or external (with synchronization) data service request or IO request for the multiplexor channel
- Channel wait (idle)

Data chaining and internal termination belong to the data service request processing, command chaining belongs to the termination processing.

Slave operations, where the 82258 is addressed by the CPU, have the highest priority of all the activities.

ADDRESSABILITY

The 82258 has two address spaces like the 80286, the 80186/188 and the 8086/88 processors:

- Memory space
- I/O space

Both the spaces are 16 MByte large for the 286/remote mode and 1 Mbyte for the 186/8086 mode. All types of transfers are possible:

- Memory/Memory
- I/O / I/O
- Memory/I/O
- I/O / Memory

Either of the memory or the peripheral can lie in either of the two spaces. Each space can be independently 8 bit or 16 bit wide. All possible Even-Odd boundary address combinations are supported for the data transfer from source (8 bit or 16 bit) to destination (8 bit or 16 bit) in the two cycle transfer mode. The source and the destination pointers can be incremented, decremented or not modified at all (INC/DEC bits of type 1 channel command in the CCR) after the corresponding data bus cycle. The 82258 does not indicate or check an 'address out of range' condition. Address overflow and underflow during a block transfer results in an address wrap around. Maximum length of the data block can be 16 MBytes in an iAPX 286 system. In the 186/86 mode the maximum byte count is (1M-1). This is not checked by the 82258.

SYNCHRONIZATION OF DATA TRANSFER

The 82258 allows both the external synchronization of a DMA transfer (from a source or a destination device) or a free running DMA (internally synchronized).

The external synchronization allows control of input/output operations in the cycle of the peripheral device, hence occupying the bus only when the peripheral is able to receive or transmit data.

Free running DMA (no external synchronization) is used for the memory to memory transfers, during a continuous DMA request or, in the block multiplex subchannel after the channel start. It is not supported for the single cycle transfer mode.

286 PROTECTION

The 82258 needs special consideration to operate in an iAPX 286 system in the protected mode. The 82258 works only with the real addresses but it supports a protected mode iAPX 286 system if the following conditions are fulfilled:

- The 286 kernel software must check all the protection rules during the set up routine for the 82258 and perform the limit checks for the block transfers. This is supported by the 80286 instructions e.g. VERR (verify Read Access), VERW (verify Write Access), LSL (load Segment Limit).
- The 286 kernel has to translate the logical addresses into the physical addresses.
- All the 82258 registers should be memory mapped and access to them should be allowed only for a 286 kernel routine (task isolation).

Normally an I/O utility routine is provided by the operating system to service the 82258. No direct user access should be allowed to the 82258 from the lower privilege levels. The real addresses can be generated only by using the 286 protection mechanism and are so checked against any protection violation.

82258 REGISTER MODEL

The 82258 has three sets of registers (Figure 22):

General Registers
Channel Registers
Multiplexor Channel Registers

All registers can be read or written into by the CPU but, most are accessed only for the test purposes. The CPU loads some registers (e.g. General Mode Register) during the initialization after the reset, and others during the invocation of a channel (General Command Register). Some of the channel registers are programmed or read by the CPU but most of them are loaded by the 82258 itself during the setup routine after a channel start. All accessible registers can be accessed bytewise or wordwise by the CPU.

Figure 23 gives a layout of the registers. Note that all registers lie on even addresses.

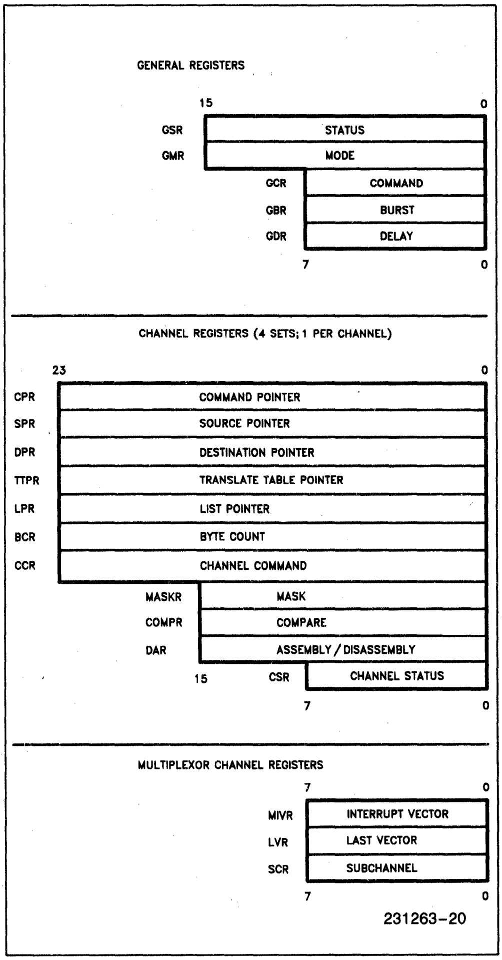


Figure 22. 82258 Register Set

Address Bits 5-0 (hexadecimal)	Address Bits 7,6				Address Bits 5-0 (binary)
	00	01	10	11	
0	GCR				000000
2	SCR				000010
4	GSR				000100
6	RESERVED	RESERVED	RESERVED	RESERVED	000110
8	GMR				001000
A	GBR				001010
C	GDR				001100
E	RESERVED				001110
10	CSR0	CSR1	CSR2	CSR3	010000
12	DAR0	DAR1	DAR2	DAR3	010010
14	MASKR0	MASKR1	MASKR2	MASKR3	010100
16	COMPR0	COMPR1	COMPR2	COMPR3	010110
18				MIVR	011000
1A	RESERVED	RESERVED	RESERVED	LVR	011010
1C				RESERVED	011100
1E					011110
20	CPRL0	CPRL1	CPRL2	CPRL3	100000
22	CPRH0	CPRH1	CPRH2	CPRH3	100010
24	SPRL0	SPRL1	SPRL2	SPRL3	100100
26	SPRH0	SPRH1	SPRH2	SPRH3	100110
28	DPRL0	DPRL1	DPRL2	DPRL3	101000
2A	DPRH0	DPRH1	DPRH2	DPRH3	101010
2C	TTPRL0	TTPRL1	TTPRL2	TTPRL3	101100
2E	TTPRH0	TTPRH1	TTPRH2	TTPRH3	101110
30	LPRL0	LPRL1	LPRL2	LPRL3/MTPRL	110000
32	LPRH0	LPRH1	LPRH2	LPRH3/MTPRH	110010
34	RESERVED	RESERVED	RESERVED	RESERVED	110100
36	RESERVED	RESERVED	RESERVED	RESERVED	110110
38	BCRL0	BCRL1	BCRL2	BCRL3	111000
3A	BCRH0	BCRH1	BCRH2	BCRH3	111010
3C	CCRL0	CCRL1	CCRL2	CCRL3	111100
3E	CCRH0	CCRH1	CCRH2	CCRH3	111110

GCR = General Command Register
SCR = Subchannel Register
GSR = General Status Register
GMR = General Mode Register
GBR = General Burst Register
GDR = General Delay Register
CSR = Channel Status Register
DAR = Data Assembly Register
MASKR = Mask Register
COMPR = Compare Register
L = Low Word
H = High Byte

MIVR = Multiplexor Interrupt Vector Register
LVR = Last Vector Register
CPR = Command Pointer Register
SPR = Source Pointer Register
DPR = Destination Pointer Register
TTPR = Translate Table Pointer Register
LPR = List Pointer Register
MTPR = Multiplexor Table Pointer Register
BCR = Byte Count Register
CCR = Channel Command Register
0, 1, 2, 3 = Channel Number

Figure 23. Layout of Register Addresses

GENERAL REGISTERS

These registers are common to all the channels.

General Mode Register (GMR)

This is the first register to be programmed after the reset since it describes the 82258 environment. Here the system wide parameters are specified. The 16 bit register is loaded bytewise with the low byte being programmed first.

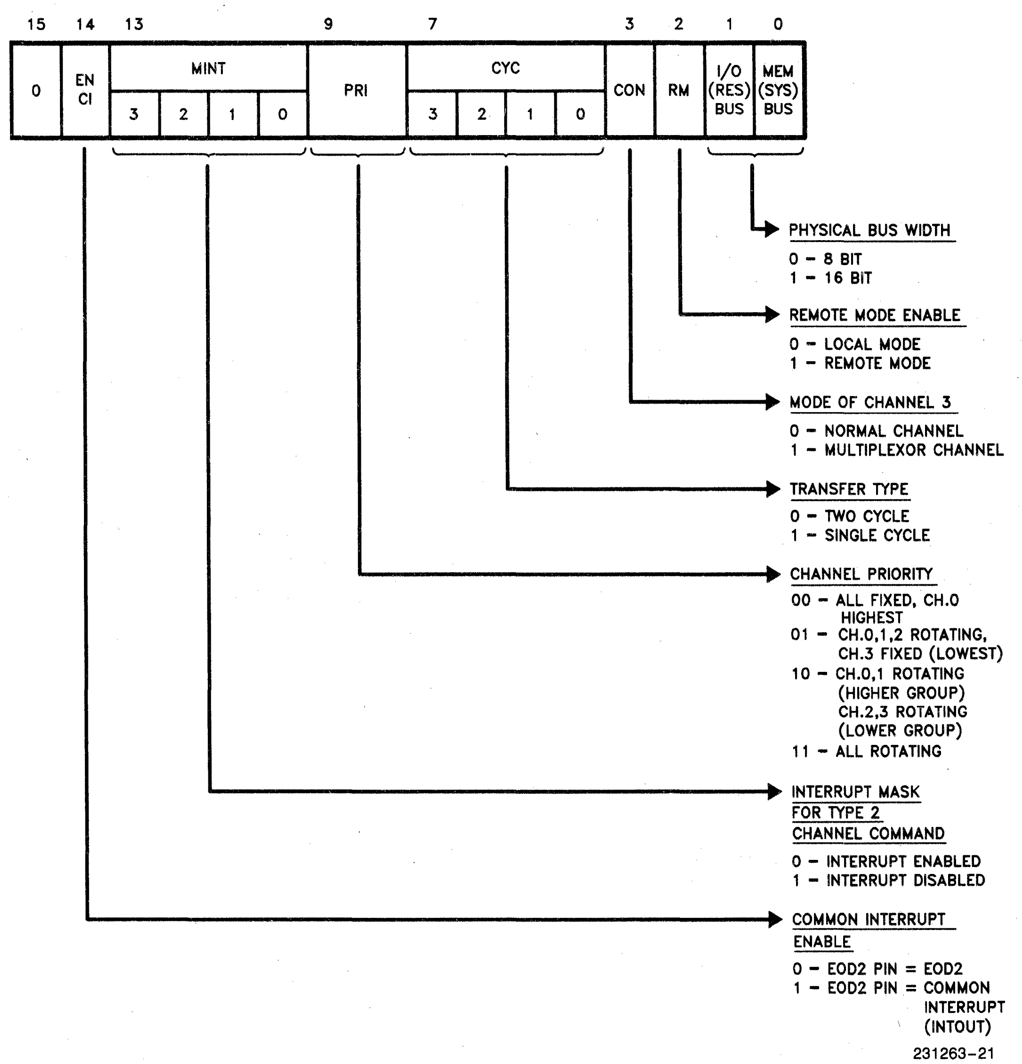


Figure 24. General Mode Register

General Status Register (GSR)

This register provides the status information for all the channels. It also shows which channels have interrupts pending and, where the channel control space lies. It is a 16 bit register.

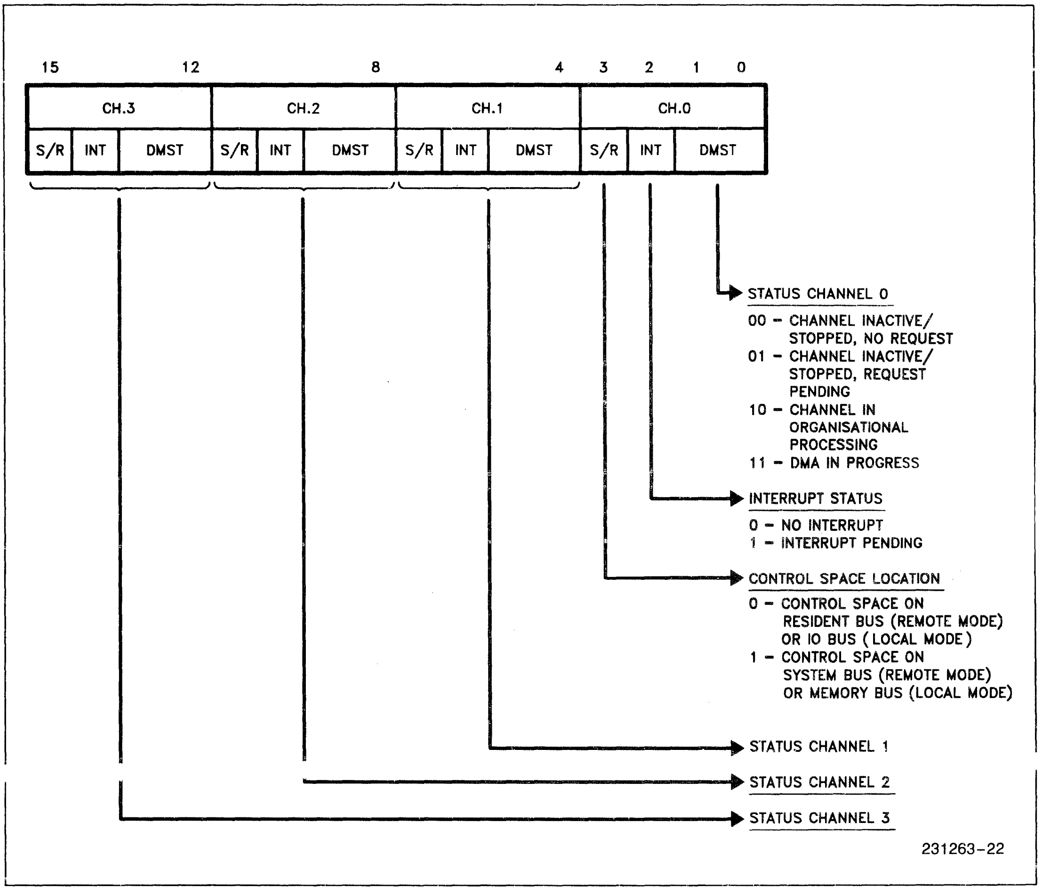


Figure 25. General Status Register

General Command Register (GCR)

GCR is an 8 bit register directly loaded by the CPU to start or stop a channel. The START command also defines the control space assignment. The pending interrupt from any channel is also cleared through the GCR. Any combination of channels can be addressed simultaneously. To start/stop a multiplexor subchannel, the subchannel number must be first loaded in the Subchannel Register (SCR). The Halt/single step command is useful for the system debugging.

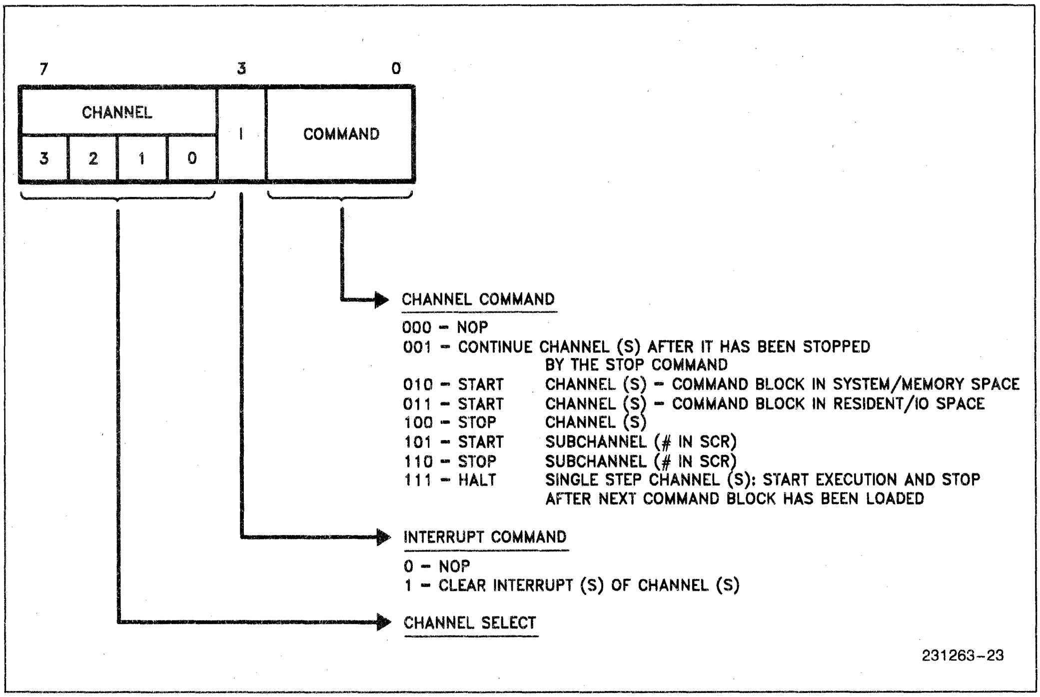


Figure 26. General Command Register

General Burst Register (GBR)

This 8 bit register determines the maximum number of contiguous bus cycles that can be requested by the 82258. GBR = 0 means unlimited contiguous bus cycles for the 82258. The GBR must be directly loaded by the CPU.

General Delay Register (GDR)

GDR is an 8 bit register which determines the minimum number of clocks between the 82258 burst accesses. GDR = 0 means no minimum delay between the HOLD request.

Burst/Delay Algorithm

Both the GBR and the GDR do their actual counting through their respective counters the GBC and the GDC. For the burst and delay counters, the following rules apply:

- Whenever the 82258 controls a bus cycle the burst counter is decremented by one but not beyond zero.
- Whenever the 82258, in the local mode, does not have the bus, the delay counter is decremented by one: every second T-state in the 286 mode or, every fourth T-state in the 186 mode.
- Whenever the delay counter is zero, the burst and the delay counters are loaded from the burst and the delay registers.
- If the burst counter is zero (and no exception occurs), the 82258 releases the bus and the delay counter counts until it is zero. Then both counters are loaded from their corresponding registers and the 82258 can again request the bus by activating HOLD signal. Unseparable bus cycles are the exception to this rule. Counting of the burst is not prevented but surrendering of the bus is.
- In the remote mode the burst and the delay are relevant only for the system bus cycles. The GBC is only decremented while the 82258 performs the system bus cycles and the GDC decrements when the 82258 does not control the system bus (idling or the resident bus cycles).

CHANNEL REGISTERS

Each of the four 82258 channels has these registers. All the channel registers are loaded by the 82258 from the memory except the Command Pointer (CPR) [Multiplexor Table Pointer (MTPR) & Subchannel Register (SCR) for the channel 3 in the multiplexor mode]. The initial contents of the registers are specified, by the CPU in the command blocks in the memory.

Command Pointer Register (CPR)

This 24 bit register contains the physical address of the command block. It must be loaded by the CPU before starting the channel. For the channel 3 in the multiplexor mode, the CPR is loaded by the 82258 from the multiplexor table (MT) in the memory.

Source Pointer Register (SPR)

SPR is 24 bits and contains the physical address of the source (memory or I/O, system or resident space) in a DMA transfer. In the single cycle transfer mode, it contains the only address pointer (source or destination).

Destination Pointer Register (DPR)

DPR contains the physical address of the destination (memory or I/O, system or resident space) in a DMA transfer. During Verify operations it contains the verify pointer (pointer to compare the data block). For the single cycle transfer mode it is only used for the verify and save operation. It is 24 bit register.

Translate Table Pointer Register (TTP

This 24 bit register is used to reference translate table in the memory when the translation is enabled in the channel command register extension (CCRX).

List Pointer Register (LPR)

LPR is used for data chaining (list ed list) operation. It is a 24 bit register and p the list element. In the multiplexor mode for t el 3, it is used as the Multiplexor Table Register (MTPR). (Multiplexor mode does n t data chaining).

Byte Count Register (BCR)

BCR is a 24 bit register and conta count for the DMA transfer.

Channel Command Register (CCR)

CCR specifies the type of DMA transfer or the type of internal operation. The channel commands are contained in a channel command block. The 82258 has two types of channel commands:

— Type 1 for data movement

— Type 2 for command chaining control

The channel command register has three configurations:

— Short Type 1 command: SYN field NE. 00 and ECX = 0. Upper 8 bits, i.e., Channel Command Register Extension (CCRX field), are not valid.

— Long Type 1 command: SYN field. NE. 00 and ECX = 1. All 24 bits are valid.

— Type 2 command: SYN field = 00, Upper 8 bits (CCRX field) are not valid.

Figure 27 shows CCR for Type 1 command and Figure 28 has the CCRX (Channel Command Register Extension). Figure 29 shows CCR for type 2 command.

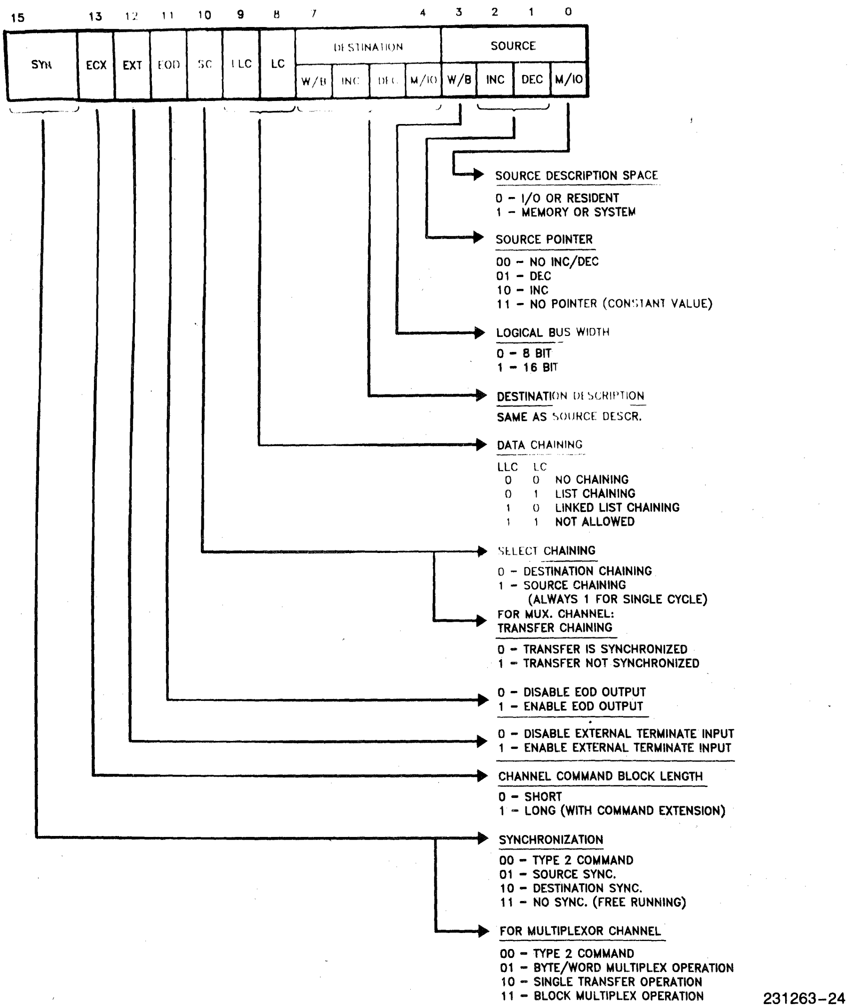


Figure 27. Type 1 Channel Command CCR

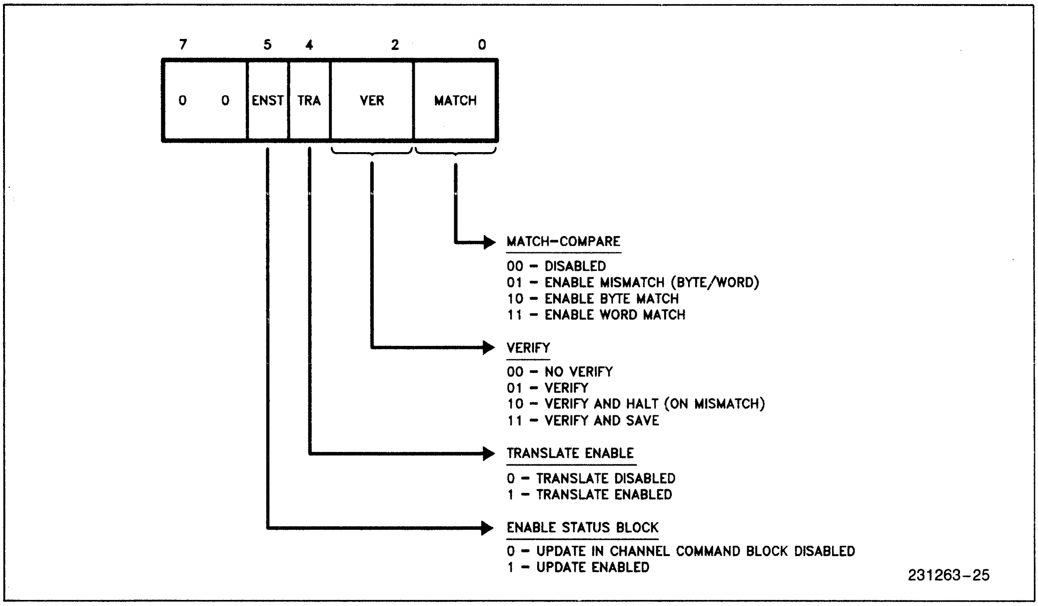
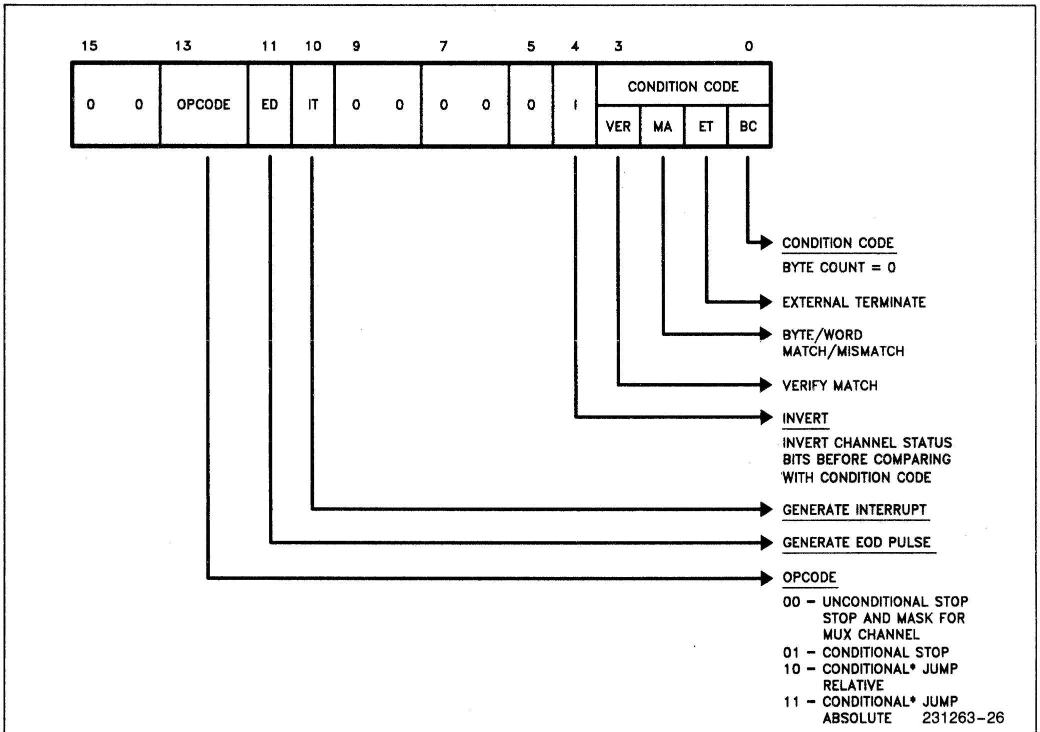


Figure 28. Channel Command Register Extension CCRX



* If all condition code bits are 1, Jump becomes unconditional.

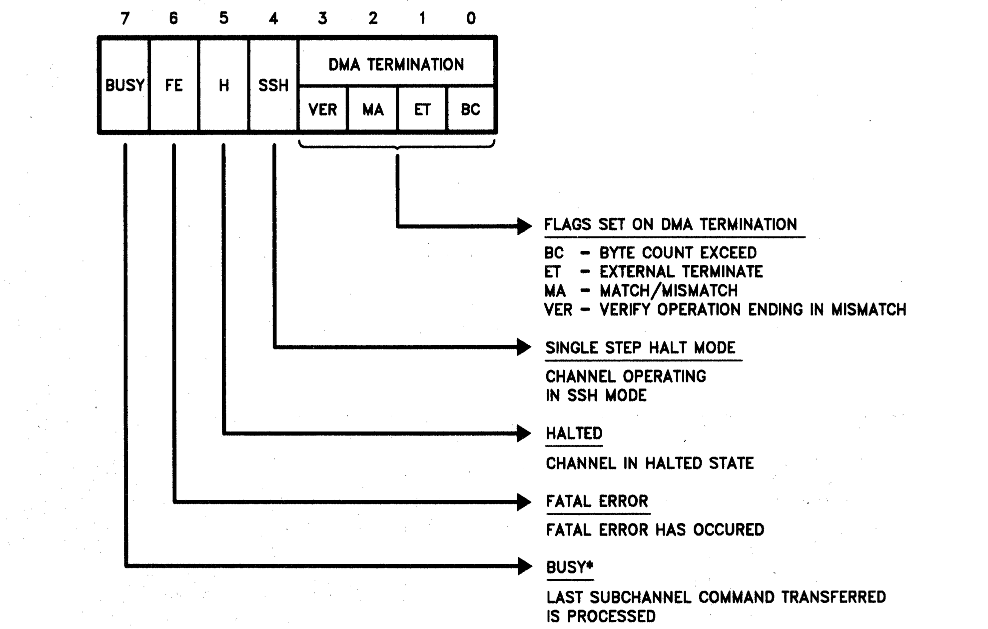
Figure 29. Type 2 Channel Command CCR

Mask Register (MASKR) and Compare Register (COMPR)

Both of these registers are 16 bit and are used during the match/mismatch operation. For comparison with the transferred data, only those bit positions in the Compare Register which are not masked with 1's in the Mask register are considered. These two registers together allow byte, word or bit level comparisons. MASKR is also used during the verify operations. MASKR and COMPR each should contain two identical bytes for Byte Match/Mismatch operations.

Channel Status Register (CSR)

CSR, an 8 bit register, reflects the status of the channel. The least significant half byte is the termination condition and the most significant half byte indicates fatal error, busy state and halted state.



* Valid only for channel 3 in multiplexor mode; zero otherwise.

Figure 30. Channel Status Register

Data Assembly Register (DAR)

This 16 bit register is used for automatic assembly/disassembly of data.

Mutiplexor Channel Registers

These registers are valid only for channel 3, when used as a multiplexor channel.

Multiplexor Table Pointer (MTPR)

This register is used to reference the multiplexor table in the memory when channel 3 is programmed as a multiplexor channel. Since data chaining is not allowed for the multiplexor channel, the List Pointer Register (LPR) is used as the MTPR. MTPR is 24 bit and must be loaded by the CPU.

Multiplexor Interrupt Vector Register (MIVR)

This 8 bit register is used by the CPU to determine which channels are stopped. The vectors of the stopped subchannels are output in the priority order (0 has the highest priority) upon each reference of this register, until the NV bit is set. A maximum of 32 vectors can be distinguished.

Last Vector Register (LVR)

LVR gives the last vector read by the 82258 (from the 8259A). In case of a fatal error stop of channel 3, LVR determines the guilty subchannel. LVR is an 8 bit register.

Subchannel Register (SCR)

This register gives the 8 bit subchannel number for the general commands START/STOP Subchannel. It must be loaded by the CPU before a subchannel command is written into the GCR. MIVR limits the number of subchannels supported to 32 (5 bits).

82258 OPERATION AND PROGRAMMING OVERVIEW

INITIAL STATE

Upon activation of the RESET signal:

- all channels are disabled (by clearing the DMA status bits in the General Status Register)
- all bus activities are stopped
- all tristate signals are tristated and the others enter the inactive state

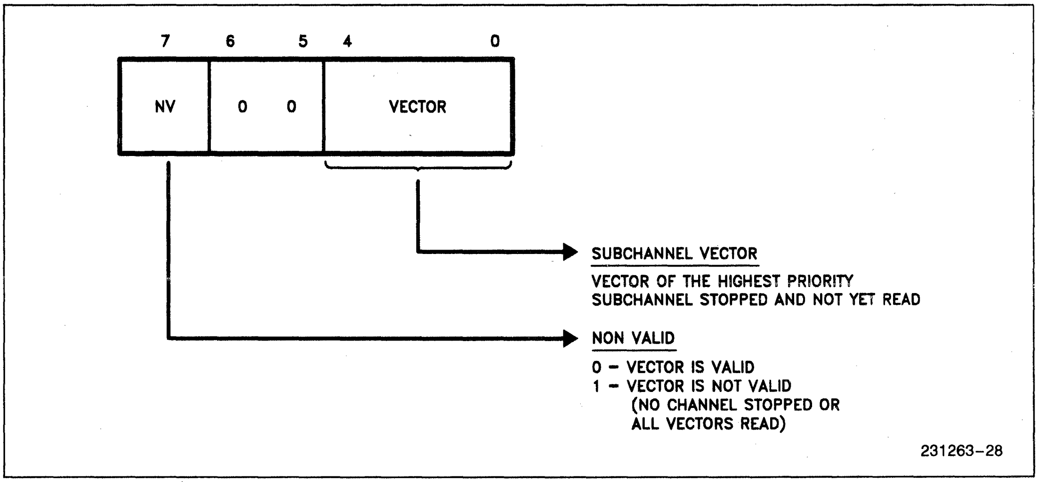


Figure 31. Multiplexor Interrupt Vector Register

After the RESET signal becomes inactive, the 82258 state gets defined:

— it is in the 186 mode if A23 pin was low at the falling edge of RESET; otherwise it is in the 286 mode

— it is in the 8086 max (Request/Grant) mode if the 186 mode is detected and HLDA pin was high at the falling edge of RESET; otherwise it is in the 186/8086 Min. (HOLD/HLDA) mode.

— The contents of the 82258 registers are as follows:

- GMR: All bits are zero
- GBR: Zero value
- GDR: Zero value
- GSR: — DMST bits for channels: 0X (Stopped)
  - — INT for all channels: 0 (no interrupt pending)
  - — S/R = 0 (I/O or resident space)
- All Channel Status Registers (CSR): Zero Values
- MIVR: NV = 1 (Vector not valid)
  - — Vector is all 1, rest zero
  - — All stop bits in matrix are reset
- All other registers (GCR, LVR, SCR, CPRn, SPRn, DPRn, TTPRn, LPRn, BCRn, CCRn, COMPRn, MASKRn, MTPR) are undefined

INITIALIZATION AND CHANNEL INVOCATION

After RESET, the 82258 has to be initialized by the CPU. The General Mode Register (GMR) should be loaded first in the 16 bit systems; the lower byte of the GMR (which gives main configuration information) in the 8 bit systems.

SYSBUS (MEMBUS) bit of the GMR determines the physical bus width of the CPU-82258 communication. All register write and read operations are executed:

— Bytewise on the lower half of the data bus (D7–D0), if SYSBUS (MEMBUS) = 0

— wordwise on D15–D0 if SYSBUS (MEMBUS) = 1. Byte transfers are also possible here with the bytes being transferred on that half of the data bus which is addressed by the least significant bit of the register address.

Internally the 82258 uses $\overline{\text{BHE}}$ and A0 to detect the effective transfer width of the 82258—CPU communications. After the GMR, the General Burst Register (GBR) and the General Delay Register (GDR) should be programmed, if needed (Initial state = 0 for both), by the CPU.

Before a channel is invoked, the control space in the memory and the channel registers in the 82258 have to be initialized:

Selector Channel Start

Following conditions should be met:

— channel program in the control space

— if data chaining enabled, the chaining list or the linked lists in the control space

— if translate enabled, the translate table in the control space

— load the CPR with the start address of the channel program

Multiplexor Channel Start

For the multiplexor channel operation, the following is essential:

— the multiplexor table MT in the control space with the subchannel command pointer and the mask register pointer of the associated 8259A for each subchannel

— initialization of the 8259A's mask registers by masking off all the request inputs. In the remote mode, this can also be done by the data transfer operation on the selector channel (or by stop subchannel commands)

— load MTPR with the base address of the multiplexor table (MT)

For the subchannel start

— the subchannel program should be in the control space

— if translate enabled, the translate table should be in the control space

— the subchannel command pointer should be in the multiplexor table

— read the multiplexor channel status register CSR3. Write a new subchannel number into the SCR only if BUSY bit = 0.

In case of a normal channel start, the last CPU operation is to write the general command into the GCR. Then the start will be processed by the 82258 according to the requested channel's priority, with the highest priority being processed first. If the addressed channel is already active, the start command is ignored. If I = 1 in GCR, the INT bit(s) of the indicated channel(s) will be erased in the GSR.

COMMAND EXECUTION

Selector Channel: The command bits in the GCR give the commands available to a selector channel. Execution of the continue and the start commands is prioritized; the stop commands are executed immediately. The stop command forces the DMA status bit (DMST) in the GSR to channel inactive (stopped) without any additional routine. The continue command works directly with internal stored register parameters and continues a previously stopped channel operation. The start commands define the location of the control space and initiate the set up routine. The halt command has multiple functions:

— It forces the channel into the single step and halt mode, indicated by the SSH bit in the CSR

— If the channel is running, it will be halted after the completion of the current command block execution; the halted data is shown by the H bit of the CSR; the DMST bits of the GSR are not changed

— If the channel is halted (or stopped) the halt/single step command starts the channel, and the channel will again be halted after the completion of the next command block execution (type 1 or 2)

The single step and halt mode is finished by a start or a continue command. After a channel start, first the general status reflected in the GSR is changed into 'DMA in organizational processing'. GSR also indicates the location of the control space (S/R bit). After the prioritization of the start command, the channel's set up routine is executed.

After the set up routine execution, all the transfer parameters are accessible in the 82258 internal registers. The SYN bits in the CCR decide:

— if the channel activity is continued by an immediate start of the data transfer (i.e., free running mode or an internal data transfer service request)

— or the channel is waiting for a DMA request i.e., external synchronization mode.

Multiplexor Channel: On the multiplexor channel, there are two cases:

a. The whole channel has to be treated by a general command

b. Only the addressed subchannel has to be treated by a general command

a. In case of the whole channel, the commands are the same as the selector channel commands. Execution of the continue and the stop (stops whole channel) is the same. The channel 3 start command has only two functions:

— specify whether the system/memory or the resident/IO control space has to be used on the multiplexor channel (S/R bit in GSR)

— change of the general status of the channel 3 (DMST bits in GSR) into "Channel started but idling" thus, enabling the IOREQs and the Subchannel commands.

The general channel command "Halt/Single Step" has a slightly different interpretation for the multiplexor channel. While the selector channel can only be halted during the chaining of the command blocks, the multiplexor channel in the single step/halt mode will also be halted when it takes the idle state. In that case, a new halt/single step command will only be executed if an IOREQ or a subchannel start/stop command is pending.

b. With the start subchannel command, the 82258 unmasks the corresponding bit in the 8259A mask register for the addressed subchannel, thus enabling the subchannel. The BUSY bit in the CSR is set indicating the state: "subchannel command pending". After prioritization, the subchannel routine is executed. When an I/O request is received on the subchannel, the command pointer is fetched from the MT and the channel's set up routine is executed. After the reset of the BUSY bit, a new start/stop subchannel command can be accepted by the multiplexor channel.

Only distinction between the stop subchannel command and the start subchannel command is the handling of the mask bit in the 8259A. For the STOP command, the vector specific mask bit is set by the 82258. As the start command, the stop command has also to be prioritized before execution.

For the multiplexor channel the following rules are observed:

— Before any IOREQ can be processed, the whole channel 3 has to be started and the channel 3 must be in the idle state

— In any state a subchannel command can be accepted and transferred into the state "subchannel command pending"

— A pending subchannel command can be processed only in the idle state

— In the idle state, a subchannel command has a higher priority than an IOREQ

— In case of a fatal error stop of a subchannel, the whole channel 3 is stopped. LVR identifies the guilty subchannel. To stop (mask) this subchannel, the CPU at first has to issue a START CH3 command and then stop the affected subchannel.

TERMINATION CONDITIONS

The 82258 distinguishes the following conditions for termination of a block transfer:

- byte count is zero and the data chaining not enabled; a standard termination condition
- data chaining enabled and the new fetched byte count is zero
- external termination via the channel's $\overline{EOD}$ line if enabled by the EXT bit in the CCR
- match/mismatch during the masked byte or word compare, as specified and enabled in the command extension CCRX
- mismatch during a verify & halt operation, as specified and enabled in the command extension CCRX
- The CPU loading the GCR with a stop command, though the channel is not really terminated.

INTERRUPT CONTROL

The 82258 has four programmable $\overline{EOD}$ pins (one for each channel) for the CPU interruption and for communication with the system environment. As inputs, the $\overline{EOD}$ pins are used for external termination, enabled by the EXT bit of the type 1 channel command in the CCR. When used as output, the $\overline{EOD}$ pins provide two basic functions:

$\overline{EOD}$ (end of DMA), a channel specific active LOW pulse signal of 2 T-states length, always enabled by the software. With a type 1 channel command, $\overline{EOD}$s, if enabled, are synchronous and always controlled by the byte count. If data chaining is enabled, type 1 $\overline{EOD}$s should not be used for interrupts since multiple $\overline{EOD}$s (with every exceeding byte count) are issued. With a type 2 command, the $\overline{EOD}$, if enabled (ED = 1 in the CCR), is an asynchronous signal generated after a command execution.

INTOUT (interrupt output) is a hardware generated (error detection) or a software enabled static active HIGH signal on the $\overline{EOD2}$ pin, if programmed (ENCI = 1 in the GMR). The channel generating the INTOUT is indicated by the INT bit in the GSR. Hardware generated interrupt occurs in case of a fatal error (INTOUT issued if not masked by the MINT bit in the GMR). Type 2 channel command allows software generated INTOUT if programmed (IT = 1 in the CCR and not masked by the MINT bit in the GMR). A channel's INT bit in the GSR is activated independent of the MINT (in GMR). INTOUT remains active until all INT bits in the CSR are reset by the CPU with the general command CLEAR INTERRUPT.

Multiplexor Channel Interrupts

Interrupts from the multiplexor channel belong to a certain subchannel. For program controlled interrupts, the status and the context information cannot be fetched from the internal 82258 registers (since the multiplexor channel is not stopped). Hence, the CPU can only investigate the interrupt via the MIVR register. After the MIVR read from the CPU, the valid bit and matrix stop bit (the vector of which was indicated in the MIVR) are erased. For multiple stop conditions in the stop matrix, the stopped subchannels get their vectors in the MIVR in the priority order (highest for vector zero). The MIVR is activated independent of the programming of $\overline{EOD}$ or INTOUT. Therefore, the CPU can sample the MIVR in a polling mode when neither $\overline{EOD}$ nor INTOUT is used. With the interrupt vector out of the MIVR, the CPU finds the related command pointer (in MT) which points to the last executed channel command (stop and mask). For status information of last block transfer, the CPU has to find the last type 1 command block in the channel program. Programmable intermediate interrupt messages should not be used on the multiplexor subchannels (MIVR is activated only for the stopped subchannel).

For hardware generated INTOUT the whole channel 3 is stopped with the LVR indicating the last (guilty) vector. After the error investigation the CPU should start the channel 3 and then stop the affected subchannel.

FAULT DETECTION

On detecting a fatal error, the 82258 does the following:

immediately stops the affected channel
sets error bit in the channel's status register
sets channel specific INT bit in the GSR
sends interrupt if not masked (in GMR)

For error investigation, the CPU should:

read GSR (what channel?, channel stopped?)
read CSR (error?)
read CPR and investigate the channel command (type 1 command)
read LVR for multiplexor channel, if affected (what subchannel?)

The 82258 recognizes only type 1 command errors. Other error types are defaulted into non-fatal errors and not identified. The FE bit in the CSR indicates the fatal errors.

Fatal Errors: Fatal errors are detected during the decoding of a type 1 channel command with the GMR. Six conditions are used for detection and the allowed six combinations of them lead to six different transfer executions (Table 7). All other combinations of the six conditions generate a fatal error.

Table 7 Fatal Error Detection

Valid Combination	Conditions Decoded						Operation Performed
	Single Cycle	No Dst. Ptr.	No Src. Ptr.	Verify & Save	Trans-late	Sync. Error	
1	False	False	False	False	False	—	Two Cycle DMA
2	False	False	False	False	True	—	Translate
3	False	False	True	False	False	False	No Source Ptr. DMA
4	False	True	False	False	False	False	No Dest. Ptr. DMA
5	True	False	False	False	False	False	Single Cyc. DMA
6	True	False	False	True	False	False	Verify & Save

The synchronization error is predecoded and activated in the following cases:

Single cycle combined with free running

No source pointer mode combined with the source synchronization on a selector channel

No destination pointer combined with the destination synchronization on a selector channel

Non Fatal Errors and Undetected Fatal Errors

A non fatal error is not indicated in the channel status register. It is only defaulted. Channel processing is not interrupted. Following are some examples of non fatal errors and the undetected fatal errors:

Fault	Action
Remote mode + 186 mode	RM not inhibited but read/write pins are also used as outputs
Both list chaining and linked list chaining enabled	Linked list data chaining executed
Start/Stop subchannel and BUSY active	New command overwrites old command (Fatal Error)
Data chaining enabled on the multiplexor channel	MTPR is overwritten with the list pointer (Fatal Error)

TRANSFER RATES

Selector Channel

Table 8 illustrates the different transfer rates (in MBytes/sec) for the 286 mode of operation. These transfer rates are not affected by switching channels and are halved for both 186 and 86 modes of operation.

Table 8. Cummulative Selector Channel Transfer Rates (8 MHz 286 System)

Transfer	Single Cycle	Two Cycle
Word → Word	8	4
Word → Byte	not possible	2.66
Byte → Word	not possible	2.66
Byte → Byte	4	2
Byte → Byte w/ Translate	not possible	800 KBytes

Multiplexor Channel

The transfer rates on the multiplexor channel are different from the selector channel and depend on the mode of operation and the size of the command block.

Table 9. Cummulative Multiplexor Channel Transfer Rates

Mode	Command Block	Word Transfers	Byte Transfers
Byte/Word Multiplex	short	275 KBytes/sec	138 KBytes/sec
	long	240 KBytes/sec	120 KBytes/sec
Block Multiplex	short	4 MBytes/sec	2 MBytes/sec
	long	4 MBytes/sec	2 MBytes/sec

Data Chaining

The transfer rate for data chaining depends on the block length of each chained data block, the number of blocks in the chain and also the type of chaining that is being done. See the section on data chaining latencies.

LATENCIES

The latency calculations do not take into account set up, hold and output delay times which are specified in the A.C. Characteristics section. These should be added to get the final latency figures. All timings are in units of T-states (125 ns in an 8 MHz system). If bus cycles are involved then the following abbreviations are used:

T = time for one bus transfer

W = wait time during bus cycles for a slow device

In case of various influences affecting the timing, the most typical case is mentioned in the table and explained in notes.

Assumptions:

1. The channel for which latencies are calculated currently has the highest priority and will not be blocked by other still higher priority requests.
2. In remote mode delays due to CPU accesses to the 82258 are not taken into account for latencies.
3. All control space accesses are on a 16 bit bus and command blocks and data chain lists are addressed on even boundaries.
4. Organizational and other unsynchronized transfers (e.g. prefetch) have been completed before the processing of DREQ starts.

DMA Request Processing:

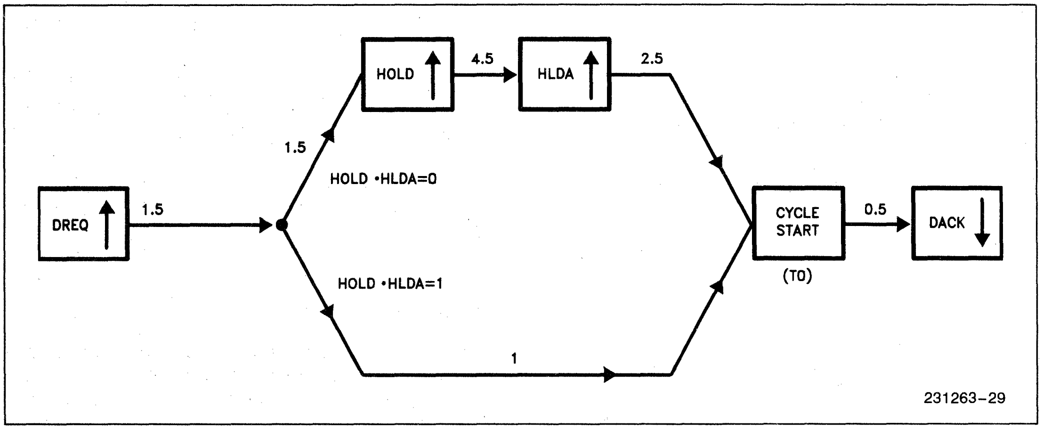


Figure 32. DREQ to DACK Latency in Local Mode*

Table 10. DREQ to DACK in Local Mode*

	Minimum	Typical	Maximum
DREQ to HOLD	2.5	3	3 + W (1) (2)
HOLD to HLDA	1	4.5	(3)
HLDA to CYCLE START	1.5	2.5	2.5
DREQ to CYCLE START (without bus arbitration)	2	2.5	4 + W (1)
CYCLE START to DACK	0.5	0.5	0.5

Notes are indicated in parenthesis

*All timings are in units of T-states (125 ns in an 8 MHz system). If bus cycles are involved then the following abbreviations are used:
T = Time for one bus transfer
W = Wait time during bus cycles for a slow device

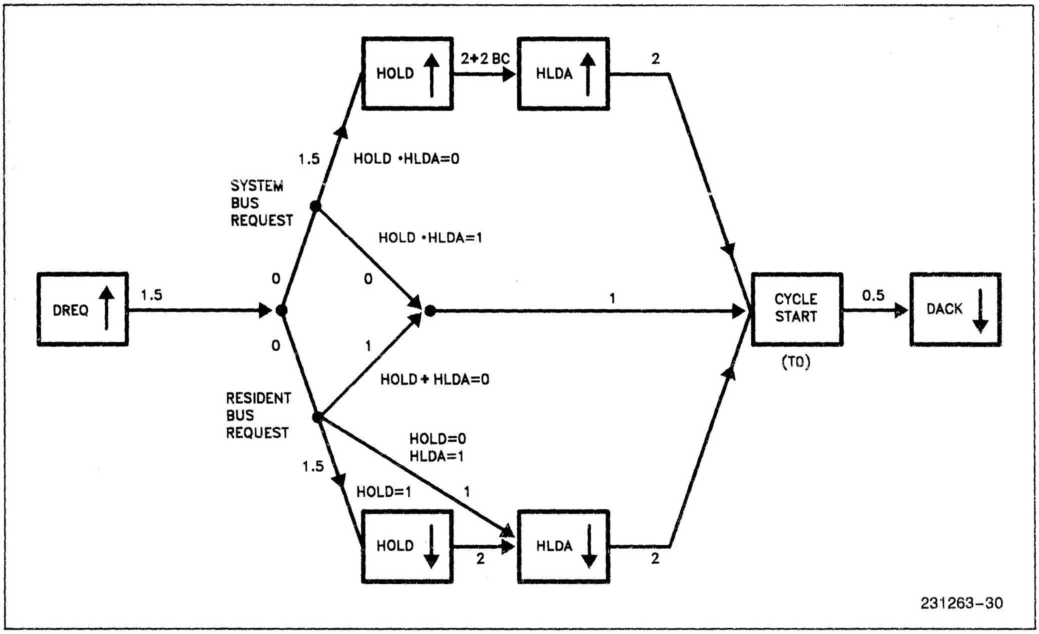


Figure 33. DREQ to DACK Latency in Remote Mode*

Table 11. DREQ to DACK in Remote Mode*

	Minimum	Typical	Maximum
DREQ to HOLDset	2.5	3	3 + W (1) (2)
HOLDset to HLDAset	2 BC	2 + 2 BC	(4)
HOLDAset TO CYCLE START	1.5	2	2.5
DREQ to HOLDreset	1.5	3	5.5 + W (1)
HOLDreset to HLDAreset	1	2	2
HLDAreset to CYCLE START	1.5	2	2.5
DREQ to CYCLE START (without bus change)	2	3.5	5 + W (1)
CYCLE START to DACK	0.5	0.5	0.5

Notes:

(1) Single bus cycle running: 1 + W
unseparable bus cycles running:
—word access at odd addresses (and pointer transfers): 3 + 2W
—IOACK cycle (only multiplexor channel): 7 + 2W

(2) General Burst Counter = 0: 2 × GDR
HLDA = 1, HOLD = 0: Wait for HLDA = 0
HLDA lost: 2

(3) 16 + 15W (from the 286 manual, assumed repeat and lock prefix not combined)

(4) Bus arbitration + currently running bus transfers.
BC = Multibus clock cycle.

* All timings are in units of T-states (125 ns in an 8 MHz system).
If bus cycles are involved then the following abbreviations are used:
T = Time for one bus transfer
W = Wait time during bus cycles for a slow device

General Command Processing:*

	Minimum	Typical	Maximum
WRITE to Set Up	6.5	8	9.5
	+ HOLD/HOLDA sequence		

At this point the start command is ready for the start of the channel set up routine

Set Up Processing:*

Standard command block	: 7T + 4
additional for long command block	: 5T
additional for list data chaining	: 1T + 2
additional for linked list data chaining	: 3T + 2

Type 1 Command Processing:*

Chaining : same as the set up processing

Termination :

store CSR and calculate next command pointer	: 1T + 6
store status block (if programmed)	: 6T

Type 2 Command Processing:*

Standard :

CCR load	: 1T
CCR decode and execution	: 2T + 2
additional for jump	: 4

START/STOP Subchannel:*

(see General Command Processing for set up)

Execution : 4T + 6

Multiplexor Channel:*

(see General Command Processing for set up)

IOREQ to IOACK: identical to DREQ to DACK timing

First IOACK to second IOACK	: 1T + 2
Second IOACK to vector in LVR	: 1T + 2
Calculate MT address and read command pointer into CPR	: 2T + 4
Data transfer	: 2T + 2
Restore pointers	: 4T + 4
Restore byte count	: 2T

Data Chaining:*

Latencies in data chaining occur when transfers are changed between data blocks.

List Chaining	: 3T + 6
Linked List Chaining	: 5T + 6

* All timings are in units of T-states (125 ns in an 8 MHz system).

If bus cycles are involved then the following abbreviations are used:

T = Time for one bus transfer
W = Wait time during bus cycles for a slow device

Absolute Maximum Ratings

Ambient Temperature Under Bias	0°C to 55°C
Case Temperature	0°C to 85°C
Storage Temperature	−65°C to +150°C
Voltage on Any Pin with Respect to Ground	−1.0V to +7V
Power Dissipation	3.6 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. Characteristics $V_{CC}=5V \pm 5\%$; $T_A = 0°C$ to $+55°C$, or $T_{CASE} = 0°C$ to $+85°C$

Symbol	Parameter	Limit Values		Units	Test Conditions
		Min	Max		
V_{IL}	Input Low Voltage (except CLK)	−0.5	+0.8	V	—
V_{IH}	Input High Voltage (except CLK)	2.0	V_{CC} + 0.5		
V_{OL}	Output Low Voltage	—	0.45		I_{OL} = 3.00 mA
V_{OH}	Output High Voltage	2.4	—		I_{OH} = −400 μA
I_{CC}	Power Supply Current	—	450	mA	T_A = 25°C, all outputs open
I_{LI}	Input Leakage Current		±10	μA	0V ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current				0.45V ≤ V_{OUT} = V_{CC}
	$\overline{S0}$, $\overline{S1}$, $\overline{S2}$, $\overline{BHE}$, $\overline{RD}$, $\overline{WR}$, M/$\overline{IO}$		−200	μA	
	HOLD (RQ/GT mode), $\overline{EOD}$		−1.5	mA	
	other pins		± 10	μA	
V_{CL}	Clock Input Low Voltage	−0.5	+0.6	V	—
V_{CH}	Clock Input High Voltage	3.8	V_{CC}+1.0		
C_{IN}	Capacitance of Inputs (except CLK)	—	10	pF	fc = 1 MHz
C_O	Capacitance of I/O or Outputs		20		
CCLK	Capacitance of CLK Input		12		

A.C. Characteristics $V_{CC} = 5V \pm 5\%$; $T_A = 0°C$ to $+55°C$, or $T_{CASE} = 0°C$ to $+85°C$

AC timings are referenced to 0.8V and 2.0V points of signals as illustrated in datasheet waveforms, unless otherwise noted.

Sym	Parameter	6 MHz		8 MHz		Unit	Test Conditions
		Min	Max	Min	Max		
1	CLK Cycle Period (286 Mode)	83	250	62	250	ns	
2	CLK Low Time (286 Mode)	20	225	15	230	ns	at 0.6V
3	CLK High Time (286 Mode)	25	230	20	235	ns	at 3.2V
4	Output Valid Delay	1	80	1	60	ns	CL = 125 pF
5	Output Valid Delay	1	55	1	40	ns	CL = 125 pF
6	Data Setup Time	15		10		ns	
6a	Address Input Setup (186 Mode)	20		15		ns	
7	Data Hold Time	8		5		ns	
8	READY Setup Time	50		38		ns	
9	READY Hold Time	35		25		ns	
10	Input Setup Time	25		20		ns	
11	Input Hold Time	25		20		ns	
11a	BHE Hold Time (186 Mode)	15		10		ns	
12	Address Setup Time	3		2		ns	
13	Data Valid Delay	0	60	0	50	ns	
14	Data Float Delay	8	80	5	60	ns	
15	Chip Select Setup	30		20		ns	
16	Command Length	320		290		ns	
17	Data Setup Time	185		165		ns	
18	Address Setup Time	30		20		ns	
19	Command Inactive	320		290		ns	
19a	Access Time		420		380	ns	
20	CLK Period (186 Mode)	166	500	125	500	ns	
21	CLK Low Time (186 Mode)	76		55		ns	
22	CLK High Time (186 Mode)	76		55		ns	
23	CLK Rise Time (186 Mode)		15		15	ns	
24	CLK Fall Time (186 Mode)		15		15	ns	
25	READY Active Setup Time	20		20		ns	
26	READY Hold Time	10		10		ns	
26a	SREADY Hold Time (186 Mode)	15		15		ns	
27	READY Inactive Setup Time	35		35		ns	
28	Control Reset Setup Time	25		20		ns	
29	Reset Control Setup Time	0		0		ns	
30	Address/Data Valid Delay	10	55	10	50	ns	
31	Status Delay	10	75	10	55	ns	
32	Address/Data Float Delay	10	50	10	50	ns	
33	DT/$\overline{R}$ Delay (186 Mode)	10	76	10	55	ns	
34	DEN Delay (186 Mode)	10	80	10	60	ns	

A.C. MEASUREMENT POINT DESCRIPTION

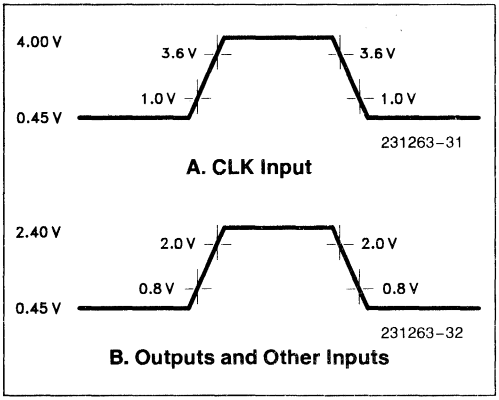


Figure 33a. AC Drive and Measurement Points

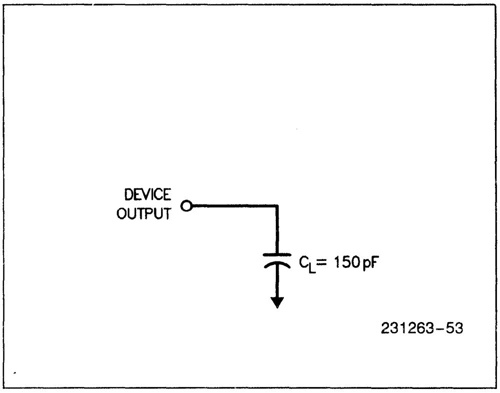


Figure 33b. AC Test Loading on Outputs

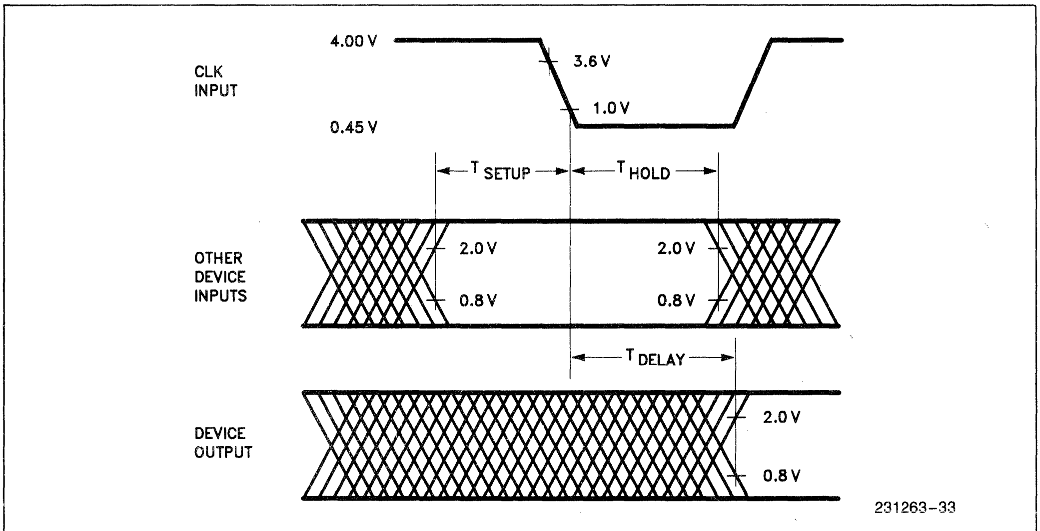


Figure 34. AC Setup, Hold and Delay Time Measurement - General

BUS CYCLE T-STATES:

The bus cycles are subdivided into T-states which are interpreted differently depending on whether the 82258 is in the 286 mode or the 186 mode.

286 Mode T-states: Each T-state is two clock cycles long and starts in the middle of a processor cycle and ends in the middle of the succeeding processor cycle.

TI: [The bus is idle] This state will occur if the 82258 cannot start the next bus cycle.

T0: [A new bus cycle is beginning] When the address and status of a new bus cycle is to be sent as output, this state is used.

T1: [A bus cycle is proceeding] This state is used to allow the bus controller commands to become active and, to output data during a write cycle.

T2I: [A bus cycle is prepared for termination with no new cycle ready to begin] If the $\overline{READY}$ signal is active and no new bus cycle is ready to begin, this will be the state used. Input data will be accepted during this state if the $\overline{READY}$ signal is active and if the bus cycle is an input cycle.

T20: [A bus cycle is prepared for termination with a new cycle ready to begin] This state terminates a bus cycle if the $\overline{READY}$ signal is active and if a new bus cycle is ready to begin. As with the T2I state, input data will be accepted during this state if the cycle is an input cycle and if the $\overline{READY}$ signal is active. This state will also output the address of the new bus cycle, and if $\overline{READY}$ is active, the status also.

186 Mode T-states: The T-states are one CLK period long, beginning and ending with the falling edge of the CLK signal.

TI: [The bus is idle] This state occurs if the 82258 cannot start the following bus cycle.

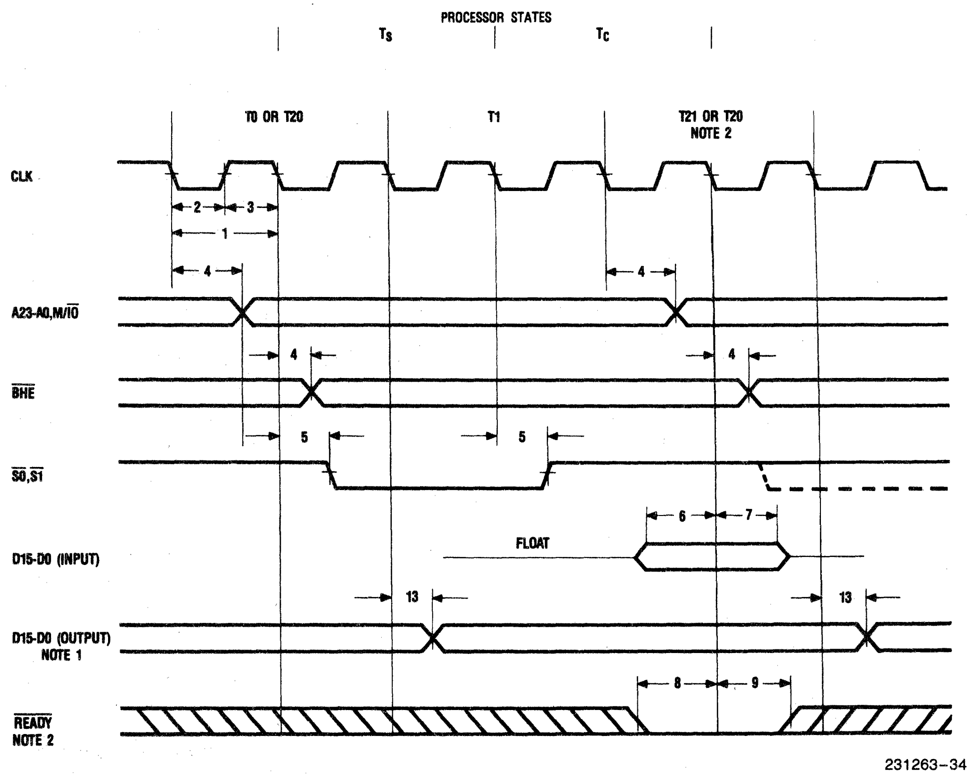
T1: [The first bus cycle T-state] During this state, address information is output to the A19/S6–A16/S3 and AD15–AD0 pins. The status is activated with the rising edge of the CLK previous to this state.

T2: [The second bus cycle T-state] This state allows the bus controller and the 82258 commands to become active and outputs data if the cycle is a write cycle.

T3: [The third bus cycle T-state] This state is used to synchronize the ready signals. If the bus is not ready, then the bus cycle is extended by repeating this state, with the status lines going inactive during the last T3-state.

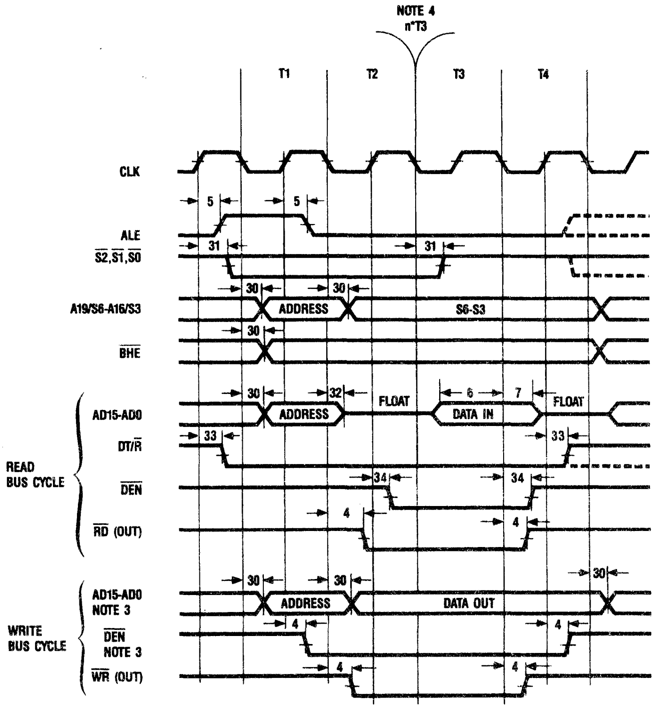
T4: [The last bus cycle T-state] During this cycle, data is input for input cycles and the bus controller and the 82258 commands are disactivated. If the following state is T1, then the status is activated during this state.

Waveforms



NOTES:
1. D15–D0 floats during Single Cycle Transfer like a Read Cycle.
2. T2 will be repeated, if READY is inactive.

Figure 35. Timing of an Active Bus Cycle (286 and Remote modes)

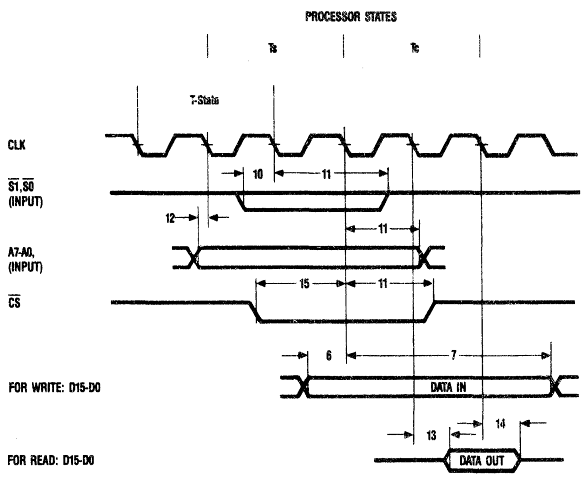


231263-35

NOTES:

3. For a Single Cycle Transfer the timings of AD15–AD0, $\overline{DEN}$ and DT/$\overline{R}$ are the same as in a Read Bus Cycle.
4. Additional T3 cycles will be inserted if bus is not ready (see Figure 40).

Figure 36. Timing of an Active Bus Cycle (186 and 8086 Modes)



231263-36

Figure 37. Timing of a Synchronous Access to the 82258 (286 Mode)

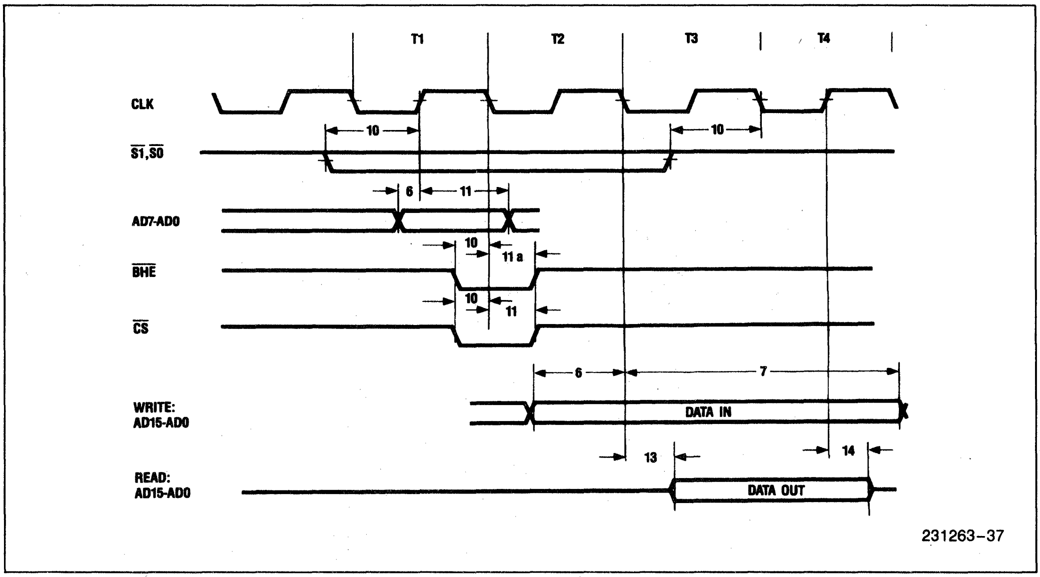


Figure 38. Timing of a Synchronous Access to the 82258 (186 and 8086 Modes)

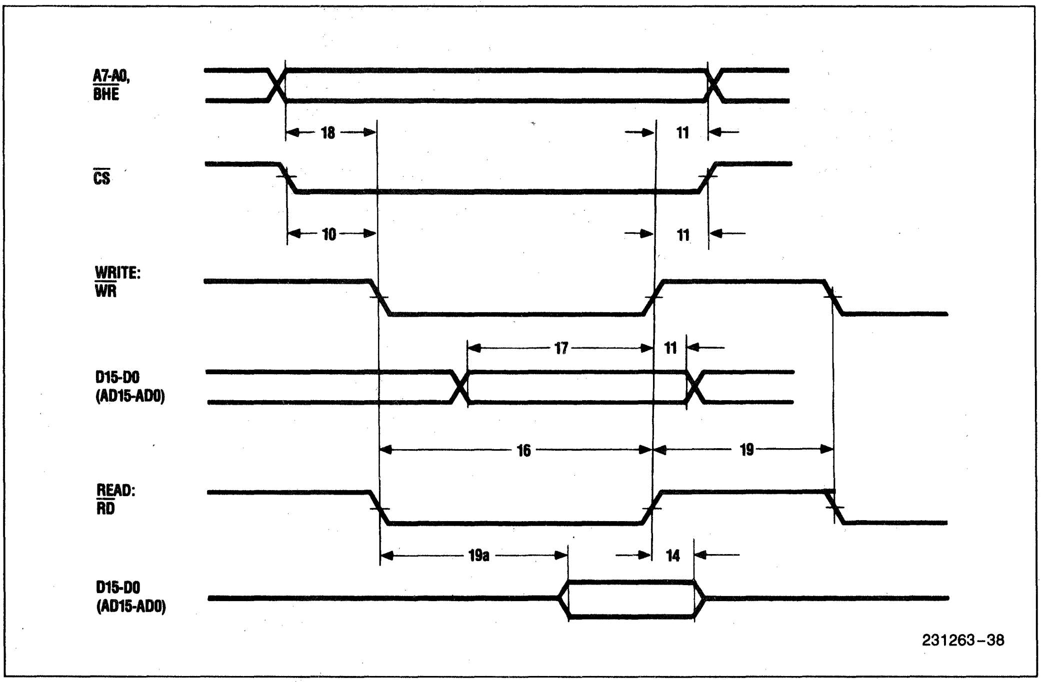


Figure 39. Timing of an Asynchronous Access to the 82258 (All Modes)

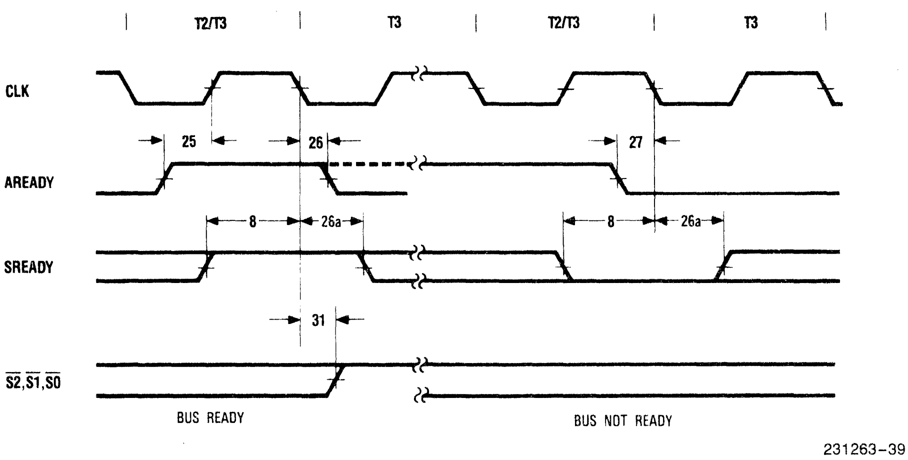
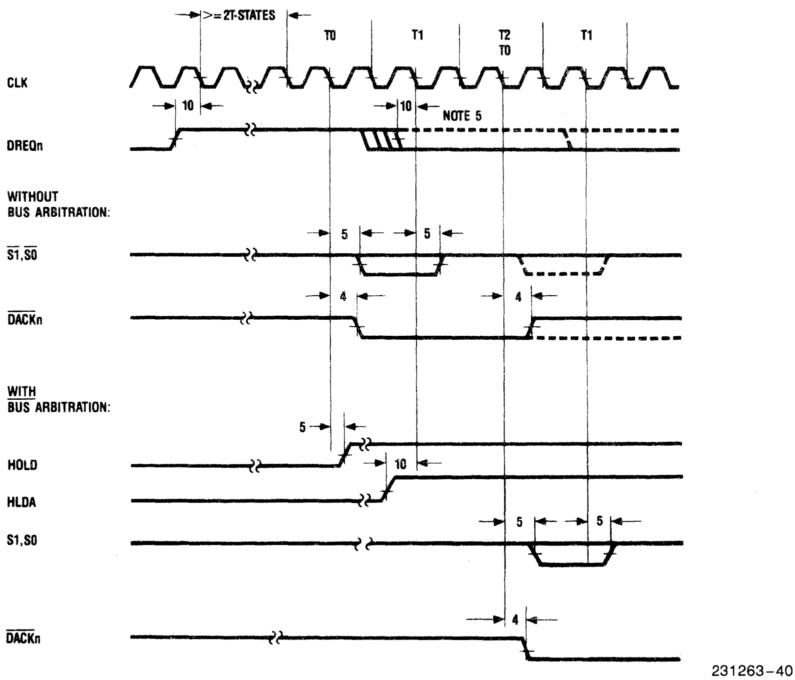
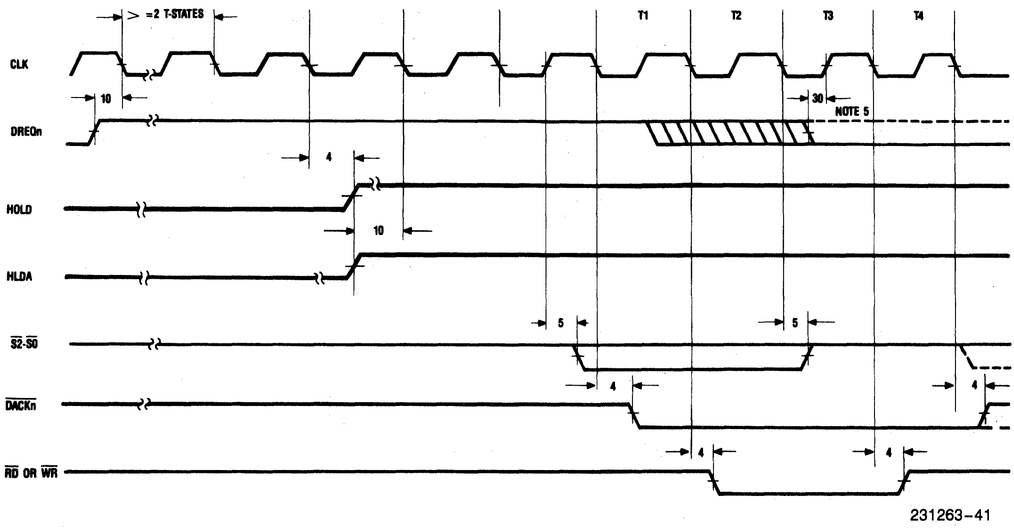


Figure 40. READY Timing (186 Mode)



NOTE:
5. The trailing edge of DREQn, as specified in this diagram, is necessary if only one bus cycle should be executed. A later trailing edge may cause an additional bus cycle (continuous DREQ), if no READY-wait-states are inserted.

Figure 41. DREQ, $\overline{\text{DACK}}$ Timing (286 and Remote Modes)



NOTE:
The DREQ and $\overline{DACK}$ signal timings are the same for the 8086 mode.

Figure 42. DREQ, $\overline{DACK}$ Timing (186 Mode)

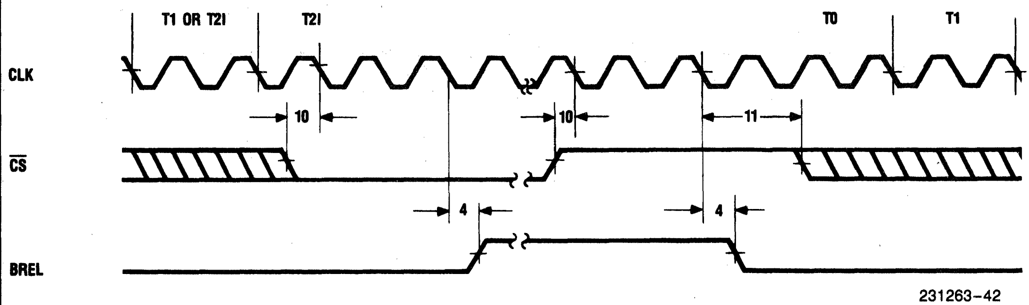


Figure 43. BREL, Bus Tristate Timing (Remote Mode)

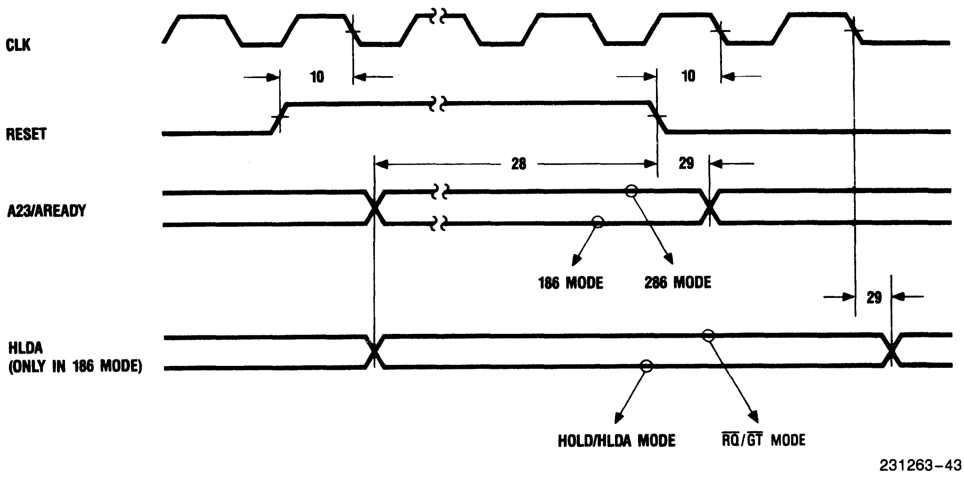
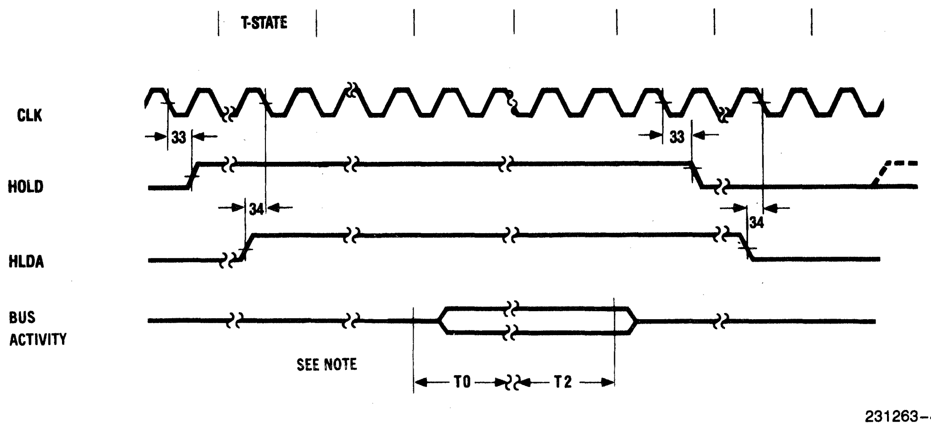
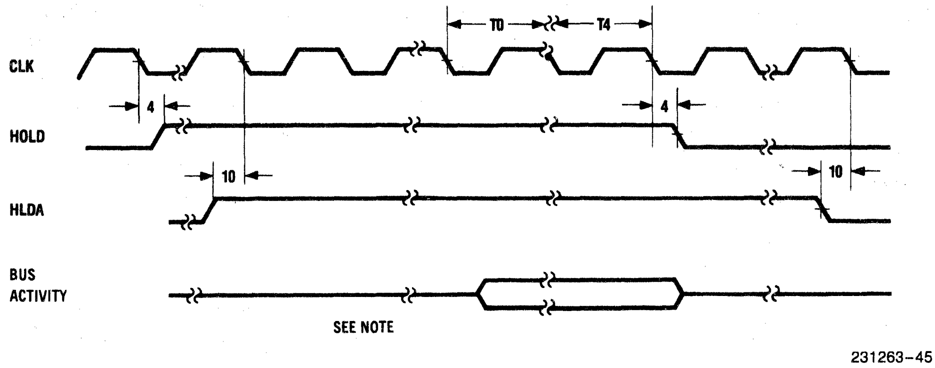


Figure 44. RESET Timing (All Modes)



NOTE:
See Figure 32 and Table 10 (Pg. 41) for HLDA to Cycle Start Latency.

Figure 45. HOLD, HLDA Timing (286 and Remote Modes)



NOTE:
See Figure 32 and Table 10 (Pg. 41) for HLDA to Cycle Start Latency.

Figure 46. HOLD, HLDA Timing (186 Mode)

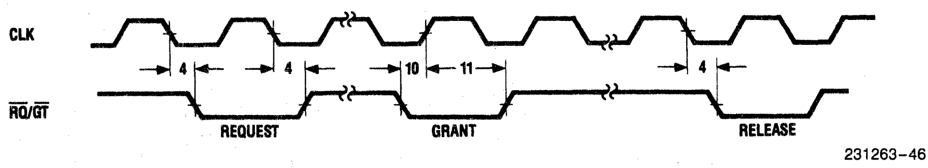


Figure 47. $\overline{RQ}/\overline{GT}$ Timing (8086 Mode)

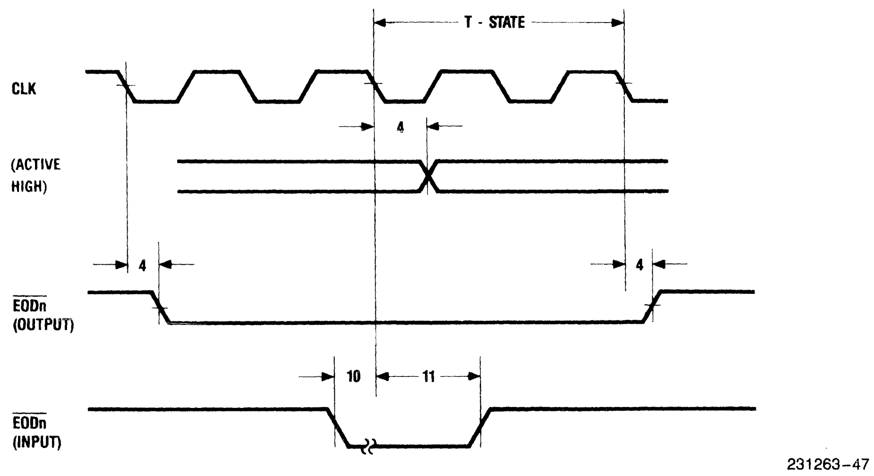


Figure 48. INTOUT, $\overline{\text{EOD}}$ Timing (286 and Remote Modes)

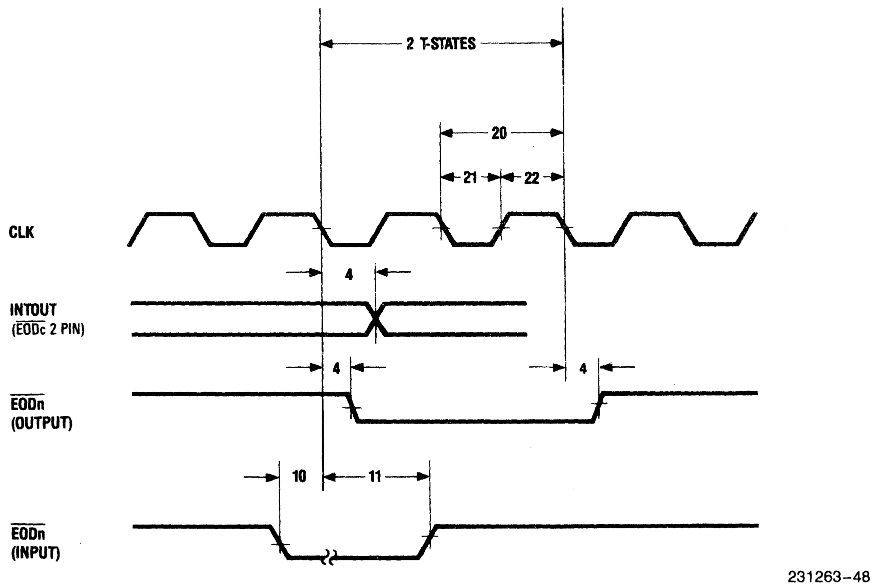


Figure 49. INTOUT, $\overline{\text{EOD}}$ Timing (186 and 8086 Modes)

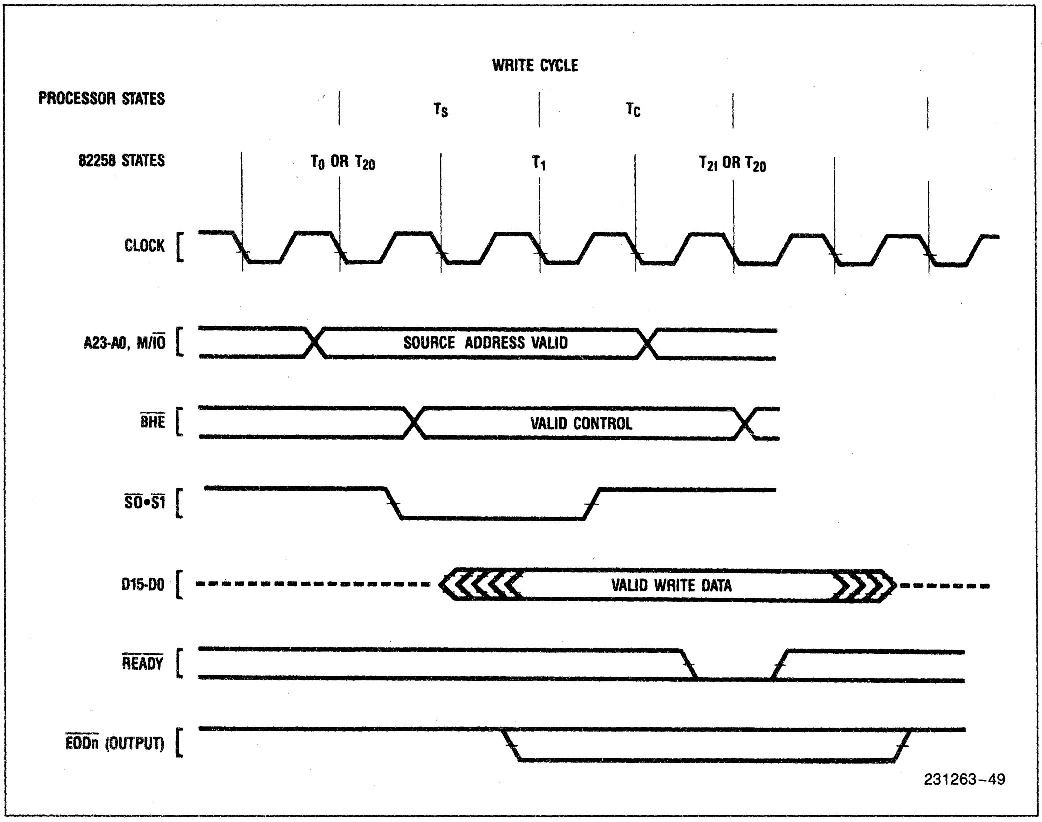


Figure 50. Single Cycle Transfer (286 mode)

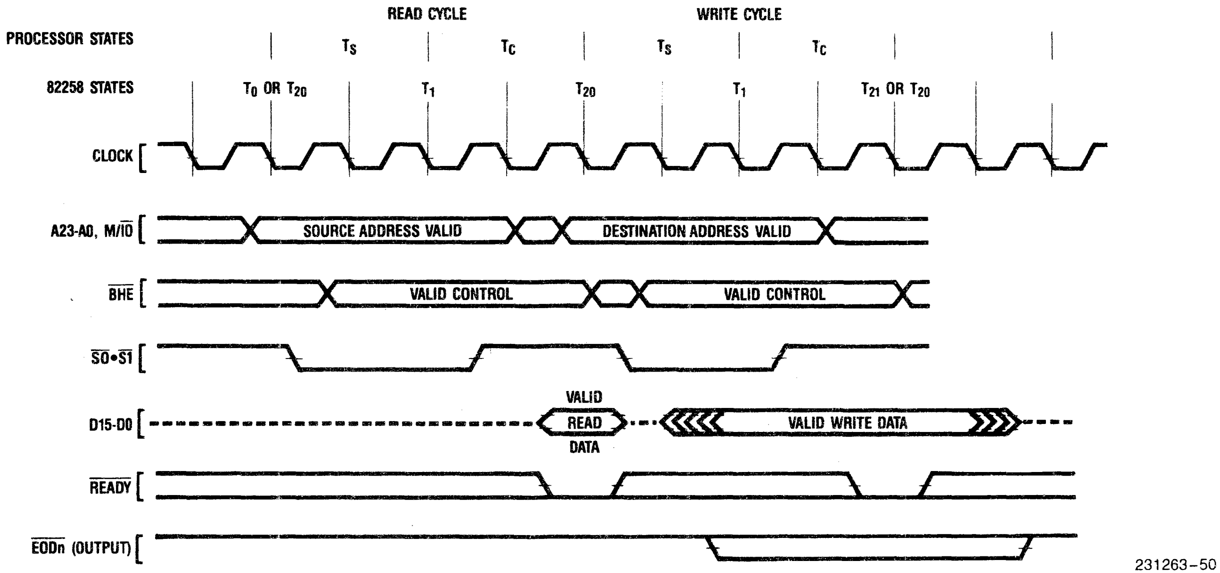
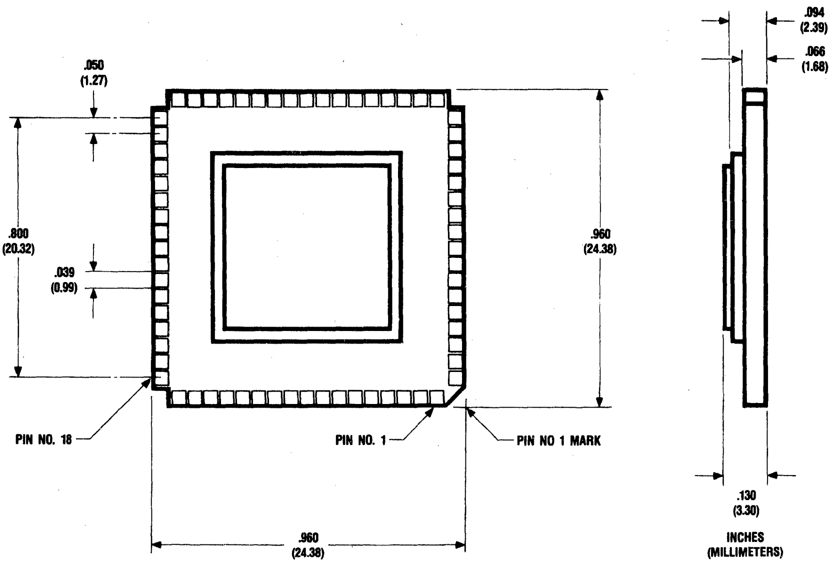


Figure 51. Two Cycle Transfer (286 mode)

PACKAGE

The 82258 is packaged in a 68-pin, leadless JEDEC type A hermetic leadless chip carrier.



231263-51

Figure 52. JEDEC Type A Package

intel

82284
CLOCK GENERATOR AND READY INTERFACE FOR iAPX 286 PROCESSORS

(82284-10, 82284-8, 82284-6)

- **Generates System Clock for iAPX 286 Processors**
- **Uses Crystal or TTL Signal for Frequency Source**
- **Provides Local READY and MULTIBUS®* READY Synchronization**
- **Available in 18-Lead Cerdip Package (See Packaging Spec, Order #231369)**
- **Single +5V Power Supply**
- **Generates System Reset Output from Schmitt Trigger Input**
- **Available in EXPRESS**
  - **Standard Temperature Range**
  - **Extended Temperature Range**

The 82284 is a clock generator/driver which provides clock signals for iAPX 286 processors and support components. It also contains logic to supply READY to the CPU from either asynchronous or synchronous sources and synchronous RESET from an asynchronous input with hysteresis.

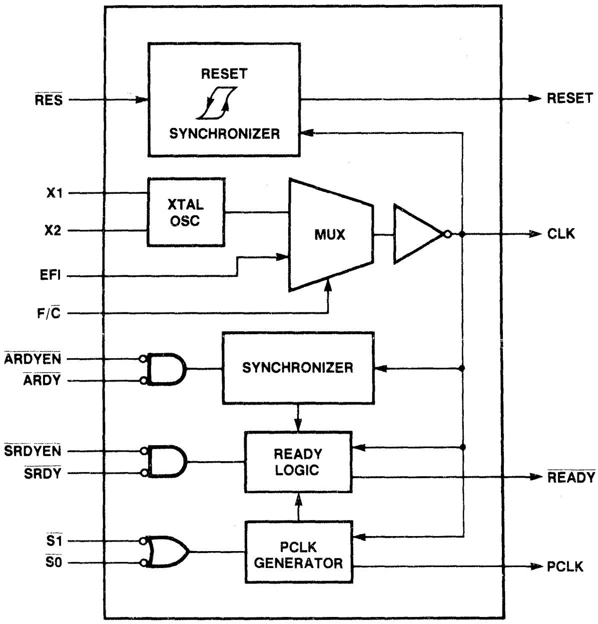


Figure 1. 82284 Block Diagram

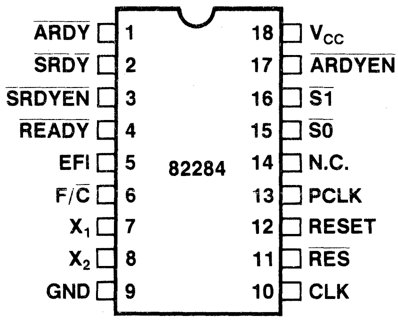


Figure 2. 82284 Pin Configuration

*MULTIBUS is a patented bus of Intel.

Table 1. Pin Description

The following pin function descriptions are for the 82284 clock generator.

Symbol	Type	Name and Function
CLK	O	**System Clock** is the signal used by the processor and support devices which must be synchronous with the processor. The frequency of the CLK output has twice the desired internal processor clock frequency. CLK can drive both TTL and MOS level inputs.
F/C	I	**Frequency/Crystal Select** is a strapping option to select the source for the CLK output. When F/C is strapped LOW, the internal crystal oscillator drives CLK. When F/C is strapped HIGH, the EFI input drives the CLK output.
X1, X2	I	**Crystal In** are the pins to which a parallel resonant fundamental mode crystal is attached for the internal oscillator. When F/C is LOW, the internal oscillator will drive the CLK output at the crystal frequency. The crystal frequency must be twice the desired internal processor clock frequency.
EFI	I	**External Frequency In** drives CLK when the F/C input is strapped HIGH. The EFI input frequency must be twice the desired internal processor clock frequency.
PCLK	O	**Peripheral Clock** is an output which provides a 50% duty cycle clock with 1/2 the frequency of CLK. PLCK will be in phase with the internal processor clock following the first bus cycle after the processor has been reset.
ARDYEN	I	**Asynchronous Ready Enable** is an active LOW input which qualifies the ARDY input. ARDYEN selects ARDY as the source of ready for the current bus cycle. Inputs to ARDYEN may be applied asynchronously to CLK. Setup and hold times are given to assure a guaranteed response to synchronous inputs.
ARDY	I	**Asynchronous Ready** is an active LOW input used to terminate the current bus cycle. The ARDY input is qualified by ARDYEN. Inputs to ARDY may be applied asynchronously to CLK. Setup and hold times are given to assure aguaranteedresponse to synchronous inputs.
SRDYEN	I	**Synchronous Ready Enable** is an active LOW input which qualifies SRDY. SRDYEN selects SRDY as the source for READY to the CPU for the current bus cycle. Setup and hold times must be satisfied for proper operation.
SRDY	I	**Synchronous Ready** is an active LOW input used to terminate the current bus cycle. The SRDY input is qualified by the SRDYEN input. Setup and hold times must be satisfied for proper operation.
READY	O	**Ready** is an active LOW output which signals the current bus cycle is to be completed. The SRDY, SRDYEN, ARDY, ARDYEN, S1, S0 and RES inputs control READY as explained later in the READY generator section. READY is an open collector output requiring an external pullup resistor.
S0, S1	I	**Status** inputs prepare the 82284 for a subsequent bus cycle. S0 and S1 synchronize PCLK to the internal processor clock and control READY. These inputs have pullup resistors to keep them HIGH if nothing is driving them. Setup and hold times must be satisfied for proper operation.
RESET	O	**Reset** is an active HIGH output which is derived from the RES input. RESET is used to force the system into an initial state. When RESET is active, READY will be active (LOW).
RES	I	**Reset In** is an active LOW input which generates the system reset signal RESET. Signals to RES may be applied asynchronously to CLK. A Schmitt trigger input is provided on RES, so that an RC circuit can be used to provide a time delay. Setup and hold times are given to assure a guaranteed response to synchronous inputs.
V_{CC}		**System Power**: +5V power supply
GND		**System Ground**: 0 volts

FUNCTIONAL DESCRIPTION

Introduction

The 82284 generates the clock, ready, and reset signals required for iAPX 286 processors and support components. The 82284 is packaged in an 18-pin DIP and contains a crystal controlled oscillator, MOS clock generator, peripheral clock generator, Multibus ready synchronization logic and system reset generation logic.

Clock Generator

The CLK output provides the basic timing control for an iAPX 286 system. CLK has output characteristics sufficient to drive MOS devices. CLK is generated by either an internal crystal oscillator or an external source as selected by the F/C strapping option. When

F/$\overline{C}$ is LOW, the crystal oscillator drives the CLK output. When F/$\overline{C}$ is HIGH, the EFI input drives the CLK output.

The 82284 provides a second clock output (PCLK) for peripheral devices. PCLK is CLK divided by two. PCLK has a duty cycle of 50% and TTL output drive characteristics. PCLK is normally synchronized to the internal processor clock.

After reset, the PCLK signal may be out of phase with the internal processor clock. The $\overline{S1}$ and $\overline{S0}$ signals of the first bus cycle are used to synchronize PCLK to the internal processor clock. The phase of the PCLK output changes by extending its HIGH time beyond one system clock (see waveforms). PCLK is forced HIGH whenever either $\overline{S0}$ or $\overline{S1}$ were active (LOW) for the two previous CLK cycles. PCLK continues to oscillate when both $\overline{S0}$ and $\overline{S1}$ are HIGH.

Since the phase of the internal processor clock will not change except during reset, the phase of PCLK will not change except during the first bus cycle after reset.

Oscillator

The oscillator circuit of the 82284 is a linear Pierce oscillator which requires an external parallel resonant, fundamental mode, crystal. The output of the oscillator is internally buffered. The crystal frequency chosen should be twice the required internal processor clock frequency. The crystal should have a typical load capacitance of 32 pF.

X1 and X2 are the oscillator crystal connections. For stable operation of the oscillator, two loading capacitors are recommended, as shown in Table 2. The sum of the board capacitance and loading capacitance should equal the values shown. It is advisable to limit stray board capacitances (not including the effect of the loading capacitors or crystal capacitance) to less than 10 pF between the X1 and X2 pines. Decouple V_{CC} and GND as close to the 82284 as possible.

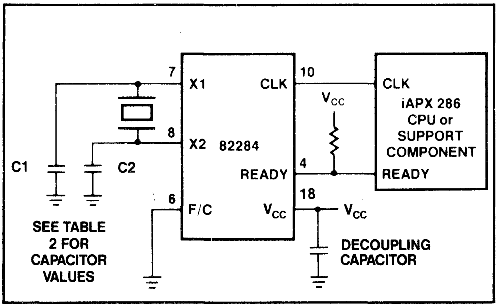


Figure 3. Recommended Crystal and $\overline{READY}$ Connections

Reset Operation

The reset logic provides the RESET output to force the system into a known, initial state. When the $\overline{RES}$ input is active (LOW), the RESET output becomes active (HIGH). $\overline{RES}$ is synchronized internally at the falling edge of CLK before generating the RESET output (see waveforms). Synchronization of the $\overline{RES}$ input introduces a one or two CLK delay before affecting the RESET output.

At power up, a system does not have have a stable V_{CC} and CLK. To prevent spurious activity, $\overline{RES}$ should be asserted until V_{CC} and CLK stabilize at their operating values. iAPX 286 processors and support components also require their RESET inputs be HIGH a minimum of 16 CLK cycles. An RC network, as shown in Figure 4, will keep $\overline{RES}$ LOW long enough to satisfy both needs.

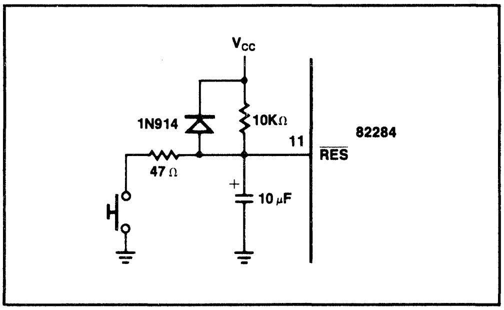


Figure 4. Typical RC $\overline{RES}$ Timing Circuit

A Schmitt trigger input with hysteresis on $\overline{RES}$ assures a single transition of RESET with an RC circuit on $\overline{RES}$. The hysteresis separates the input voltage level at which the circuit output switches between HIGH to LOW from the input voltage level at which the circuit output switches between LOW to HIGH. The $\overline{RES}$ HIGH to LOW input transition voltage is lower than the $\overline{RES}$ LOW to HIGH input transition voltage. As long as the slope of the $\overline{RES}$ input voltage remains in the same direction (increasing or decreasing) around the $\overline{RES}$ input transition voltage, the RESET output will make a single transition.

Ready Operation

The 82284 accepts two ready sources for the system ready signal which terminates the current bus cycle. Either a synchronous ($\overline{SRDY}$) or asynchronous ready ($\overline{ARDY}$) source may be used. Each ready input has an enable ($\overline{SRDYEN}$ and $\overline{ARDYEN}$) for selecting the type of ready source required to terminate the current bus cycle. An address decoder would normally select one of the enable inputs.

READY is enabled (LOW), if either SRDY + SRDYEN = 0 or ARDY + ARDYEN = 0 when sampled by the 82284 READY generation logic. READY will remain active for at least two CLK cycles.

The READY output has an open-collector driver allowing other ready circuits to be wire or'ed with it, as shown in Figure 3. The READY signal of an iAPX 286 system requires an external pull-up resistor. To force the READY signal inactive (HIGH) at the start of a bus cycle, the READY output floats when either S1 or S0 are sampled LOW at the falling edge of CLK. Two system clock periods are allowed for the pull-up resistor to pull the READY signal to V_{IH}. When RESET is active, READY is forced active one CLK later (see waveforms).

Figure 5 illustrates the operation of SRDY and SRDYEN. These inputs are sampled on the falling edge of CLK when S1 and S0 are inactive and PCLK is HIGH. READY is forced active when both SRDY and SRDYEN are sampled as LOW.

Figure 6 shows the operation of ARDY and ARDYEN. These inputs are sampled by an internal synchronizer at each falling edge of CLK. The output of the synchronizer is then sampled when PCLK is HIGH. If the synchronizer resolved both the ARDY and ARDYEN have been resolved as active, the SRDY and SRDYEN inputs are ignored. Either ARDY or ARDYEN must be HIGH at end of T_S (see figure 6).

READY remains active until either S1 or S0 are sampled LOW, or the ready inputs are sampled as inactive.

Table 2. 82284 Crystal Loading Capacitance Values

Crystal Frequency	C1 Capacitance (pin 7)	C2 Capacitance (pin 8)
1 to 8 MHz	60 pF	40 pF
8 to 20 MHz	25 pF	15 pF

NOTE: Capacitance values must include stray board capacitance.

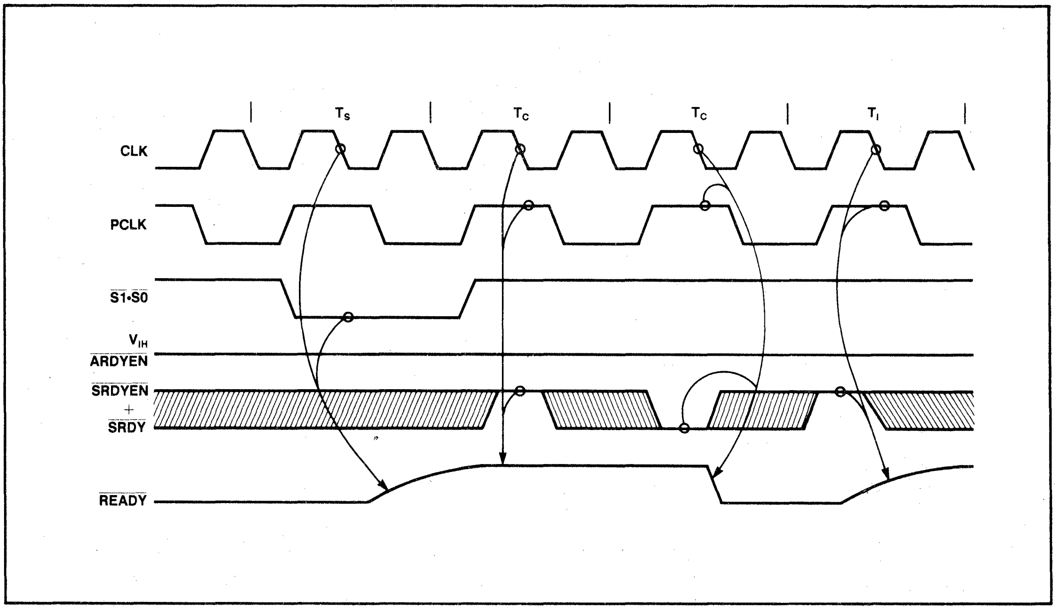


Figure 5. Synchronous Ready Operation

210453-004

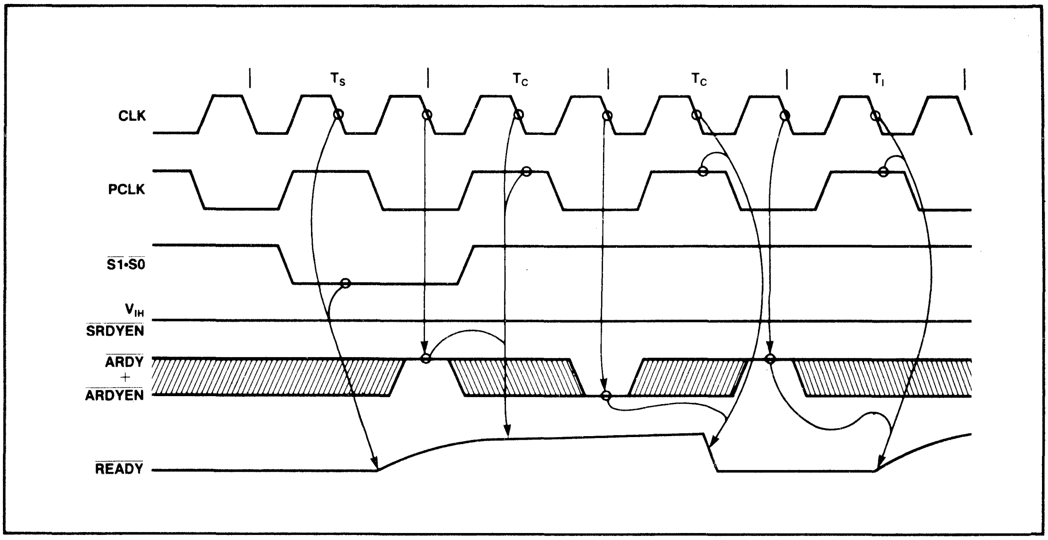


Figure 6. Asynchronous Ready Operation

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... 0°C to 70°C

Storage Temperature .... -65°C to +150°C

All Output and Supply Voltages ......-0.5V to +7V

All Input Voltages .... -1.0V to +5.5V

Power Dissipation .... 1 Watt

**Notice: Stresses above those listed under "Absolute Maxmum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, or T_{CASE} = 0°C to +85°C, V_{CC} = 5V ± 5%)

		6 MHz		8 MHz			
Sym	Parameter	-6 Min	-6 Max	-8 Min	-8 Max	Unit	Test Condition
V_{IL}	Input LOW Voltage		.8		.8	V	
V_{IH}	Input HIGH Voltage	2.0		2.0		V	
V_{IHR}	$\overline{RES}$ and EFI Input HIGH Voltage	2.6		2.6		V	
V_{HYS}	$\overline{RES}$ Input hysteresis	0.25		0.25		V	
V_{OL}	RESET, PCLK Output LOW Voltage		.45		.45	V	I_{OL} = 5mA
V_{OH}	RESET, PCLK Output HIGH Voltage	2.4		2.4		V	I_{OH} = -1mA
V_{OLR}	$\overline{READY}$, Output LOW Voltage		.45		.45	V	I_{OL} = 7mA
V_{OLC}	CLK Output LOW Voltage		.45		.45	V	I_{OL} = 5mA
V_{OHC}	CLK Output HIGH Voltage	4.0		4.0		V	I_{OH} = -800μA
V_C	Input Forward Clamp Voltage		-1.0		-1.0	V	I_C = -5mA
I_F	Forward Input Current		-.5		-.5	mA	V_F = .45V
I_R	Reverse Input Current		50		50	uA	$V_R = V_{CC}$
I_{CC}	Power Supply Current		145		145	mA	
C_I	Input Capacitance		10		10	pF	F_C = 1MHz

210453-004

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, or T_{CASE} = 0°C to +85°C, V_{CC} = 5V, ±5%)
AC timings are referenced to 0.8V and 2.0V points of signals as illustrated in datasheet waveforms, unless otherwise noted.

		6 MHz		8 MHz		10 MHz (Preliminary)			
Sym	**Parameter**	**-6 Min.**	**-6 Max.**	**-8 Min.**	**-8 Max.**	**-10 Min.**	**-10 Max.**	**Unit**	**Test Condition**
1	EFI to CLK Delay		35		30		30	ns	at 1.5V Note 1
2	EFI LOW Time	40		22		25		ns	at 1.5V Note 1 Note 7
3	EFI HIGH Time	35		40		25		ns	at 1.5V Note 1 Note 7
4	CLK Period	83	500	62	500	50	500	ns	
5	CLK LOW Time	20		15		12		ns	at 1.0V Note 1 Note 2, 7, 8
6	CLK HIGH Time	25		25		16		ns	at 3.6V Note 1 Note 2, 7, 8
7	CLK Rise Time		10		10		8	ns	1.0V to 3.6V Note 1
8	CLK Fall Time		10		10		8	ns	3.6V to 1.0V Note 1
9	Status Setup Time	28		22		—		ns	Note 1
9a	Status Setup Time for Status Going Active	—		—		20		ns	Note 1
9b	Status Setup Time for Status Going Active	—		—		20		ns	Note 1
10	Status Hold Time	0		1		1		ns	Note 1
11	$\overline{SRDY}$ or $\overline{SRDYEN}$ Setup Time	25		17		15		ns	Note 1
12	$\overline{SRDY}$ or $\overline{SRDYEN}$ Hold Time	0		0		0		ns	Note 1
13	$\overline{ARDY}$ or $\overline{ARDYEN}$ Setup Time	5		0		0		ns	Note 1 Note 3
14	$\overline{ARDY}$ or $\overline{ARDYEN}$ Hold Time	30		30		30		ns	Note 1 Note 3
15	$\overline{RES}$ Setup Time	25		20		20		ns	Note 1 Note 3
16	$\overline{RES}$ Hold Time	10		10		10		ns	Note 1 Note 3
17	$\overline{READY}$ Inactive Delay	5		5		5		ns	at 0.8V Note 4
18	$\overline{READY}$ Active Delay	0	33	0	24	0	24	ns	at 0.8V Note 4
19	PCLK Delay	0	45	0	45	0	35	ns	Note 5
20	RESET Delay	5	50	5	34	5	27	ns	Note 5
21	PCLK LOW Time	t4-20		t4-20		t4-20		ns	Note 5 Note 6
22	PCLK HIGH Time	t4-20		t4-20		t4-20		ns	Notes 5 Note 6

NOTE 1: CLK loading: C_L = 150pF. The 82284's X1 and X2 inputs are designed primarily for parallel-resonant crystals. Serial-resonant crystals may also be used, however, they may oscillate up to .01% faster than their nominal frequencies when used with the 82284. For either type of crystal, capacitive loading should be as specified by Table 2.

NOTE 2: With the internal crystal oscillator using recommended crystal and capacitive loading; or with the EFI input meeting specifications t2 and t3. The recommended crystal loading for CLK frequencies of 8-20 MHz are 25pF from pin X_1 to ground, and 15pF from pin X_2 to ground. These recommended values are ±5pF and include all stray capacitance. Decouple V_{CC} and GND as close to the 82284 as possible.

NOTE 3: This is an asychronous input. This specification is given for testing purposes only, to assure recognition at specific CLK edge.

NOTE 4: Pull-up Resistor values for READY Pin:

CPU Frequency	6-8 MHz	10 MHz
Resistor CL I_{OL}	910 ohm 150 pF 7 mA	700 ohm 150 pF 7 mA

NOTE 5: PCLK and RESET loading: C_L = 75pF. PCLK also has 750 ohm pullup resistor.

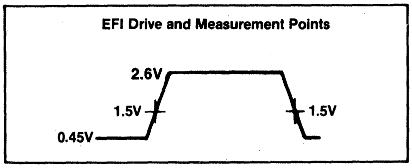
NOTE 6: t4 refers to any allowable CLK period.

CLK Output Frequency:	12MHz CLK	16MHz CLK*	20MHz CLK*
Min. required EFI HIGH time	35ns	40ns	25ns
Min. required EFI LOW time	40ns	22ns	25ns

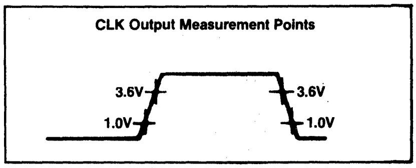
*At CLK Frequencies above 12MHz, CLK output HIGH and LOW times are guaranteed only when using crystal with recommended capacitive loading per Table 1, not when driving component from EFI.

NOTE 7: When driving the 82284 with EFI, provide minimum EFI HIGH and LOW times as follows:

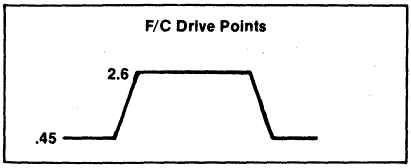
NOTE 8: When using crystal (with recommended capacitive loading per Table 2) appropriate for speed of 80286, CLK output HIGH and LOW times guaranteed to meet 80286 requirements.



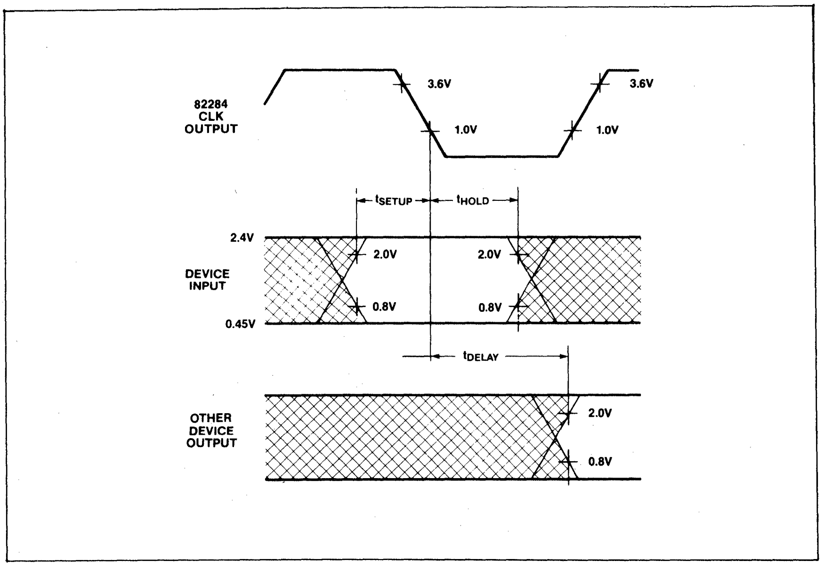
NOTE 9:



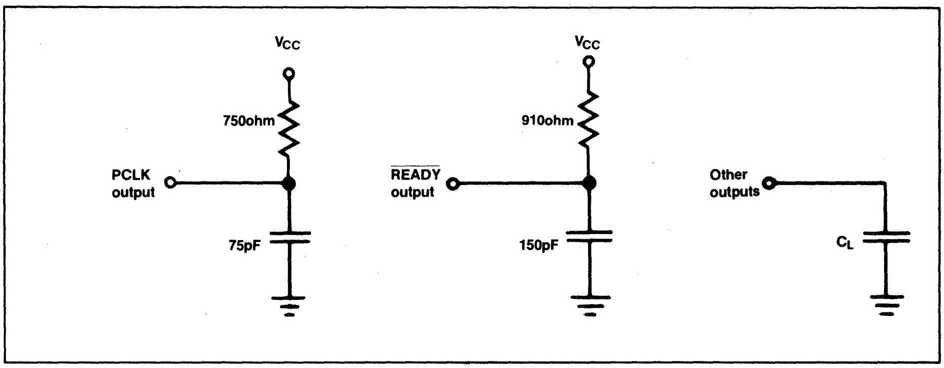
NOTE 10:



NOTE 11:



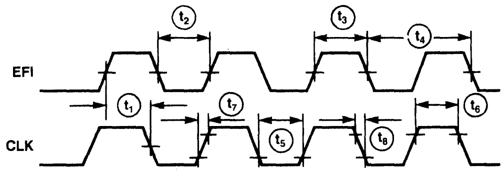
NOTE 12: AC Setup, Hold and Delay Time Measurement—General



NOTE 13: AC Test Loading on Outputs

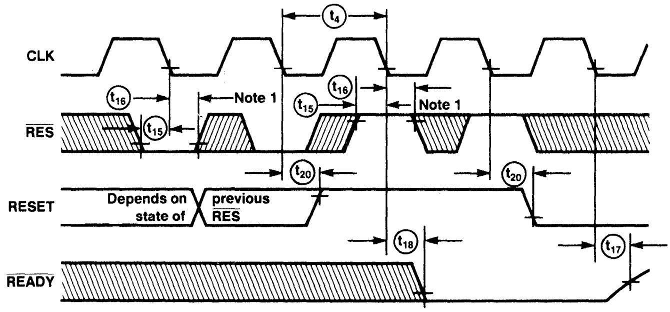
Waveforms

CLK as a Function of EFI (82284-6 only)



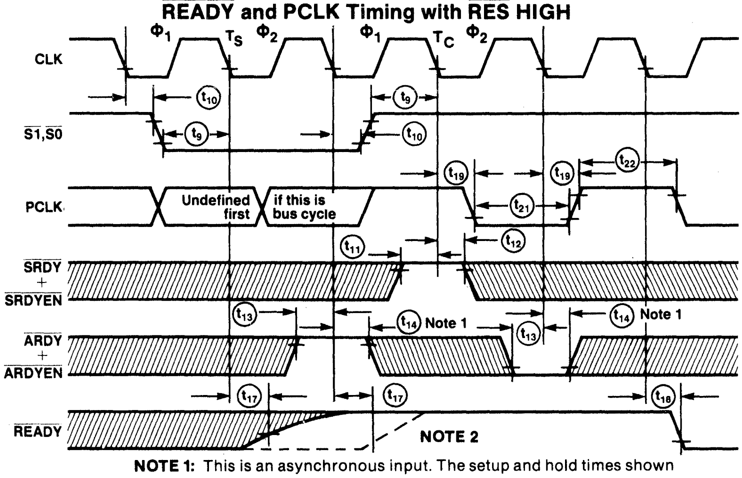
NOTE: The EFI input LOW and HIGH times as shown are required to guarentee the CLK LOW and HIGH times shown.

RESET and $\overline{READY}$ Timing as a Function of $\overline{RES}$ with $\overline{S1}$, $\overline{S0}$, $\overline{ARDY}$ to $\overline{ARDYEN}$, and $\overline{SRDY}$ + $\overline{SRDYEN}$ HIGH



NOTE 1: This is an asynchronous input. The setup and hold times shown are required to guarantee the response shown.

$\overline{READY}$ and PCLK Timing with $\overline{RES}$ HIGH



NOTE 1: This is an asynchronous input. The setup and hold times shown are required to guarantee the response shown.
NOTE 2: If $\overline{SRDY}$ + $\overline{SRDYEN}$ or $\overline{ARDY}$ + $\overline{ARDYEN}$ are active before and/or during the first bus cycle after RESET, $\overline{READY}$ may not be deasserted until after the falling edge of Φ2 of T_S.

210453-004

82288 BUS CONTROLLER FOR iAPX 286 PROCESSORS

(82288-10, 82288-8, 82288-6)

- **Provides Commands and Control for Local and System Bus**
- **Offers Wide Flexibility in System Configurations**
- **Flexible Command Timing**
- **Optional MULTIBUS® Compatible Timing**
- **Control Drivers with 16 ma I_{OL} and 3-State Command Drivers with 32 ma I_{OL}**
- **Single +5V Supply**
- **Available in 20 pin Cerdip Package**
  (See Packaging Spec, Order #231369)

The Intel 82288 Bus Controller is a 20-pin HMOS component for use in iAPX 286 microsystems. The bus controller provides command and control outputs with flexible timing options. Separate command outputs are used for memory and I/O devices. The data bus is controlled with separate data enable and direction control signals.

Two modes of operation are possible via a strapping option: MULTIBUS compatible bus cycles, and high speed bus cycles.

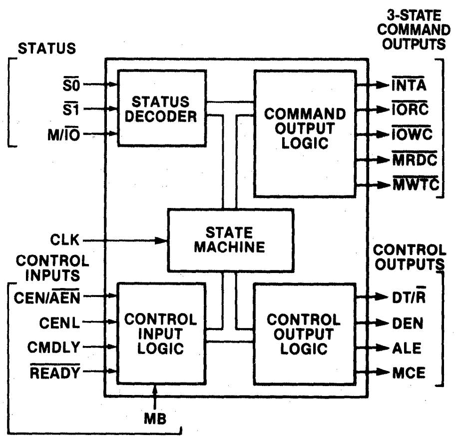


Figure 1. 82288 Block Diagram

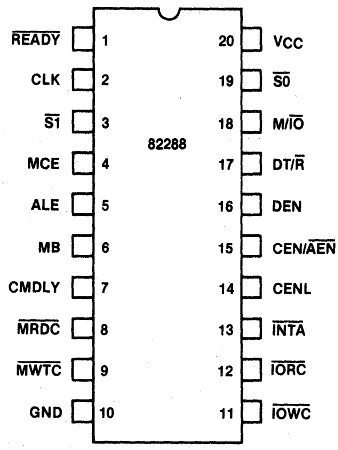


Figure 2. 82288 Pin Configuration

*MULTIBUS is a patented bus of Intel.

January 1986
ORDER NUMBER: 210471-005

Table 1. Pin Description

The following pin function descriptions are for the 82288 bus controller.

Symbol	Type	Name and Function
CLK	I	**System Clock** provides the basic timing control for the 82288 in an iAPX 286 microsystem. Its frequency is twice the internal processor clock frequency. The falling edge of this input signal establishes when inputs are sampled and command and control outputs change.
$\overline{S0}$, $\overline{S1}$	I	**Bus Cycle Status** starts a bus cycle and, along with $M/\overline{IO}$, defines the type of bus cycle. These inputs are active LOW. A bus cycle is started when either $\overline{S1}$ or $\overline{S0}$ is sampled LOW at the falling edge of CLK. Setup and hold times must be met for proper operation.

iAPX 286 Bus Cycle Status Definition

$M/\overline{IO}$	$\overline{S1}$	$\overline{S0}$	Type of Bus Cycle
0	0	0	Interrupt acknowledge
0	0	1	I/O Read
0	1	0	I/O Write
0	1	1	None; idle
1	0	0	Halt or shutdown
1	0	1	Memory read
1	1	0	Memory write
1	1	1	None; idle

Symbol	Type	Name and Function
$M/\overline{IO}$	I	**Memory or I/O Select** determines whether the current bus cycle is in the memory space or I/O space. When LOW, the current bus cycle is in the I/O space. Setup and hold times must be met for proper operation.
MB	I	**MULTIBUS Mode Select** determines timing of the command and control outputs. When HIGH, the bus controller operates with MULTIBUS compatible timings. When LOW, the bus controller optimizes the command and control output timing for short bus cycles. The function of the $CEN/\overline{AEN}$ input pin is selected by this signal. This input is typically a strapping option and not dynamically changed.
CENL	I	**Command Enable Latched** is a bus controller select signal which enables the bus controller to respond to the current bus cycle being initiated. CENL is an active HIGH input latched internally at the end of each T_S cycle. CENL is used to select the appropriate bus controller for each bus cycle in a system where the CPU has more than one bus it can use. This input may be connected to V_{CC} to select this 82288 for all transfers. No control inputs affect CENL. Setup and hold times must be met for proper operation.
CMDLY	I	**Command Delay** allows delaying the start of a command. CMDLY is an active HIGH input. If sampled HIGH, the command output is not activiated and CMDLY is again sampled at the next CLK cycle. When sampled LOW the selected command is enabled. If $\overline{READY}$ is detected LOW before the command output is activated, the 82288 will terminate the bus cycle, even if no command was issued. Setup and hold times must be satisfied for proper operation. This input may be connected to GND if no delays are required before starting a command. This input has no effect on 82288 control outputs.
$\overline{READY}$	I	**READY** indicates the end of the current bus cycle. $\overline{READY}$ is an active LOW input. MULTIBUS mode requires at least one wait state to allow the command outputs to become active. $\overline{READY}$ must be LOW during reset, to force the 82288 into the idle state. Setup and hold times must be met for proper operation. The 82284 drives $\overline{READY}$ LOW during RESET.

210471-005

Table 2. Command and Control Outputs for Each Type of Bus Cycle

Type of Bus Cycle	M/$\overline{IO}$	$\overline{S1}$	$\overline{S0}$	Command Activated	DT/$\overline{R}$ State	ALE, DEN Issued?	MCE Issued?
Interrupt Acknowledge	0	0	0	$\overline{INTA}$	LOW	YES	YES
I/O Read	0	0	1	$\overline{IORC}$	LOW	YES	NO
I/O Write	0	1	0	$\overline{IOWC}$	HIGH	YES	NO
None; idle	0	1	1	None	HIGH	NO	NO
Halt/Shutdown	1	0	0	None	HIGH	NO	NO
Memory Read	1	0	1	$\overline{MRDC}$	LOW	YES	NO
Memory Write	1	1	0	$\overline{MWTC}$	HIGH	YES	NO
None; idle	1	1	1	None	HIGH	NO	NO

Operating Modes

Two types of buses are supported by the 82288: MULTIBUS and non-MULTIBUS. When the MB input is strapped HIGH, MULTIBUS timing is used. In MULTIBUS mode, the 82288 delays command and data activation to meet IEEE-796 requirements on address to command active and write data to command active setup timing. MULTIBUS mode requires at least one wait state in the bus cycle since the command outputs are delayed. The non-MULTIBUS mode does not delay any outputs and does not require wait states. The MB input affects the timing of the command and DEN outputs.

Command and Control Outputs

The type of bus cycle performed by the local bus master is encoded in the M/$\overline{IO}$, $\overline{S1}$, and $\overline{S0}$ inputs. Different command and control outputs are activated depending on the type of bus cycle. Table 2 indicates the cycle decode done by the 82288 and the effect on command, DT/$\overline{R}$, ALE, DEN, and MCE outputs.

Bus cycles come in three forms: read, write, and halt. Read bus cycles include memory read, I/O read, and interrupt acknowledge. The timing of the associated read command outputs ($\overline{MRDC}$, $\overline{IORC}$, and $\overline{INTA}$), control outputs (ALE, DEN, DT/$\overline{R}$) and control inputs (CEN/$\overline{AEN}$, CENL, CMDLY, MB, and $\overline{READY}$) are identical for all read bus cycles. Read cycles differ only in which command output is activated. The MCE control output is only asserted during interrupt acknowledge cycles.

Write bus cycles activate different control and command outputs with different timing than read bus cycles. Memory write and I/O write are write bus cycles whose timing for command outputs ($\overline{MWTC}$ and $\overline{IOWC}$), control outputs (ALE, DEN, DT/$\overline{R}$) and control inputs (CEN/$\overline{AEN}$, CENL, CMDLY, MB, and $\overline{READY}$) are identical. They differ only in which command output is activated.

Halt bus cycles are different because no command or control output is activated. All control inputs are ignored until the next bus cycle is started via $\overline{S1}$ and $\overline{S0}$.

Table 1. Pin Description (Cont.)

Symbol	Type	Name and Function
CEN/$\overline{\text{AEN}}$	I	**Command Enable/Address Enable** controls the command and DEN outputs of the bus controller. CEN/$\overline{\text{AEN}}$ inputs may be asynchronous to CLK. Setup and hold times are given to assure a guaranteed response to synchronous inputs. This input may be connected to V_{CC} or GND. When MB is HIGH this pin has the $\overline{\text{AEN}}$ function. $\overline{\text{AEN}}$ is an active LOW input which indicates that the CPU has been granted use of a shared bus and the bus controller command outputs may exit 3-state OFF and become inactive (HIGH). $\overline{\text{AEN}}$ HIGH indicates that the CPU does not have control of the shared bus and forces the command outputs into 3-state OFF and DEN inactive (LOW). $\overline{\text{AEN}}$ would normally be controlled by an 82289 bus arbiter which activates $\overline{\text{AEN}}$ when that arbiter owns the bus to which the bus controller is attached. When MB is LOW this pin has the CEN function. CEN is an unlatched active HIGH input which allows the bus controller to activate its command and DEN outputs. With MB LOW, CEN LOW forces the command and DEN outputs inactive but does not tristate them.
ALE	O	**Address Latch Enable** controls the address latches used to hold an address stable during a bus cycle. This control output is active HIGH. ALE will not be issued for the halt bus cycle and is not affected by any of the control inputs.
MCE	O	**Master Cascade Enable** signals that a cascade address from a master 8259A interrupt controller may be placed onto the CPU address bus for latching by the address latches under ALE control. The CPU's address bus may then be used to broadcast the cascade address to slave interrupt controllers so only one of them will respond to the interrupt acknowledge cycle. This control output is active HIGH. MCE is only active during interrupt acknowledge cycles and is not affected by any control input. Using MCE to enable cascade address drivers requires latches which save the cascade address on the falling edge of ALE.
DEN	O	**Data Enable** controls when data transceivers connected to the local data bus should be enabled. DEN is an active HIGH control output. DEN is delayed for write cycles in the MULTIBUS mode.
DT/$\overline{\text{R}}$	O	**Data Transmit/Receive** establishes the direction of data flow to or from the local data bus. When HIGH, this control output indicates that a write bus cycle is being performed. A LOW indicates a read bus cycle. DEN is always inactive when DT/$\overline{\text{R}}$ changes states. This output is HIGH when no bus cycle is active. DT/$\overline{\text{R}}$ is not affected by any of the control inputs.
$\overline{\text{IOWC}}$	O	**I/O Write Command** instructs an I/O device to read the data on the data bus. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. $\overline{\text{READY}}$ controls when it becomes inactive.
$\overline{\text{IORC}}$	O	**I/O Read Command** instructs an I/O device to place data onto the data bus. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. $\overline{\text{READY}}$ controls when it becomes inactive.
$\overline{\text{MWTC}}$	O	**Memory Write Command** instructs a memory device to read the data on the data bus. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. $\overline{\text{READY}}$ controls when it becomes inactive.
$\overline{\text{MRDC}}$	O	**Memory Read Command** instructs the memory device to place data onto the data bus. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. $\overline{\text{READY}}$ controls when it becomes inactive.
$\overline{\text{INTA}}$	O	**Interrupt Acknowledge** tells an interrupting device that its interrupt request is being acknowledged. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. $\overline{\text{READY}}$ controls when it becomes inactive.
V_{CC}		**System Power:** +5V power supply
GND		**System Ground:** 0 volts

FUNCTIONAL DESCRIPTION

Introduction

The 82288 bus controller is used in iAPX 286 systems to provide address latch control, data transceiver control, and standard level-type command outputs. The command outputs are timed and have sufficient drive capabilities for large TTL buses and meet all IEEE-796 requirements for MULTIBUS. A special MULTIBUS mode is provided to statisfy all address/data setup and hold time requirements. Command timing may be tailored to special needs via a CMDLY input to determine the start of a command and $\overline{READY}$ to determine the end of a command.

Connection to multiple buses are supported with a latched enable input (CENL). An address decoder can determine which, if any, bus controller should be enabled for the bus cycle. This input is latched to allow an address decoder to take full advantage of the pipelined timing on the iAPX 286 local bus.

Buses shared by several bus controllers are supported. An $\overline{AEN}$ input prevents the bus controller from driving the shared bus command and data signals except when enabled by an external bus arbiter such as the 82289.

Separate DEN and DT/$\overline{R}$ outputs control the data transceivers for all buses. Bus contention is eliminated by disabling DEN before changing DT/$\overline{R}$. The DEN timing allows sufficient time for tristate bus drivers to enter 3-state OFF before enabling other drivers onto the same bus.

The term CPU refers to any iAPX 286 processor or iAPX 286 support component which may become an iAPX 286 local bus master and thereby drive the 82288 status inputs.

Processor Cycle Definition

Any CPU which drives the local bus uses an internal clock which is one half the frequency of the system clock (CLK) (see Figure 3). Knowledge of the phase of the local bus master internal clock is required for proper operation of the iAPX 286 local bus. The local bus master informs the bus controller of its internal clock phase when it asserts the status signals. Status signals are always asserted beginning in Phase 1 of the local bus master's internal clock.

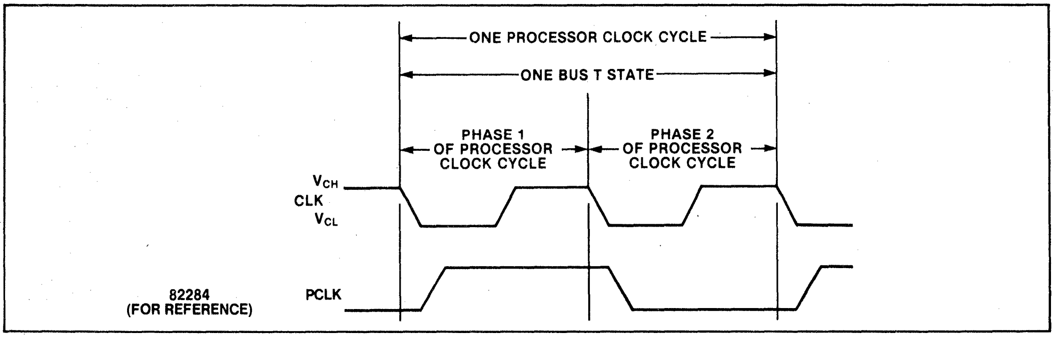


Figure 3. CLK Relationship to the Processor Clock and Bus T-States

Bus State Definition

The 82288 bus controller has three bus states (see Figure 4): Idle (T_I) Status (T_S) and Command (T_C). Each bus state is two CLK cycles long. Bus state phases correspond to the internal CPU processor clock phases.

The T_I bus state occurs when no bus cycle is currently active on the iAPX 286 local bus. This state may be repeated indefinitely. When control of the local bus is being passed between masters, the bus remains in the T_I state.

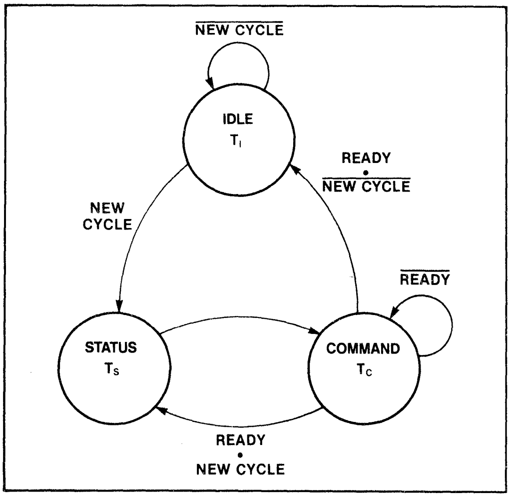


Figure 4. 82288 Bus States

Bus Cycle Definition

The $\overline{S1}$ and $\overline{S0}$ inputs signal the start of a bus cycle. When either input becomes LOW, a bus cycle is started. The T_S bus state is defined to be the two CLK cycles during which either $\overline{S1}$ or $\overline{S0}$ are active (see Figure 5). These inputs are sampled by the 82288 at every falling edge of CLK. When either $\overline{S1}$ or $\overline{S0}$ are sampled LOW, the next CLK cycle is considered the second phase of the internal CPU clock cycle.

The local bus enters the T_C bus state after the T_S state. The shortest bus cycle may have one T_S state and one T_C state. Longer bus cycles are formed by repeating T_C states. A repeated T_C bus state is called a wait state.

The $\overline{\text{READY}}$ input determines whether the current T_C bus state is to be repeated. The $\overline{\text{READY}}$ input has the same timing and effect for all bus cycles. $\overline{\text{READY}}$ is sampled at the end of each T_C bus state to see if it is active. If sampled HIGH, the T_C bus state is repeated. This is called inserting a wait state. The control and command outputs do not change during wait states.

When $\overline{\text{READY}}$ is sampled LOW, the current bus cycle is terminated. Note that the bus controller may enter the T_S bus state directly from T_C if the status lines are sampled active at the next falling edge of CLK.

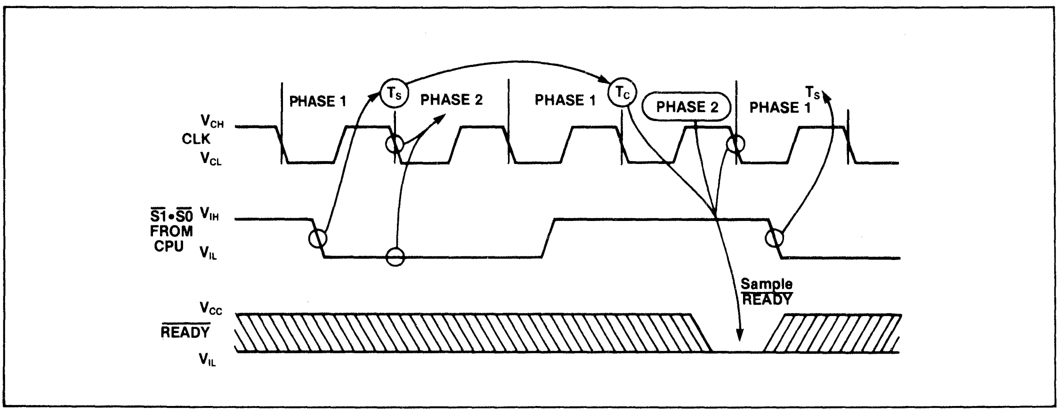


Figure 5. Bus Cycle Definition

210471-005

Figures 6-10 show the basic command and control output timing for read and write bus cycles. Halt bus cycles are not shown since they activate no outputs. The basic idle-read-idle and idle-write-idle bus cycles are shown. The signal label CMD represents the appropriate command output for the bus cycle. For Figures 6-10, the CMDLY input is connected to GND and CENL to V_{CC}. The effects of CENL and CMDLY are described later in the section on control inputs.

Figures 6, 7 and 8 show non-MULTIBUS cycles. MB is connected to GND while CEN is connected to V_{CC}. Figure 6 shows a read cycle with no wait states while Figure 7 shows a write cycle with one wait state. The $\overline{READY}$ input is shown to illustrate how wait states are added.

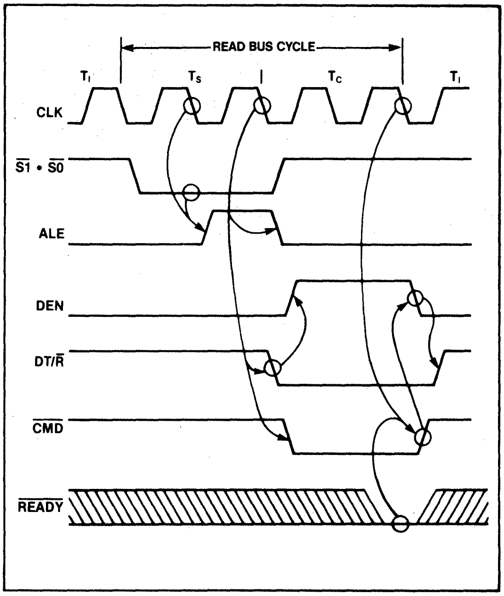


Figure 6. Idle-Read-Idle Bus Cycles with MB = 0

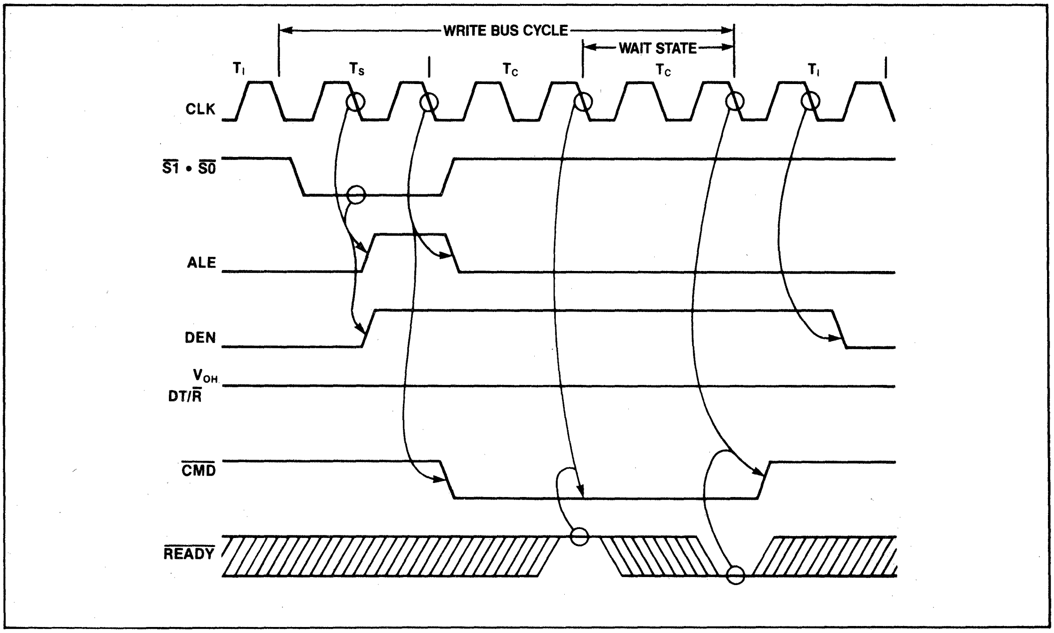


Figure 7. Idle-Write-Idle Bus Cycles with MB = 0

Bus cycles can occur back to back with no T_I bus states between T_C and T_S. Back to back cycles do not affect the timing of the command and control outputs. Command and control outputs always reach the states shown for the same clock edge (within T_S, T_C, or following bus state) of a bus cycle.

A special case in control timing occurs for back to back write cycles with MB = 0. In this case, DT/$\overline{R}$ and DEN remain HIGH between the bus cycles (see Figure 8). The command and ALE output timing does not change.

Figures 9 and 10 show a MULTIBUS cycle with MB = 1. $\overline{AEN}$ and CMDLY are connected to GND. The effects of CMDLY and $\overline{AEN}$ are described later in the section on control inputs. Figure 9 shows a read cycle with one wait state and Figure 10 shows a write cycle with two wait states. The second wait state of the write cycle is shown only for example purposes and is not required. The $\overline{READY}$ input is shown to illustrate how wait states are added.

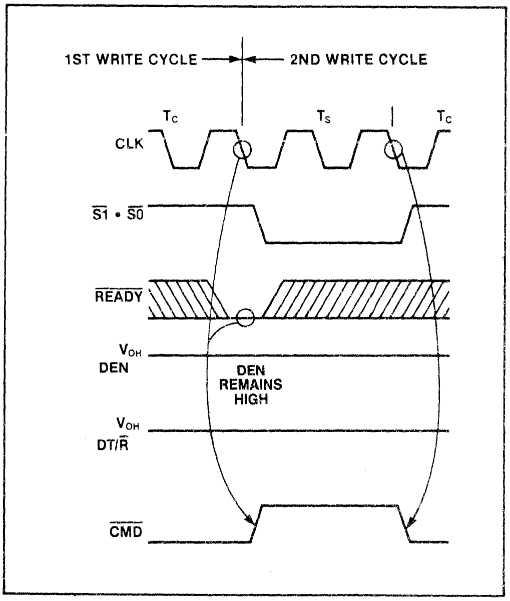


Figure 8. Write-Write Bus Cycles with MB = 0

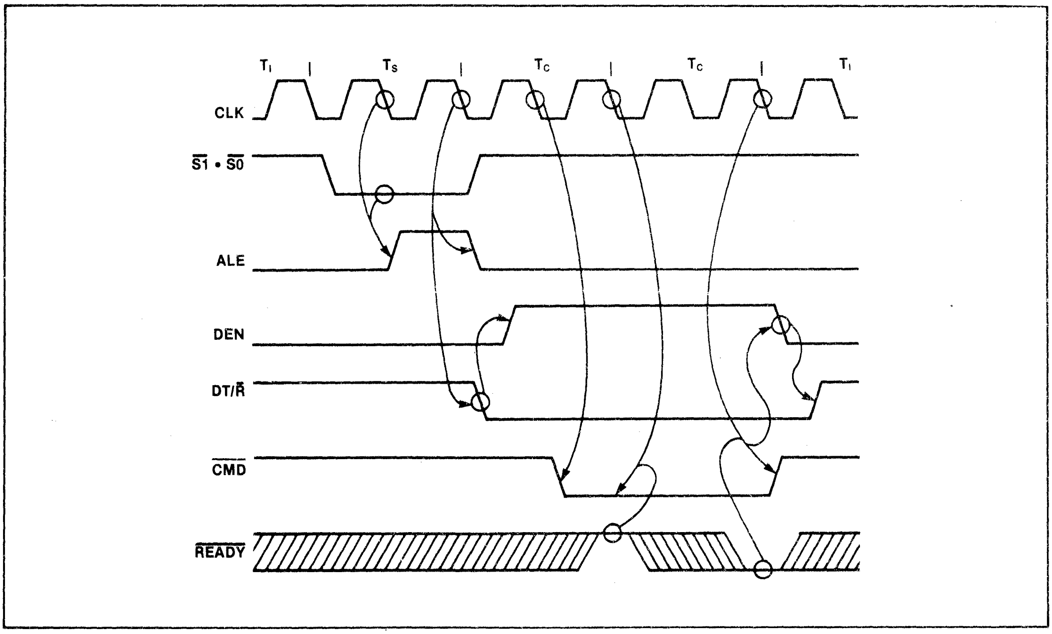


Figure 9. Idle-Read-Idle Bus Cycles with MB = 1

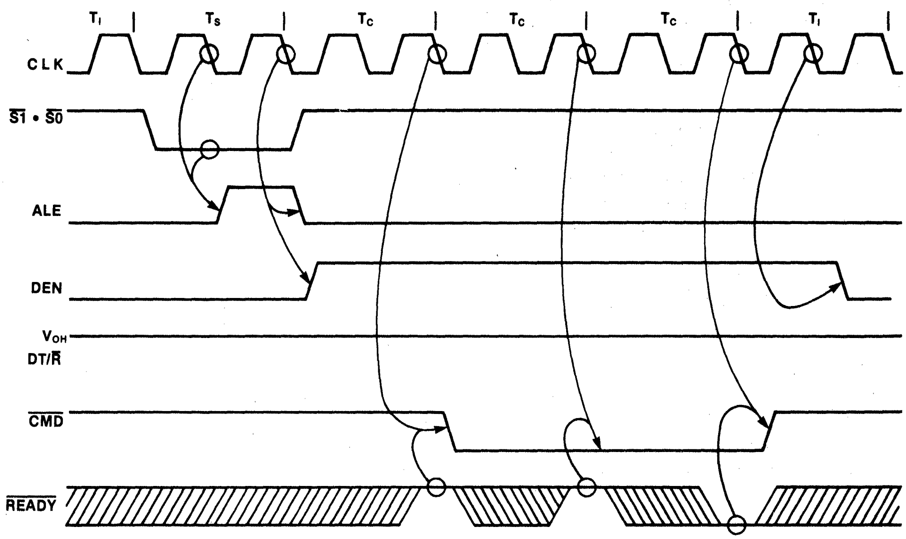


Figure 10. Idle-Write-Idle Bus Cycles with MB = 1

The MB control input affects the timing of the command and DEN outputs. These outputs are automatically delayed in MULTIBUS mode to satisfy three requirements:

1) 50 ns minimum setup time for valid address before any command output becomes active.

2) 50 ns minimum setup time for valid write data before any write command output becomes active.

3) 65 ns maximum time from when any read command becomes inactive until the slave's read data drivers reach 3-state OFF.

Three signal transitions are delayed by MB = 1 as compared to MB = 0:

1) The HIGH to LOW transition of the read command outputs ($\overline{IORC}$, $\overline{MRDC}$, and $\overline{INTA}$) are delayed one CLK cycle.

2) The HIGH to LOW transition of the write command outputs ($\overline{IOWC}$ and $\overline{MWTC}$) are delayed two CLK cycles.

3) The LOW to HIGH transition of DEN for write cycles is delayed one CLK cycle.

Back to back bus cycles with MB = 1 do not change the timing of any of the command or control outputs. DEN always becomes inactive between bus cycles with MB = 1.

Except for a halt or shutdown bus cycle, ALE will be issued during the second half of T_S for any bus cycle. ALE becomes inactive at the end of the T_S to allow latching the address to keep it stable during the entire bus cycle. The address outputs may change during Phase 2 of any T_C bus state. ALE is not affected by any control input.

Figure 11 shows how MCE is timed during interrupt acknowledge (INTA) bus cycles. MCE is one CLK cycle longer than ALE to hold the cascade address from a master 8259A valid after the falling edge of ALE. With the exception of the MCE control output, an INTA bus cycle is identical in timing to a read bus cycle. MCE is not affected by any control input.

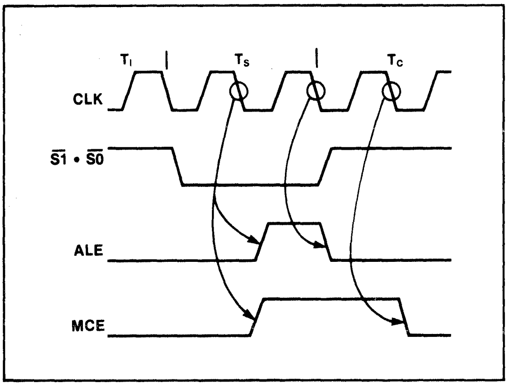


Figure 11. MCE Operation for an INTA Bus Cycle

Control Inputs

The control inputs can alter the basic timing of command outputs, allow interfacing to multiple buses, and share a bus between different masters. For many iAPX 286 systems, each CPU will have more than one bus which may be used to perform a bus cycle. Normally, a CPU will only have one bus controller active for each bus cycle. Some buses may be shared by more than one CPU (i.e. MULTIBUS) requiring only one of them use the bus at a time.

Systems with multiple and shared buses use two control input signals of the 82288 bus controller, CENL and $\overline{\text{AEN}}$ (see Figure 12). CENL enables the bus controller to control the current bus cycle. The $\overline{\text{AEN}}$ input prevents a bus controller from driving its command outputs. $\overline{\text{AEN}}$ HIGH means that another bus controller may be driving the shared bus.

In Figure 12, two buses are shown: a local bus and a MULTIBUS. Only one bus is used for each CPU bus cycle. The CENL inputs of the bus controller select which bus controller is to perform the bus cycle. An address decoder determines which bus to use for each bus cycle. The 82288 connected to the shared MULTIBUS must be selected by CENL and be given access to the MULTIBUS by $\overline{\text{AEN}}$ before it will begin a MULTIBUS operation.

CENL must be sampled HIGH at the end of the T_S bus state (see waveforms) to enable the bus controller to activate its command and control outputs. If sampled LOW the commands and DEN will not go active and DT/$\overline{\text{R}}$ will remain HIGH. The bus controller will ignore the CMDLY, CEN, and READY inputs until another bus cycle is started via $\overline{\text{S1}}$ and $\overline{\text{S0}}$. Since an address decoder is commonly used to identify which bus is required for each bus cycle, CENL is latched to avoid the need for latching its input.

The CENL input can affect the DEN control output. When MB = 0, DEN normally becomes active during Phase 2 of T_S in write bus cycles. This transition occurs before CENL is sampled. If CENL is sampled LOW, the DEN output will be forced LOW during T_C as shown in the timing waveforms.

When MB = 1, CEN/$\overline{\text{AEN}}$ becomes $\overline{\text{AEN}}$. $\overline{\text{AEN}}$ controls when the bus controller command outputs enter and exit 3-state OFF. $\overline{\text{AEN}}$ is intended to be driven by a bus arbiter, like the 82289, which assures only one bus controller is driving the shared bus at any time. When $\overline{\text{AEN}}$ makes a LOW to HIGH transition, the command outputs immediately enter 3-state OFF and DEN is forced inactive. An inactive DEN should force the local data transceivers connected to the shared data bus into 3-state OFF (see Figure 12). The LOW to HIGH transition of $\overline{\text{AEN}}$ should only occur during T_I or T_S bus states.

The HIGH to LOW transition of $\overline{\text{AEN}}$ signals that the bus controller may now drive the shared bus command signals. Since a bus cycle may be active or be in the process of starting, $\overline{\text{AEN}}$ can become active during any T-state. $\overline{\text{AEN}}$ LOW immediately allows DEN to go to the appropriate state. Three CLK edges later, the command outputs will go active (see timing waveforms). The MULTIBUS requires this delay for the address and data to be valid on the bus before the command become active.

When MB = 0, CEN/$\overline{\text{AEN}}$ becomes CEN. CEN is an asynchronous input which immediately affects the command and DEN outputs. When CEN makes a HIGH to LOW transition, the commands and DEN are immediately forced inactive. When CEN makes a LOW to HIGH transition, the commands and DEN outputs immediately go to the appropriate state (see timing waveforms). $\overline{\text{READY}}$ must still become active to terminate a bus cycle if CEN remains LOW for a selected bus controller (CENL was latched HIGH).

Some memory or I/O systems may require more address or write data setup time to command active than provided by the basic command output timing. To provide flexible command timing, the CMDLY input can delay the activation of command outputs. The CMDLY input must be sampled LOW to activate the command outputs. CMDLY does not affect the control outputs ALE, MCE, DEN, and DT/$\overline{\text{R}}$.

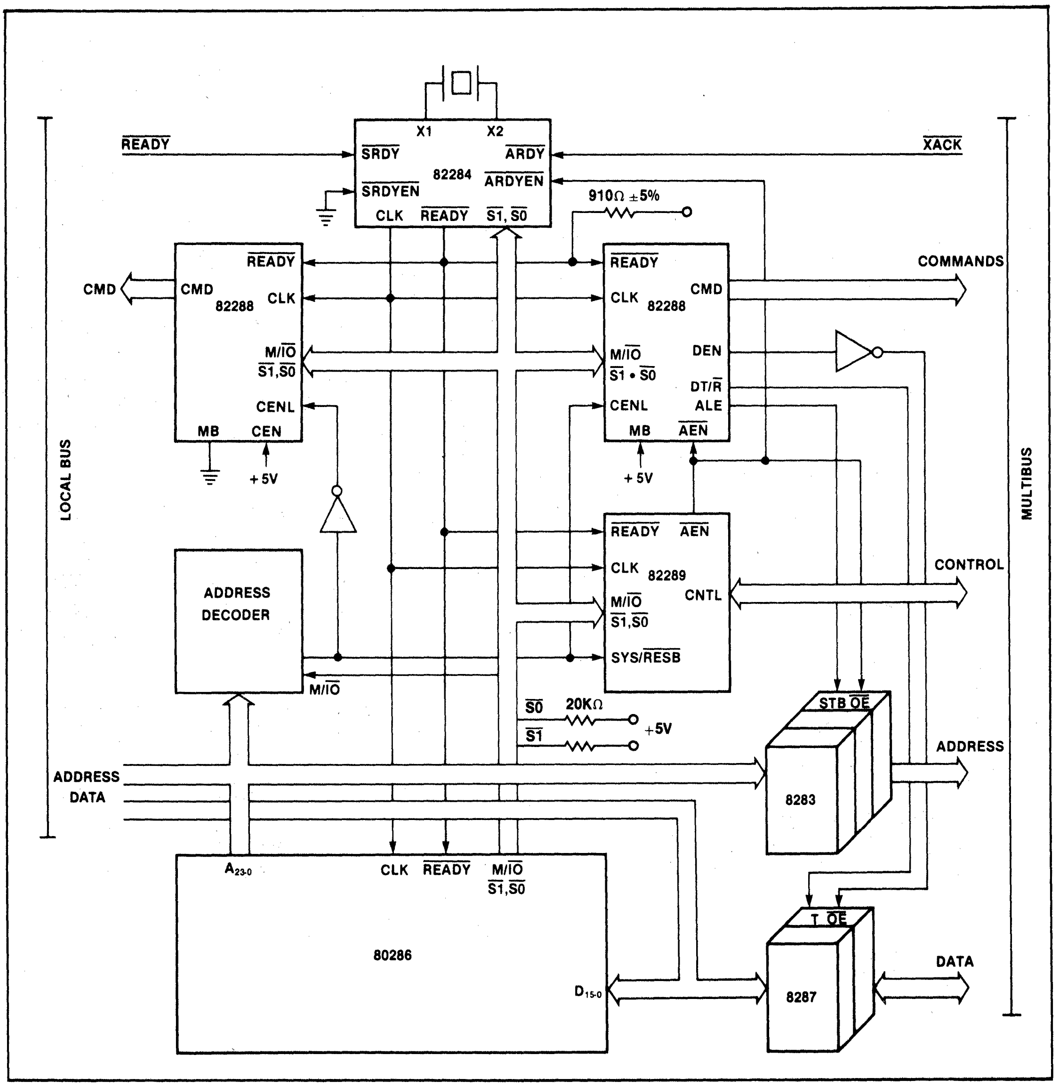


Figure 12. System Use of $\overline{\text{AEN}}$ and CENL

CMDLY is first sampled on the falling edge of the CLK ending T_S. If sampled HIGH, the command output is not activated, and CMDLY is again sampled on the next falling edge of CLK. Once sampled LOW, the proper command output becomes active immediately if MB=0. If MB=1, the proper command goes active no earlier than shown in Figures 9 and 10.

$\overline{\text{READY}}$ can terminate a bus cycle before CMDLY allows a command to be issued. In this case no commands are issued and the bus controller will deactivate DEN and DT/$\overline{\text{R}}$ in the same manner as if a command had been issued.

Waveforms Discussion

The waveforms show the timing relationships of inputs and outputs and do not show all possible transitions of all signals in all modes. Instead, all signal timing relationships are shown via the general cases. Special cases are shown when needed. The waveforms provide some functional descriptions of the 82288; however, most functional descriptions are provided in Figures 5 through 11.

To find the timing specification for a signal transition in a particular mode, first look for a special case in the waveforms. If no special case applies, then use a timing specification for the same or related function in another mode.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias. . . . 0°C to 70°C
Storage Temperature. . . . . −65°C to +150°C
Voltage on Any Pin with
Respect to GND. . . . . −0.5V to +7V
Power Dissipation. . . . . 1 Watt

**NOTICE:* *Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (V_{CC} = 5V ±5%, T_A = 0°C to 70°C, or T_{CASE} = 0°C to 85°C)

		6 MHz		8 MHz			
Symbol	**Parameter**	**-6 Min.**	**-6 Max.**	**-8 Min.**	**-8 Max.**	**Units**	**Test Conditions**
V_{IL}	Input LOW Voltage	−.5	.8	−.5	.8	V	
V_{IH}	Input HIGH Voltage	2.0	V_{CC} + .5	2.0	V_{CC} + .5	V	
V_{ILC}	CLK Input LOW Voltage	−.5	.6	−.5	.6	V	
V_{IHC}	CLK Input HIGH Voltage	3.8	V_{CC} +.5	3.8	V_{CC} +.5	V	
V_{OL}	Output LOW Voltage						
	Command Outputs		.45		.45	V	I_{OL} = 32mA Note 1
	Control Outputs		.45		.45	V	I_{OL} = 16mA Note 2
V_{OH}	Output HIGH Voltage						
	Command Outputs	2.4		2.4		V	I_{OH} = −5mA Note 1
	Control Outputs	2.4		2.4		V	I_{OH} = −1mA Note 2
I_F	Input Current ($\overline{S0}$ and $\overline{S1}$ inputs)		−.5		−.5	mA	Vf = .45V
I_{IL}	Input Leakage current (all other inputs)		±10		±10	μA	0V ≤ VI_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current		±10		±10	μA	.45V ≤ V_{OUT} ≤ V_{CC}
I_{CC}	Power Supply Current		120		120	mA	
C_{CLK}	CLK Input Capacitance		12		12	pF	F_C = 1 MHz
C_I	Input Capacitance		10		10	pF	F_C = 1 MHz
C_O	Input/Output Capacitance		20		20	pF	F_C = 1 MHz

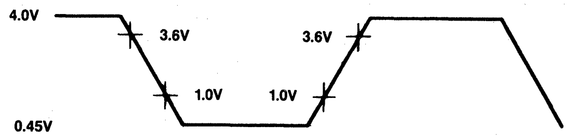
NOTE: 1. Command Outputs are $\overline{INTA}$, $\overline{IORC}$, $\overline{IOWC}$, $\overline{MRDC}$, $\overline{MWRC}$.
2. Control Outputs are DT/$\overline{R}$, DEN, ALE and MCE.

A.C. CHARACTERISTICS

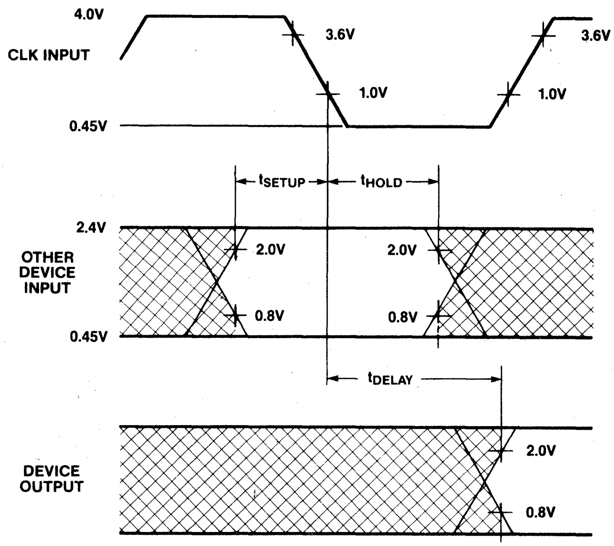
(T_A = 0°C to 70°C, V_{CC} = 5V, ±5%)
AC timings are referenced to 0.8V and 2.0V points of signals as illustrated in data sheet waveforms, unless otherwise noted.

		6 MHz		8 MHz		10 MHz (Preliminary)			
Sym	**Parameter**	**-6 Min.**	**-6 Max.**	**-8 Min.**	**-8 Max.**	**-10 Min.**	**-10 Max.**	**Unit**	**Test Condition**
1	CLK Period	83	250	62	250	50	250	ns	
2	CLK HIGH Time	25	230	20	235	16	238	ns	at 3.6V
3	CLK LOW Time	20	225	15	230	12	234	ns	at 1.0V
4	CLK Rise Time		10		10		8	ns	1.0V to 3.6V
5	CLK Fall Time		10		10		8	ns	3.6V to 1.0V
6	M/$\overline{IO}$ and Status Setup Time	28		22		18		ns	
7	M/$\overline{IO}$ and Status Hold Time	1		1		1		ns	
8	CENL Setup Time	30		20		15		ns	
9	CENL Hold time	1		1		1		ns	
10	$\overline{READY}$ Setup Time	50		38		26		ns	
11	$\overline{READY}$ Hold Time	35		25		25		ns	
12	CMDLY Setup Time	25		20		15		ns	
13	CMDLY Hold Time	1		1		1		ns	
14	$\overline{AEN}$ Setup Time	25		20		15		ns	Note 3
15	$\overline{AEN}$ Hold Time	0		0		0		ns	Note 3
16	ALE, MCE Active Delay from CLK	3	25	3	20	3	16	ns	Note 4
17	ALE, MCE Inactive Delay from CLK		35		25		19	ns	Note 4
18	DEN (Write) Inactive from CENL		35		35		23	ns	Note 4
19	DT/$\overline{R}$ LOW from CLK		40		25		23	ns	Note 4
20	DEN (Read) Active from DT/$\overline{R}$	5	50	5	35	5	21	ns	Note 4
21	DEN (Read) Inactive Dly from CLK	3	40	3	35	3	21	ns	Note 4
22	DT/$\overline{R}$ HIGH from DEN Inactive	5	45	5	35	5	20	ns	Note 4
23	DEN (Write) Active Delay from CLK		35		30		23	ns	Note 4
24	DEN (Write) Inactive Dly from CLK	3	35	3	30	3	19	ns	Note 4
25	DEN Inactive from CEN		40		30		25	ns	Note 4
26	DEN Active from CEN		35		30		24	ns	Note 4
27	DT/$\overline{R}$ HIGH from CLK (when CEN = LOW)		50		35		25	ns	Note 4
28	DEN Active from $\overline{AEN}$		35		30		26	ns	Note 4
29	$\overline{CMD}$ Active Delay from CLK	3	40	3	25	3	21	ns	Note 5
30	$\overline{CMD}$ Inactive Delay from CLK	5	30	5	25	5	20	ns	Note 5
31	$\overline{CMD}$ Inactive from CEN		35		25		25	ns	Note 5
32	$\overline{CMD}$ Inactive from CEN		45		25		25	ns	Note 5
33	$\overline{CMD}$ Inactive Enable from $\overline{AEN}$		40		40		40	ns	Note 5
34	$\overline{CMD}$ Float Delay from $\overline{AEN}$		40		40		40	ns	Note 6
35	MB Setup Time	25		20		20		ns	
36	MB Hold Time	0		0		0		ns	
37	Command Inactive Enable from MB↓		40		40		40	ns	Note 5
38	Command Float Time from MB↑		40		40		40	ns	Note 6
39	DEN Inactive from MB↑		40		30		26	ns	Note 4
40	DEN Active from MB↓		35		30		30	ns	Note 4

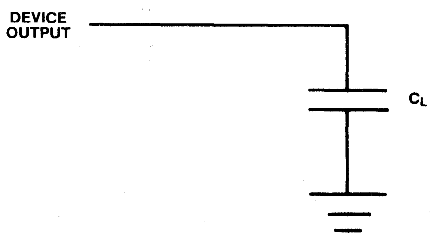
NOTE: 3. $\overline{AEN}$ is an asynchronous input. This specification is for testing purposes only, to assure recognition at a specific CLK edge.
4. Control output load: Cl = 150pF.
5. Command output load: Cl = 300pF.
6. Float condition occurs when output current is less



NOTE 7: AC Drive and Measurement Points — CLK Input



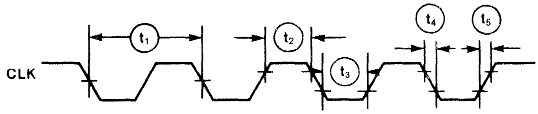
NOTE 8: AC Setup, Hold and Delay Time Measurement — General



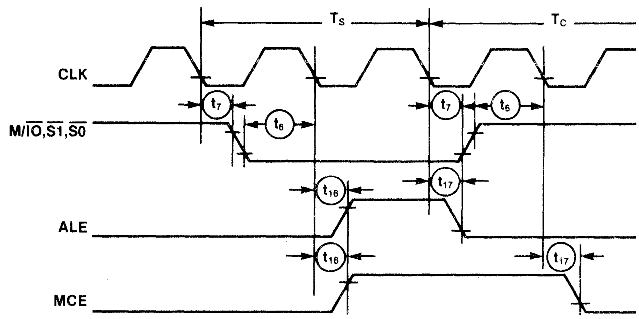
NOTE 9: AC Test Loading on Outputs

WAVEFORMS

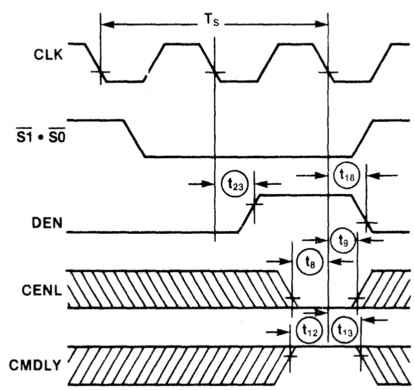
CLK CHARACTERISTICS



STATUS, ALE, MCE, CHARACTERISTICS



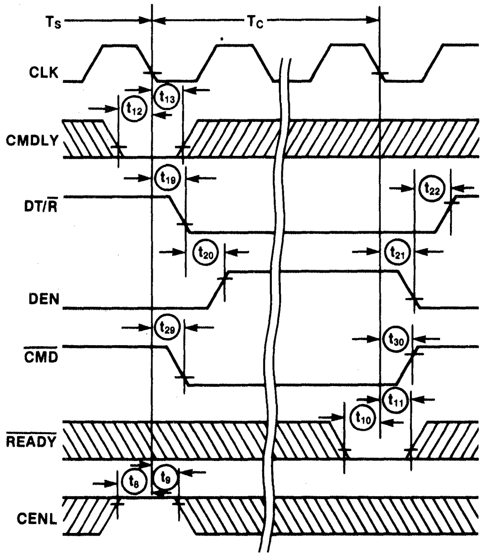
CENL, CMDLY, DEN CHARACTERISTICS WITH MB = 0 AND CEN = 1 DURING WRITE CYCLE



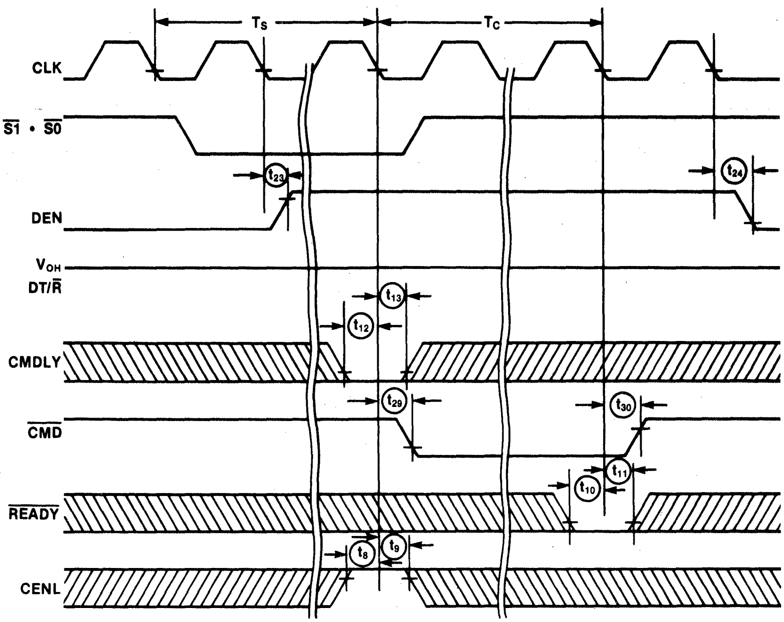
210471-005

WAVEFORMS (Continued)

READ CYCLE CHARACTERISTICS WITH MB = 0 AND CEN = 1



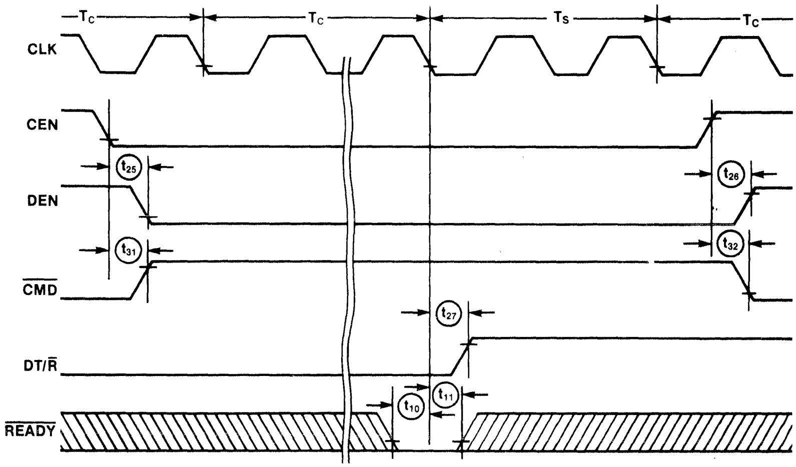
WRITE CYCLE CHARACTERISTICS WITH MB = 0 AND CEN = 1



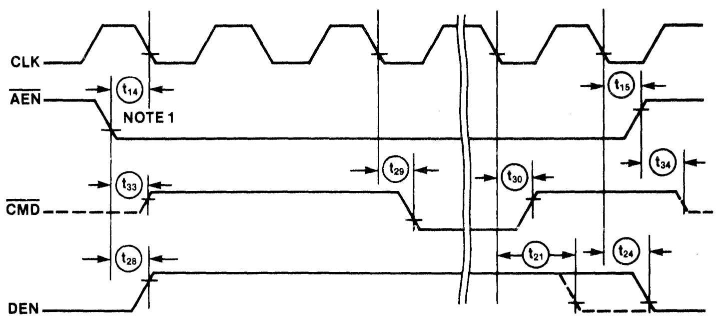
210471-005

WAVEFORMS (Continued)

CEN CHARACTERISTICS WITH MB = 0



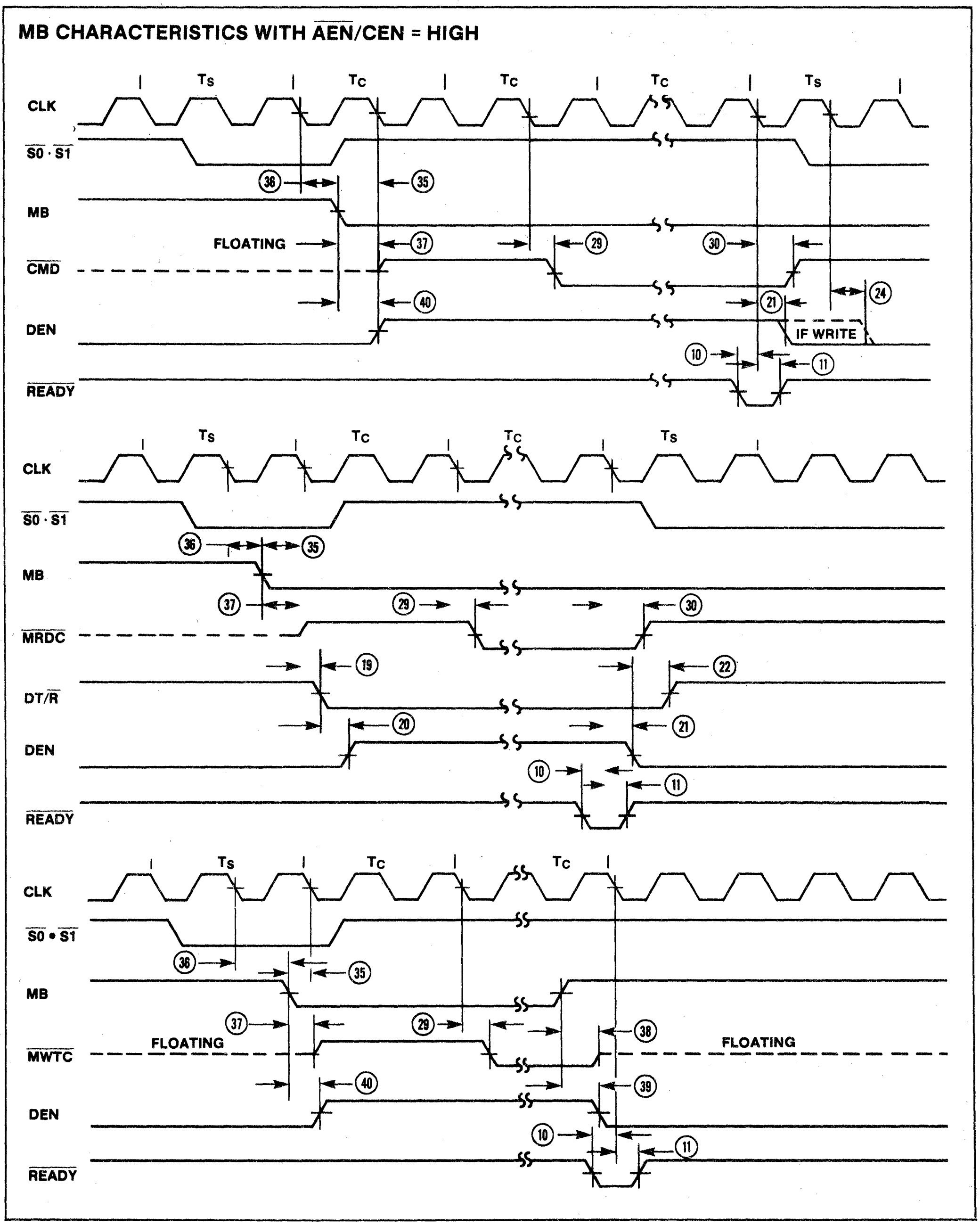
AEN CHARACTERISTICS WITH MB = 1



NOTE 1: AEN is an asynchronous input. AEN setup and hold time is specified to guarantee the response shown in the waveforms.

210471-005

WAVEFORMS (Continued)



NOTE 1: MB is an asynchronous input. MB setup and hold times specified to guarantee the response shown in the waveforms.

NOTE 2: If the setup time, t35, is met two clock cycles will occur before $\overline{CMD}$ becomes active after the falling edge of MB.

intel®

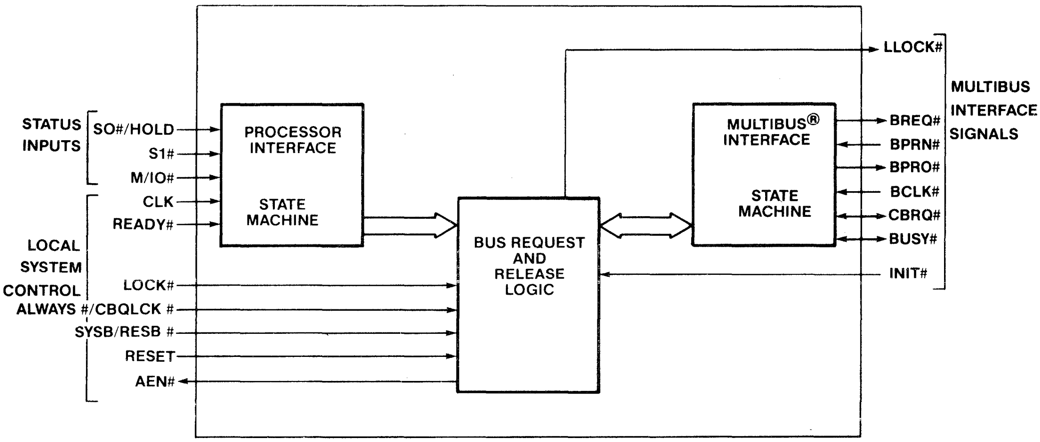
ADVANCE INFORMATION

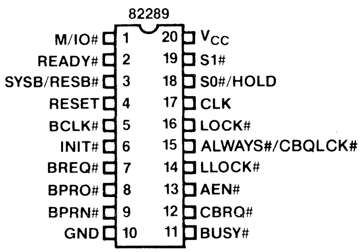
82289 BUS ARBITER FOR iAPX 286 PROCESSOR FAMILY

- **Supports Multi-master System Bus Arbitration Protocol**
- **Synchronizes 80286 Processor with Multi-master Bus**
- **Compatible With Intel Bus Standard MULTIBUS® (IEEE 796 Standard)**
- **Three Modes of Bus Release Operation for Flexible System Configuration**
- **Supports Parallel, Serial, and Rotating Priority Resolving Schemes**
- **Available in EXPRESS - Standard Temperature Range**
- **Available in 20 Pin Plastic Dip and Cerdip Packages (See Packaging Spec Order #231369)**

The Intel 82289 Bus Arbiter is a 5-Volt, 20-pin HMOS III component for use in multiple bus master iAPX 286 systems. The 82289 provides a compact solution to system bus arbitration for the 80286 CPU.

The complete IEEE 796 Standard bus arbitration protocol is supported. Three modes of bus release operation support a number of bus usage models.





# INDICATES FUNCTION IS ACTIVE LOW

Figure 1. 82289 Block Diagram

Figure 2. 82289 Pin Diagram

Table 1. 82289 Pin Definition

<table>
<tr><th>Symbol</th><th>Pin(s)</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>CLK</td><td>17</td><td>I</td><td>SYSTEM CLOCK accepts the CLK signal from the 82284 Clock Generator chip as the timing reference for the bus arbiter and processor interface signals.</td></tr>
<tr><td>S0#/HOLD</td><td>18</td><td>I</td><td>STATUS INPUT S0# or HOLD is either the S0# status signal from 80286 or the HOLD signal from some other bus master. The function of this input is established during the processor reset of the 82289 Bus Arbiter. The 80286 S0# pin meets the setup and hold time requirements of this pin.

The S0# pin function is selected by forcing this input high during the falling edge of processor reset. If the 82289 is used to support an 80286 processor, the S0# output of the processor will be high during reset.

In supporting the 80286 processor, the 82289 decodes the S0# pin together with the other status input pins, S1# and M/IO#, to determine the beginning of a processor bus cycle and initiate bus request and surrender actions.

The HOLD function of the S0#/HOLD pin is selected by holding this input low during the falling edge of processor reset. When supporting a bus master other than 80286, the 82289 monitors the HOLD signal to initiate bus request and surrender actions.</td></tr>
<tr><td>S1#, M/IO#</td><td>19, 1</td><td>I</td><td>STATUS INPUTS are the status input signal pins from the 80286 processor. The arbiter decodes these inputs together with S0#/HOLD input to initiate bus request and surrender actions. A bus cycle is started when either S1# or S0# is sampled LOW at the falling edge of CLK. The 80286 S1# and M/IO# pins meet the setup and hold time requirements of these pins.

80286 Bus Cycle Status Encoding

M/IO# S1# S0#/HOLD Type of Bus Cycle
0 0 0 Interrupt acknowledge
0 0 1 I/O Read
0 1 0 I/O Write
0 1 1 None; bus idle
1 0 0 Halt or shutdown
1 0 1 Memory read
1 1 0 Memory write
1 1 1 None; bus idle

When supporting the HOLD output of another bus master, the S1# and M/IO# pins must be held HIGH during T_S, the Status Cycle, for proper operation.</td></tr>
<tr><td>SYSB/RESB#</td><td>3</td><td>I</td><td>SYSTEM BUS/RESIDENT BUS# is an input signal which determines when the multi-master system bus is required for the current bus cycle. The signal can originate from address mapping circuitry such as a decoder or PROM attached to the processor address and status pins. The arbiter will request or retain control of the multi-master system bus when the SYSB/RESB# pin is sampled HIGH at the end of the T_S bus state.

During an interrupt acknowledge cycle, this input is sampled on every falling edge of CLK starting at the end of the T_S state until either SYSB/RESB# is sampled HIGH or the bus cycle is terminated by the READY# signal. Setup and hold times for this pin must be met for proper operation.</td></tr>
</table>

Table 1. 82289 Pin Definition (continued)

Symbol	Pin(s)	Type	Name and Function
READY#	2	I	**READY#** is an active-LOW signal which indicates the end of the bus cycle. The 80286 halt or shutdown cycle does not require READY# to terminate the bus cycle. Setup and hold times for this pin must be met for proper operation.
LOCK#	16	I	**LOCK #** is a processor-generated signal which when asserted (LOW) prevents the arbiter from surrendering the multi-master system bus to any other bus arbiter, regardless of its priority. LOCK# is sampled by the arbiter at the end of the T_S (status) bus state. Setup and hold times for this pin must be met for proper operation.
ALWAYS#/ CBQLCK#	15	I	**ALWAYS RELEASE# or COMMON BUS REQUEST LOCK#** can be programmed at processor reset to be either the ALWAYS RELEASE (ALWAYS#) strapping option or the COMMON BUS REQUEST LOCK (CBQLCK#) control input. Setup and hold times for this pin must be met for proper programming. When this pin is LOW during the falling edge of processor reset (ALWAYS# option) the arbiter is programmed to surrender the multi-master system bus after each bus transfer cycle. The 82289 will remain in the ALWAYS RELEASE mode until it is reprogrammed during the next processor reset. The bus arbiter is programmed to support the COMMON BUS REQUEST LOCK function by forcing this input pin HIGH during the falling edge of the processor reset. CBQLCK# itself is an active-LOW signal which when active prevents the arbiter from surrendering the multi-master system bus to a common bus request through the CBRQ# input pin.
RESET	4	I	**PROCESSOR RESET** is an active-HIGH input synchronous to the system clock (CLK). RESET is the processor initialization of the arbiter to release the multi-master bus and clear any pending request.
INIT#	6	I	**INITIALIZE#** is an active-low MULTIBUS® signal used to reset all arbiters on the MULTIBUS system. It will cause the release of the multi-master bus, but will not clear the pending bus master request so that the arbiter can again request the multi-master bus. No arbiters have the use of the multi-master bus immediately after initialization. INIT# is an asynchronous signal to CLK.
BCLK#	5	I	**BUS CLOCK#** is the multi-master system bus clock to which the multi-master bus interface signals are synchronized. BCLK# can be asynchronous to CLK.
BREQ#	7	O	**BUS REQUEST#** is an active-LOW output signal used in the parallel and rotating priority resolving schemes. The arbiter activates BREQ# to request the use of the multi-master system bus. The arbiter holds BREQ# active as long as it is requesting or has possession of the multi-master system bus.
CBRQ#	12	I/O (open-drain)	**COMMON BUS REQUEST#** is a MULTIBUS signal that indicates when an arbiter is requesting the MULTIBUS. This pin is an open-drain input/output requiring an external pullup resistor. As an input CBRQ# indicates that another arbiter is requesting the multi-master system bus. The input function of this pin is enabled by the CBQLCK# signal. Setup and hold times for this pin must be met for proper operation. As an output CBRQ# is asserted to indicate that this arbiter is requesting the MULTIBUS. The arbiter pulls CBRQ# low when it issues a BREQ#. The arbiter release CBRQ# when it obtains the MULTIBUS.

Table 1. 82289 Pin Definition (continued)

Symbol	Pin(s)	Type	Name and Function
BPRN#	9	I	**BUS PRIORITY IN#** is an active-low input indicating that this arbiter has the highest priority of any arbiter requesting the system bus. BPRN# HIGH signals the arbiter that a higher priority arbiter is requesting or has possession of the system bus. Setup and hold times for this pin must be met for proper operation.
BPRO#	8	O	**BUS PRIORITY OUT#** is an active-low output signal used in the serial priority resolving scheme. BRPO# is connected to BPRN# of the next lower priority to grant or revoke priority from that arbiter.
BUSY#	11	I/O (open-drain)	**BUSY#** is a MULTIBUS signal which is asserted when the system bus is in use. BUSY# is an open drain input/output requiring an external pullup resistor. As an input BUSY# asserted indicates when the MULTIBUS is in use. Setup and hold times must be met for proper operation. As an output BUSY# is asserted to signal when this arbiter has taken control of the MULTIBUS.
AEN#	13	O	**ADDRESS ENABLE#** is the output of the arbiter which goes directly to the processor's address latches, the 82288 Bus Controller and the 82284 Clock Generator. AEN# asserted causes the bus controller and address latches to enable their output drivers. AEN# also drives the clock generator ARDYEN# input to enable its asynchronous ready input (ARDY#). AEN# can also be used as an active-LOW Hold Acknowledge to a bus master other than 80286. It signals to the bus master that control of the system bus has been relinquished when AEN# is inactive (HIGH). Note that AEN# goes active relative to BCLK# and goes inactive relative to CLK.
LLOCK#	14	O	**LEVEL LOCK#** is an active-low output signal decoded from processor LOCK# signal. LLOCK# can be used as MULTIBUS LOCK# when buffered with a tri-state buffer enabled by the AEN# signal. LLOCK1# will be cleared by RESET but not by INIT#.
V_{CC}	20	I	+5 volts supply voltage
GND	10	I	Ground

FUNCTIONAL DESCRIPTION

The 82289 Bus Arbiter in conjunction with the 82288 Bus Controller and the 82284 Clock Generator interfaces the 80286 processor or some other bus master to a multi-master system bus. The arbiter multiplexes a processor onto a multi-master system bus. It avoids contention with other bus masters.

The 82289 has two separate state machines which communicate through bus request and release logic. The processor interface state machine is synchronous with the local system clock (CLK) and the multi-master system bus interface state machine is synchronous with the bus clock (BCLK#).

The 82289 performs all signalling to request, obtain, and release the system bus. External logic is used to determine which bus cycles require the system bus and to resolve priorities of simultaneous requests for control of the system bus.

82289 with 80286

In an iAPX 286 system using an 82289 Bus Arbiter, the 80286 processor is unaware of the arbiter's existence and issues cammands as though it had exclusive use of the multi-master system bus such as MULTIBUS. If the processor cycle requires MULTIBUS access, the arbiter requests control of the MULTIBUS. Until the request is granted the 82289 keeps AEN# disabled to prevent the 82288 Bus Controller and the address latches from accessing the MULTIBUS. AEN# inactive also disasserts the asynchronous ready enable (ARDYEN#) input of the 82284 clock chip so that the system bus will appear as "NOT READY" to the 80286 processor.

Once the 82289 Bus Arbiter has acquired the bus, it will assert AEN# allowing the 82288 Bus Controller and the address latches to access the system bus and asserting the ARDYEN# Input of the 82284 Clock chip.

Typically, once the data transfer command has been issued by the 82288 and the data transfer has taken place, a transfer acknowledge (XACK#) signal is returned to the processor onth multi-master system bus to indicate "Ready" from the accessed slave device. The processor remains in a series of "Wait States" (Repeated Tc states) until the addressed device reponds with XACK# asserted signal to the 82284 ARDY# input and the 82284 asserts READY# to the processor. The processor then completes its bus cycle.

82289 with other Bus Masters

When supporting other bus masters, the S0#/HOLD and READY# pins of the bus arbiter can be connected to the 'Hold' pin of that master. The inverted AEN# signal from the 82289 can be used as the hold acknowledge (HLDA) input for the other bus master.

The bus master sends a HOLD signal to the bus arbiter when it needs the system bus for a memory access. If the arbiter currently controls the system bus, AEN# will be active. Otherwise, AEN# will be inactive and the arbiter will request control of the system bus. The bus master will have to wait until the 82289 has asserted AEN# (LOW), before it starts its bus cycle.

When the bus master no longer requests the MULTIBUS it will have to inactivate the HOLD signal. The arbiter interprets the MULTIBUS access as a single bus cycle which is terminated by HOLD going inactive (LOW). Thus the arbiter will not release the MULTIBUS to any other bus master during a bus access cycle.

Processor Cycle Definition

Any iAPX 286 system which gains access to the MULTIBUS through the 82289 Bus Arbiter uses an internal clock which is one half the frequency of the system clock (CLK) (see figure 3). Knowledge of the phase of the local bus master internal clock is required for proper 82289 control of the iAPX 286 interface to MULTIBUS. The local bus master informs the bus arbiter of its internal clock phase when it asserts the status signals. The 80286 S0# and S1# status signals are always first asserted in phase 1 of the local bus master's internal clock.

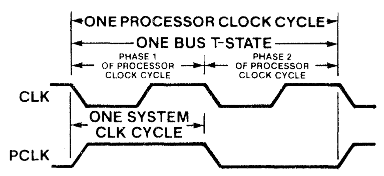


Figure 3: CLK Relationship to Internal Processor Phase, and Bus T-States

Bus State Definition

The 82289 Bus Arbiter has three processor bus states (see figure 4): Idle (T_I), Status (T_S), Command (T_C). Each bus state is two CLK cycles long. Bus state phases correspond to the internal CPU processor clock phases.

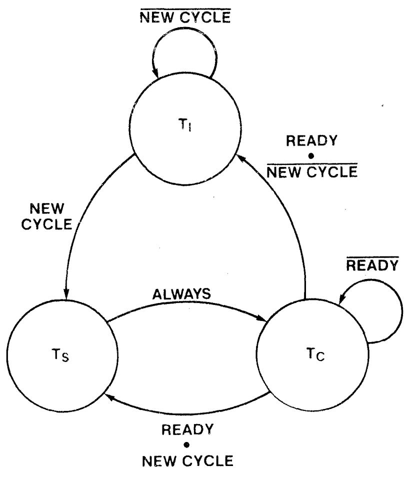


Figure 4: 82289 Processor Bus States

Bus Cycle Definition

The S1# and S0# status inputs are sampled by the 82289 on the falling edge of CLK and signal the start of a bus cycle by going active (LOW). The T_S bus state is defined to be the two CLK cycles during which either S1# or S0# is active (see figure 5). When either S1# or S0# is sampled LOW, the next CLK cycle is considered the second phase of the associated processor clock cycle.

The arbiter enters the T_C bus state after the T_S state. The shortest bus cycle may have one T_S state and one T_C state. Longer bus cycles are formed by repeating T_C states. A repeated T_C bus state is called a wait state.

The READY# input determines whether the current T_C bus state is to be repeated. The READY# input has the same timing and effect for all bus cycles. READY# is sampled at the end of each T_C bus state to see if it is active. If sampled HIGH, the T_C bus state is repeated. This is called inserting a wait state.

When READY# is sampled LOW, the current bus cycle is terminated. Note that the bus arbiter may enter the T_S bus state directly from T_C if the status lines are sampled active (LOW) at the next falling edge of CLK (see Figure 5). If neither of the status lines are sampled active at that time the 82289 will enter the T_I bus state. The T_I bus state will be repeated until the status inputs are sampled active.

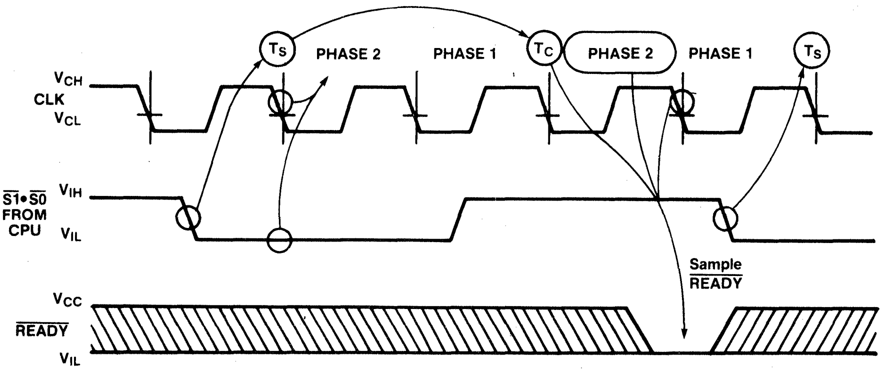


Figure 5: 80286 Bus Cycle Definition (without wait states)

Arbitration Between Bus Masters

The MULTIBUS protocol allows multiple processing elements to compete with each other to access common system resources. Since the local 80286 processor does not have exclusive use of the system bus, if the MULTIBUS is "BUSY" the 80286 processor will have to wait before it can access the system bus.

The 82289 Bus Arbiter provides an integrated solution for controlling access to a multi-master system bus. The bus arbiter allows both higher and lower priority bus masters to acquire the system bus depending on which release mode is used. In general, higher priority masters obtain the bus immediately after any lower priority master completes its present transfer cycle. Lower priority bus masters obtain the bus when a higher priority master is not accessing the system bus or the proper surrender conditions exist. The 82289 handles this arbitration in a manner completely transparent to the bus master (e.g. 80286 processor).

At the end of each transfer, the arbiter may retain or release the system bus. This decision is controlled by the processor state, but arbitration inputs and arbiter strapping options. (See Releasing The MULTIBUS, ahead).

Priority Resolving Techniques

Some means of resolving priority between bus masters requesting the multi-master bus simultaneously must be provided. The 82289 Bus Arbiter supports parallel, serial, and rotating system bus priority resolving techniques. All of these techniques are based on the concept that at a given time, one bus master will have priority above all the others.

231095-003

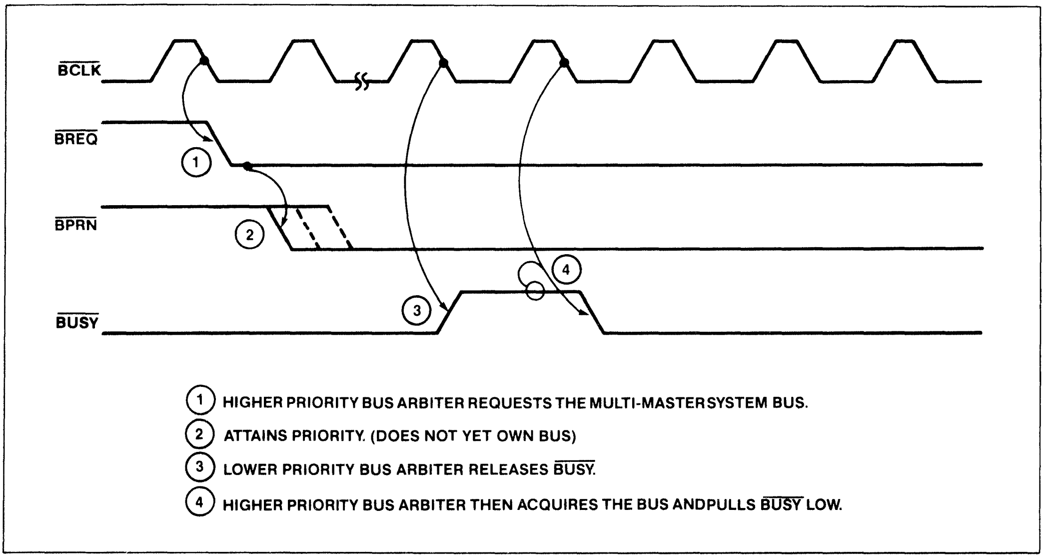


Figure 6: Bus Exchange Timing For The MULTIBUS®

An individual arbiter is the highest priority arbiter requesting the MULTIBUS when its BPRN# input is asserted (LOW). The highest priority requesting arbiter cannot immediately seize the system bus. It must wait until the present bus transaction is completed. Upon completing its current transaction the present bus owner surrenders the bus by releasing BUSY#.

BUSY# is an active-low 'Wired-OR' MULTIBUS signal which goes to every bus arbiter on the system bus. When BUSY# goes inactive, the arbiter which has requested the system bus, *and* presently has bus priority (BPRN# LOW), seize the bus by pulling BUSY# LOW (See waveform in Figure 6).

The generation of a multi-master bus request (BREQ#) is controlled by the type of bus cycle and the SYSB/RESB# input. Whenever the processor signals the status for memory read, memory write, I/O read, I/O write or interrupt acknowledge cycle, and SYSB/RESB# is HIGH at the end of T_S, a bus request is generated.

When the status inputs indicate an interrupt acknowledge bus cycle, the arbiter allows external logic to decide (through the SYSB/RESB# input) whether the interrupt acknowledge cycle should use the MULTIBUS

Figure 7 shows how SYSB/RESB# is repeatedly sampled until it is sampled HIGH or the bus cycle is terminated. If the bus cycle is completed (READY# is sampled LOW) before SYSB/RESB# is sampled HIGH, the arbiter will not request the MULTIBUS.

The 82289 bus Arbiter does not generate a separate BREQ# for each bus cycle. Instead the 82289 generates BREQ# when it requests the bus and holds BREQ# active during the time that it has possession of the bus. Note that all multi-master system bus requests (via BREQ#) are synchronized to the system bus clock (BCLK#).

Parallel Priority Resolving Technique

The parallel priority resolving technique requires a separate bus request line (BREQ#) for each arbiter on the multi-master system bus (see Figure 8). Each BREQ# line enters a priority encoder which generates the binary address of the highest priority BREQ# line currently active. The binary address is decoded to select the BPRN# line corresponding to the highest priority arbiter requesting the bus. In a parallel scheme, the BPRO# output is not used.

The arbiter receiving priority (BPRN# LOW) then allows its associated bus master onto the multi-master system bus as soon as the bus becomes available (i.e., the bus is no longer busy). Any number of bus masters may be acomodated in this way, limited only by the complexity of the external priority resolving circuitry. Such circuitry must resolve the priority within one BCLK# period.

Serial Priority Resolving Technique

The serial priority resolving technique eliminates the need for the priority circuitry of the parallel technique by daisy-chaining the bus arbiters together, that is, connecting the higher priority arbiter's BPRO# output to the BPRN# of the next lower priority arbiter (see Figure 9). The highest priority bus arbiter would have its BPRN# tied LOW in this configuration, signifying to the arbiter that it always has the highest priority when requesting the system bus. In a serial scheme, the BREQ# output is not used.

Since arbitration must be resolved within one BCLK# period the number of arbiters connected together in the serial priority is limited by arbiter BPRN# to BPRO# propagation delay (18 ns). For a 10 MHz MULTIBUS BCLK#, five 82289 Bus Arbiters may be connected together in serial configuration.

Maximum number of chained-priority devices =

$$\frac{\text{BCLK\# period}}{\text{BPRN\# to BPRO\# delay}}$$

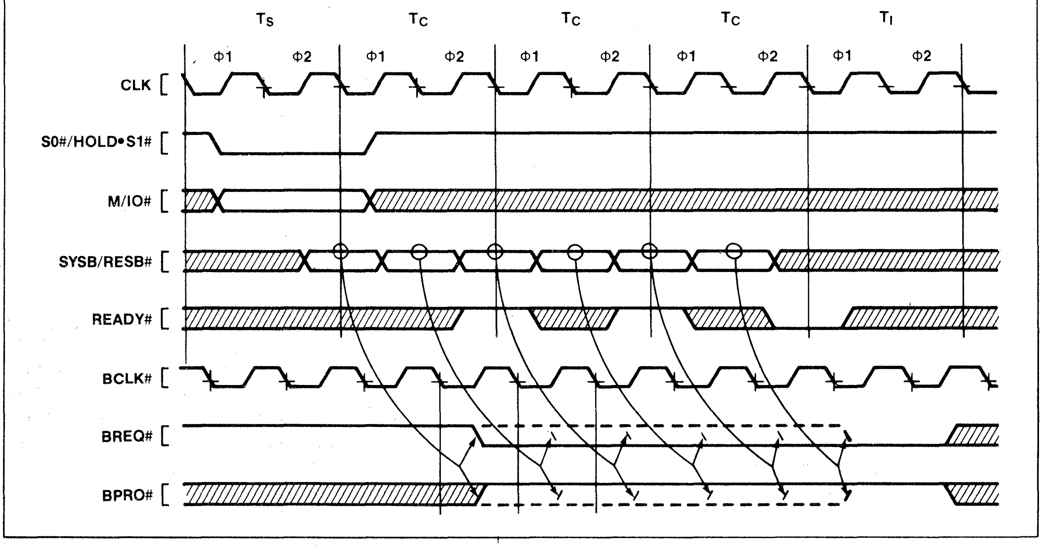


Figure 7: Bus Request Timing During an Interrupt Acknowledge Cycle

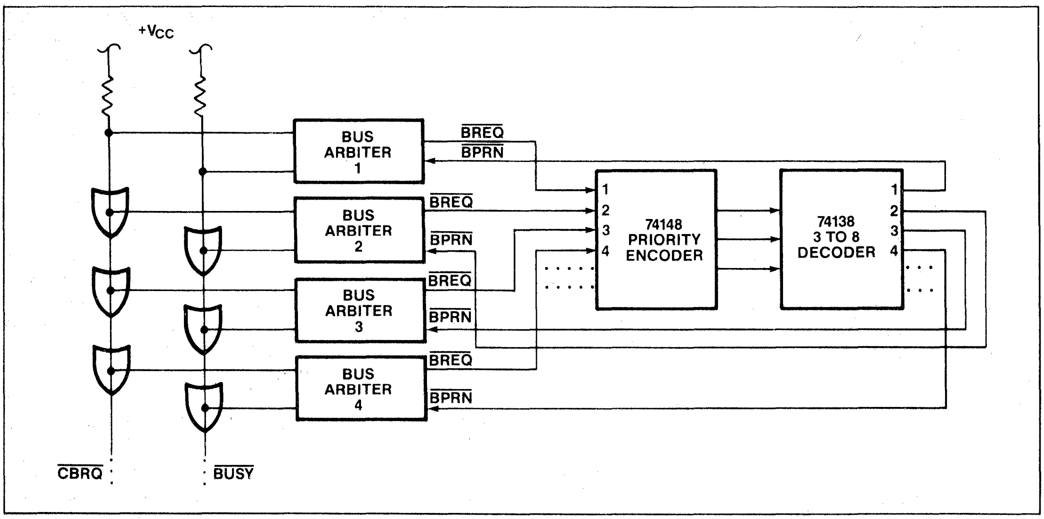


Figure 8: Parallel Priority Resolving Technique

231095-003

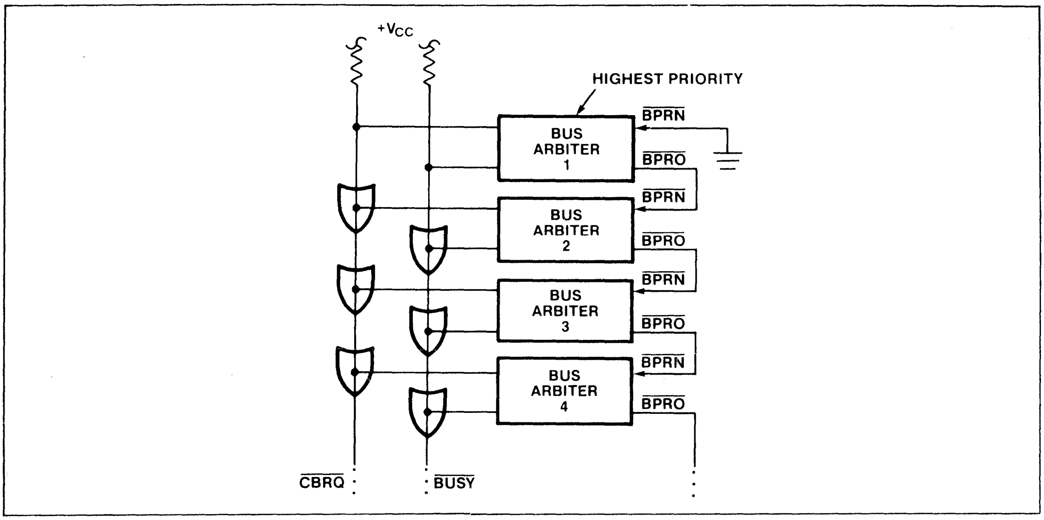
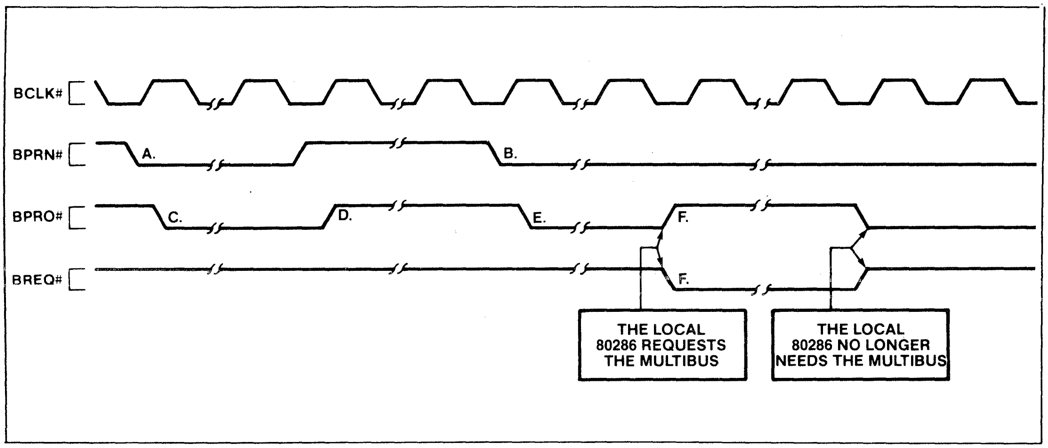


Figure 9: Connections for Serial Priority Resolving Technique



Note: Events A through F described above.

Figure 10: Serial Priority Bus Behavior

When using the serial priority resolving scheme, a higher priority arbiter (for example, arbiter 2, Figure 9) passes priority to the next lower priority arbiter (arbiter 3) by asserting its BPRO# signal (LOW). This asserts BPRN# of next arbiter (arbiter 3) as shown in Figure 10-a & 10-b. An arbiter's BPRO# is asserted if the arbiter has priority (BPRN# is asserted) but is not accessing or requesting the system bus (as indicated by BREQ# inactive as shown in Figure 10-c and 10-e for arbiter 3). Whenever a higher priority arbiter (arbiter 3) issues a bus request its BPRO# goes inactive causing the next lower priority arbiter (arbiter 4) to lose its bus priority (Figure 10-f). Any arbiter (arbiter 3) will also bring its BPRO# inactive if its BPRN# goes inactive (from arbiter 2), thereby passing the loss of bus priority on to the lower priority arbiters (e.g. arbiter 4) as shown in Figure 10-d.

Rotating Priority Resolving Technique

The rotating priority resolving technique is similar to the parallel priority resolving technique except that priority is dynamically re-assigned. The priority encoder is replaced by a more complex circuit which rotates priority between requesting arbiters, thus allowing each arbiter an equal chance to use the multi-master system bus over a given period of time.

Selecting the Appropriate Priority Resolving Technique

The choice of a priority resolving technique involves a tradeoff between external logic complexity and ease of the MULTIBUS access for the different bus masters in the system. The rotating priority resolving technique requires a substantial amount of external logic, but guarantees all the bus masters an equal opportunity to access the system bus. The serial priority resolving technique uses no external logic but has fixed bus master priority levels and can accommodate only a limited number of bus arbiters. The parallel priority resolving technique is in general a compromise between the other two techniques. (For example parallel priority configuration in Fig. 8 allows up to eight arbiters to be present on the MULTIBUS, with fixed priority levels, while not requiring a large amount of complex, external logic to implement.)

Releasing the MULTIBUS®

Following a data transfer cycle on the MULTIBUS, the 82289 Bus Arbiter can either retain control of the system bus or release the bus for use by some other bus master. The 82289 can operate in one of three modes, defining different conditions under which the arbiter relinquishes control of the multi-master system bus. These release modes are described in Table 2.

Release Mode	Conditions under which the Bus Arbiter releases the system bus (unless cycles are LOCKed)
Mode 1	The Bus Arbiter always releases the bus at the end of each transfer cycle
Mode 2	The Bus Arbiter retains the bus until: • a higher-priority bus master requests the bus, driving BPRN# HIGH • a lower-priority bus master requests the bus by pulling CBRQ# LOW
Mode 3	The Bus Arbiter retains the bus until: • a higher-priority bus master requests the bus, driving BPRN# HIGH. (CBRQ# LOW ignored)

Table 2: 82289 Release Modes

If the arbiter was programmed to operate in the Always Release mode (Mode 1) during the previous reset, it will surrender the MULTIBUS after each complete transfer cycle. If the arbiter is not in the Always Release mode, it will not surrender the bus until the local 80286 processor enters a halt state, the arbiter is forced off of the bus by the loss of BPRN# (Mode 2 or 3), or by a common bus request when the CBRQ# input is enabled by the CBQLCK# input (Mode 2).

CBRQ# can save the bus exchange overhead in many cases. If CBRQ# is high, it indicates to the bus master that no other master is requesting the bus and therefore the present bus master can retain the bus. Without CBRQ#, only BPRN# indicates whether or not another master is requesting the bus and, that only if the other master is of higher priority. Between the master's bus transfer cycles, in order to allow lower priority masters to take the bus if they need it, the master must give up the bus. At the start of the master's next transfer cycle, the bus must be regained. If no other master has the bus, this can take approximately two BCLK# periods. To avoid this overhead of unnecessarily giving up and regaining the bus when no other masters need it, CBRQ# is extremely useful. Any master that wants but does not have the bus, must assert CBRQ# (LOW). If CBRQ# line is not asserted the bus does not have to be released, thereby eliminating the delay of regaining the bus at the start of the next cycle.

The LOCK# input to the arbiter can be used to override any of the conditions shown in Table 2. While LOCK# is asserted, the arbiter will not surrender control of the MULTIBUS to any other requesting arbiter. Note that the arbiter will surrender the MULTIBUS (synchronous to BCLK#) either in response to RESET or INIT# signals independent of the current release mode or the state of the arbiter inputs.

The three bus release modes have the same operation when supporting either the 80286 processor or some other bus master.

Selecting the Appropriate Release Mode

The choice of which release mode to use may affect the bus utilization of the individual subsystems, and the system as a whole. Mode dependent performance variations are due to the bus acquisition/release overhead. The effect of these acquire and release times on system bus efficiency is illustrated in Figure 11.

An isolated transfer on the multi-master system bus is depicted in Figure 11-a. Figure 11-b shows utilization for the bus arbiter operating in Mode 1. The arbiter must request and release the system bus for each transfer cycle. Lower priority arbiters have easy access to the system bus, but overall bus efficiency is low. Bus utilization for a bus arbiter operating in Mode 2 or 3 is shown in Figure 11-c. In this situation the arbiter acquires the bus once for a sequence of transfers. The arbiter retains the bus until forced off by another bus master's request as defined in Table 2.

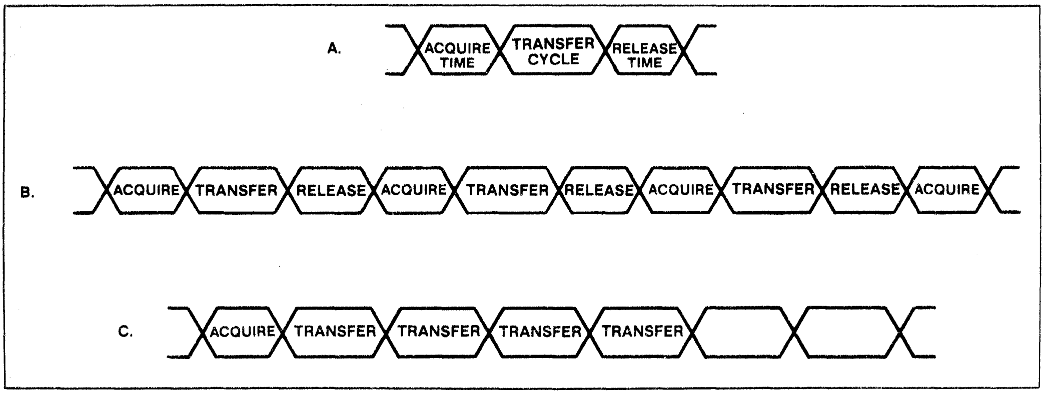


Figure 11: Effects of Bus Release Mode on Bus Efficiency

The three release modes of the 82289 allow the designer to optimize the system use of the MULTIBUS.

Configuring the 82289 Release Mode

The 82289 Bus Arbiter can be configured in any of its three bus release modes without additional hardware. the 82289 can also be configured to switch between Mode 2 and Mode 3 under software control of the 80286 processor, requiring that a parallel port or addressable latch be used to drive the ALWAYS#/CBQLCK# input pin of the 82289 (see Figure 12).

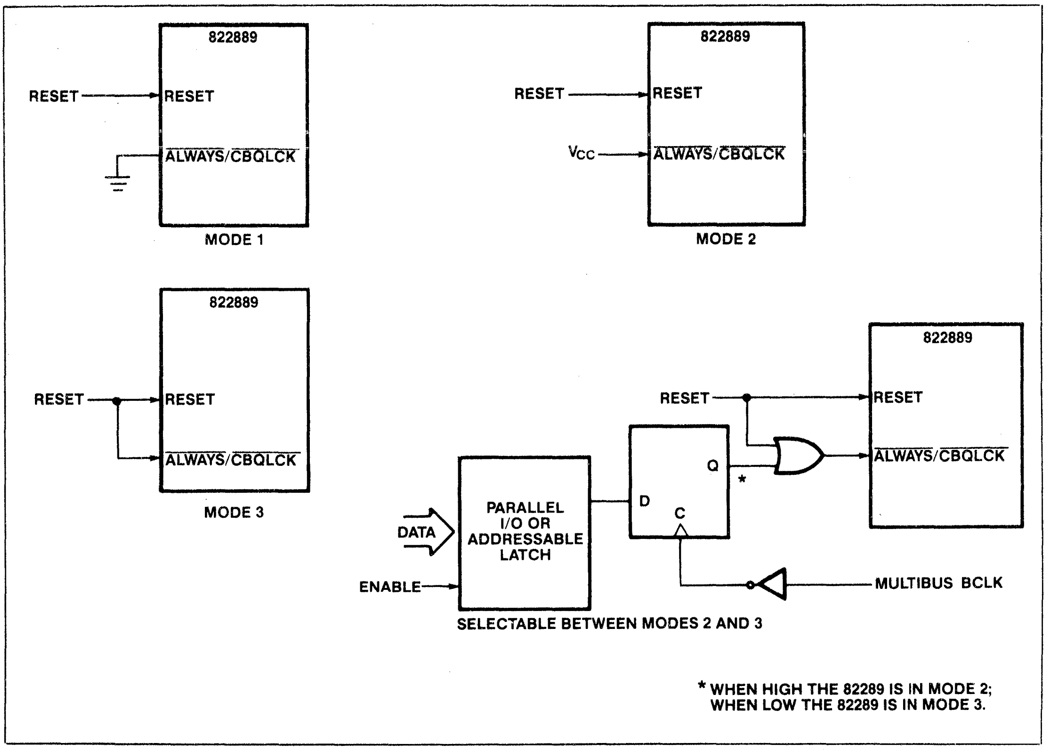


Figure 12: 82289 Release Mode Configurations

Asserting the LOCK# Signal

Independent of the particular release mode of the 82289 Bus Arbiter, the 80286 processor can assert a LOCK# signal synchronously to CLK to prevent the arbiter from releasing the Multibus. This software-controlled LOCK# signal prevents the 82289 from surrendering the system bus to any other bus master, whether that bus master is of higher or lower priority. The LOCK# signal is typically used for implementing software semaphores for shared resources or for critical processes that must run in real-time.

The 82289 LLOCK# output is the Multibus signal asserted during all bus cycles which are locked together. The LLOCK# is set or reset depending on processor LOCK# at the end of the T_S cycle. The LLOCK# will delay going inactive until the termination of the current transfer cycle.

The 82289 will continue to assert the LLOCK# signal, retaining control of the MULTIBUS, until the end of the first 'unLOCKed' 80286 bus cycle (80286 disables its LOCK# output on the last bus cycle indicating that no future locked cycles are needed). While the LOCK# signal will force the arbiter presently in control to hold the system bus, it cannot force another arbiter to surrender the bus any earlier than it normally would.

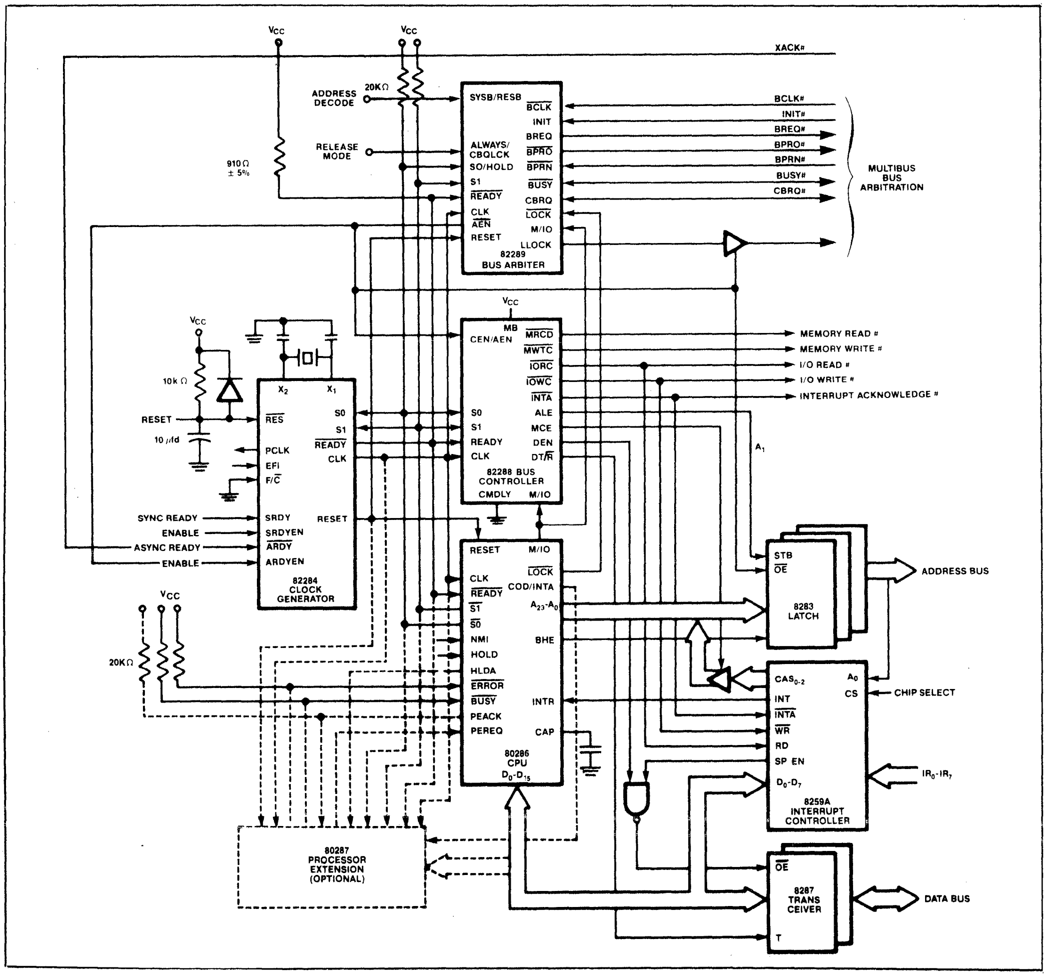
The LLOCK# signal from the 82289 must be connected to a tri-state buffer in order to drive the MULTIBUS LOCK# signal. This tri-state buffer should be enabled by the AEN# signal from the arbiter going active.

82289 Reset and Initialization

The 82289 Bus Arbiter provides the RESET and INIT# pins for initialization. RESET is a CLK synchronous signal from the 80286 processor and INIT# is an asynchronous signal on the multi-master system bus. By having RESET pin high or INIT# pin low, the BREQ#, BUSY#, and AEN# output pins will all be cleared and become inactive. RESET will also clear the LLOCK# signal. Unlike RESET, INIT# will not clear any pending bus request; the bus request would be asserted after the INIT# signal goes inactive.

Note that when the 82289 is initialized by the RESET input it does not wait until the end of the current bus cycle to reset. Any bus cycle in process when RESET goes active will be aborted by the arbiter. Although the INIT# signal will also interrupt an active bus cycle, the arbiter can request the MULTIBUS and complete the bus cycle when INIT# goes inactive.

As mentioned in the Table 1 Pin Description and Figure 12, the functions of the S0#/HOLD pin and the release mode (ALWAYS#/CBQLCK# pin) are programmed at the falling edge of RESET.



Schematic 1: Typical iAPX 286 Subsystem MULTIBUS® Interface

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature
  Under Bias .... 0°C to 70°C
Storage Temperature .... -65°C to +150°C
Case Temperature .... 0°C to 85°C
Voltage on Any Pin With
  Respect to GND .... -0.5V to +7V
Power Dissipation .... 1 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

Electrical Characteristics and Waveforms

D.C. Characteristics (T_A = 0° to 70°C, T_{CASE} = 0°C to 85°C, V_{CC} = 5V ±5%)

Symbol	Parameter	Preliminary Min.	Preliminary Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	-0.5	.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{ILC}	CLK Input Low Voltage	-0.5	.6	V	
V_{IHC}	CLK Input High Voltage	3.8	V_{CC} + 1.0	V	
V_{OL}	Output Low Voltage: BUSY#, CBRQ# BPRO#, BREQ#, AEN# LLOCK#		 .45 .45 .45	 V V V	 I_{OL} = 32mA I_{OL} = 16mA I_{OL} = 5mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = 400 μA
I_{LI}	Input Leakage Current		±10 ±1	μA mA	$0.45V \leq V_{IN} \leq V_{CC}$ $0V \leq V_{IN} < 0.45V$
I_{LO}	Output Leakage Current		±10	μA	$0.45V \leq V_{OUT} \leq V_{CC}$
I_{CC}	Power Supply Current		120	mA	
C_{CLK}	CLK, BCLK# Input Capacitance		12	pF	F_C = 1 MHz
C_{IN}	Input Capacitance		10	pF	F_C = 1 MHz
C_O	Input/Output Capacitance		20	pF	F_C = 1 MHz

231095-003

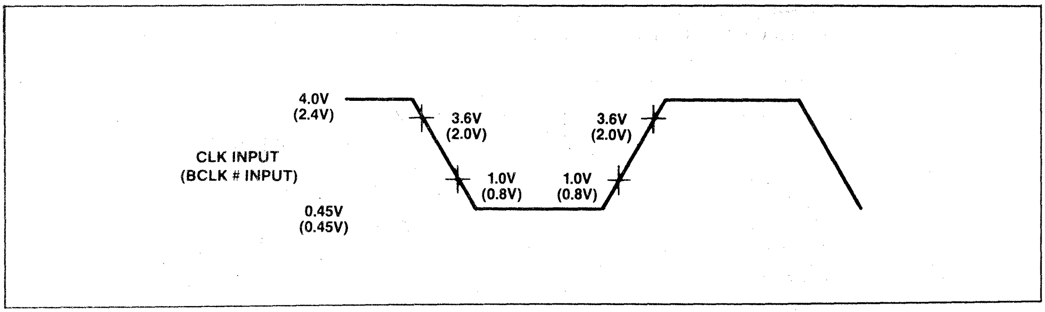
A.C. Characteristics (T_A = 0° to 70°C, T_{CASE} = 0° to 85°C, V_{CC} = 5V ± 5%)

AC timings are referenced to 0.8V and 2.0V points of signals as illustrated in datasheet waveforms, unless otherwise noted.

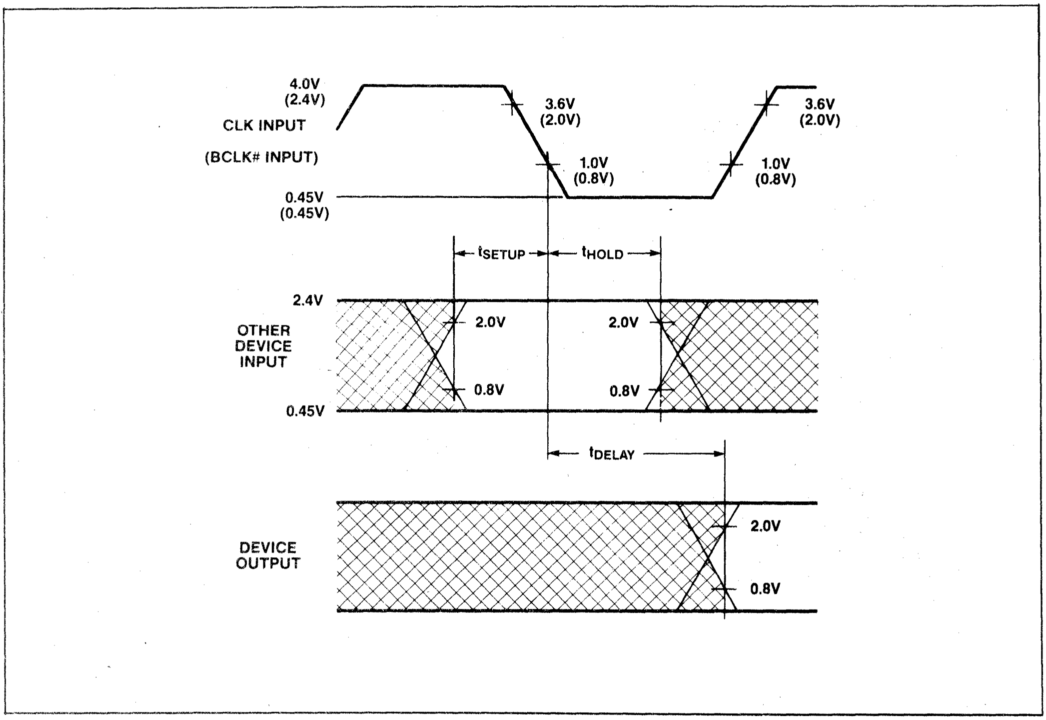
Sym	Parameter	Preliminary 6MHz Min.	Preliminary 6MHz Max.	Preliminary 8MHz Min.	Preliminary 8MHz Max.	Unit	Test Conditions	Shown in Figure
1	CLK Cycle Period	83	t5+ 50	62	t5+ 50	ns		13
2	CLK Low Time	20	225	15	230	ns	at 1.0 V	13
3	CLK High Time	25	230	20	235	ns	at 3.6 V	13
4	CLK Rise/Fall Time		10		10	ns	1.0 to 3.6 V	13
5	BCLK# Cycle Time	100	∞	100	∞	ns		13
6	BCLK# High/Low Time	30		30		ns		13
7	S0#/HOLD, S1#, M/IO# Setup	28		22		ns		13
8	S0#/HOLD, S1#, M/IO# Hold	1		1		ns		13
9	READY# Setup	50		38		ns		13
10	READY# Hold Time	35		25		ns		13
11	LOCK#, SYSB/RESB# Setup Time	28		20		ns		13, 18
12	LOCK#, SYSB/RESB# Hold Time	1		1		ns		13, 18
13	RESET Setup Time	28		20		ns		19
14	RESET Hold Time	1		1		ns		19
15	RESET ACTIVE Pulse Width	16		16		CLKs		19
16	INIT# Setup Time	45		45		ns	Note 9	20
17	INIT# Hold Time	1		1		ns	Note 9	20
18	INIT# Active Pulse Width	3(t1) +3(t14)		3(t1) +3(t14)		ns		20
19	BUSY#, BPRN#, CBRQ#, CBQLCK#/ALWAYS# Setup to BCLK# (or to RESET)	20		20		ns		13, 15, 21
20	BUSY#, BPRN#, CBRQ#, CBQLCK#/ALWAYS# Hold to BCLK# (or to RESET)	1		1		ns		13, 15, 21
21	BCLK# to BREQ# Delay		30		30	ns	Note 1	13, 14
22	BCLK# to BPRO# Delay		35		35	ns	Note 2	17
23	BPRN# to BPRO# Delay		25		25	ns	Note 2	17
24	BCLK# to BUSY# Active Delay	1	60	1	60	ns	Note 3	13
25	BCLK# to BUSY# Float Delay		35		35	ns	Note 4	13, 14
26	BCLK# to CBRQ# Active Delay		55		55	ns	Note 5	13
27	BCLK# to CBRQ# Float Delay		35		35	ns	Note 4	13, 20
28	BCLK to AEN# Active Delay	1	25	1	25	ns	Note 6	13
29	CLK to AEN# Inactive Delay	3	25	3	25	ns	Note 6	13, 14
30	CLK to LLOCK# Delay		20		20	ns	Note 7	18
31	RESET to LLOCK# Delay		35		35	ns	Note 7	19
32	CLK to BCLK# Setup Time	38		38		ns	Note 8	13, 16, 20

NOTES:

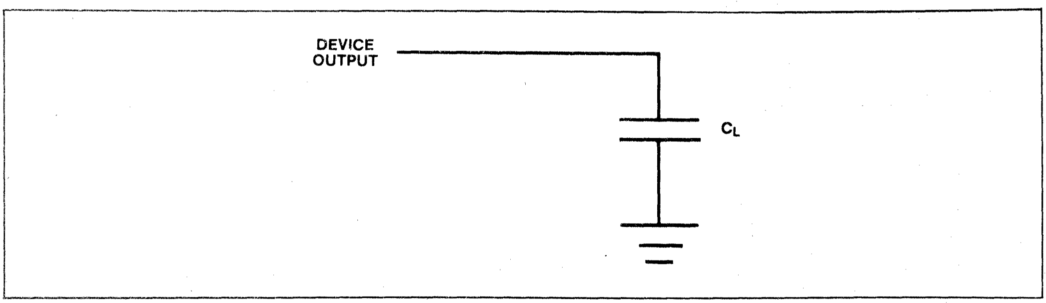
NOTE 1: BREQ# load: C_L = 60pF.
NOTE 2: BPRO# load: C_L = 60pF.
NOTE 3: BUSY# load: C_L = 300pF.
NOTE 4: Float condition occurs when output current is less that I_{LO} in magnitude.
NOTE 5: CBRQ# load: C_L = 300pF.
NOTE 6: AEN# load: C_L = 150pF.
NOTE 7: LLOCK# load: C_L = 60pF.
NOTE 8: In actual use, CLK and BCLK# are usually asynchronous to each other. However, for component testing purposes, this specification is required to assure signal recognition at specific CLK and BCLK# edges.
NOTE 9: INIT# is asynchronous to CLK and to BCLK#. However for component testing purposes, this specification is required to assure signal recognition at specific CLK and BCLK# edges.



NOTE 10: AC Drive and Measurement Points — CLK Input (BCLK# Input)



NOTE 11: AC Setup, Hold and Delay Time Measurement — General



NOTE 12: AC Test Loading on Outputs

Waveforms

The waveforms (Figure 13-21) show the timing relationships of the inputs and the outputs and do not show all possible transitions of all signals in all modes. Instead, all signal timing relationships are shown via the general cases. Special cases are shown when needed.

To find the timing specification for a signal transition in a particular mode, first look for a special case in the waveforms. If no special case applies, then use a timing specification for the same or related function in another mode.

The 82289 Bus Arbiter serves as an interface between the iAPX 286 subsystem which operates synchronous to the CLK signal and MULTIBUS which operates synchronous to BCLK# signal. CLK and BCLK# generally operate asynchronously to each other and at different frequencies. Thus, the exact clock period in which an input synchronous to one clock will cause a response synchronous to the other clock depends on the relative phase and frequency of CLK and BCLK# at the time the input is sensed.

One strict relation between CLK and BCLK# must be maintained for proper MULTIBUS arbitration. If the CLK period is too long relative to BCLK# period (t1 greater than t5 + 50ns), another arbiter could gain control of the system bus before this arbiter has released AEN# synchronous to its CLK. This situation arises since the release of AEN# is synchronous to the next falling CLK edge after the processor cycle ends but the release of BREQ# and BUSY# is synchronous to the next falling BCLK# edge after the processor cycle ends. In practice, any CLK frequency greater than 6.66 MHz (ie. 80286 processor speeds greater than 3.33 MHz) will avoid conflict with a 10 MHz BCLK#. Therefore all 80286 speed selections are MULTIBUS compatible.

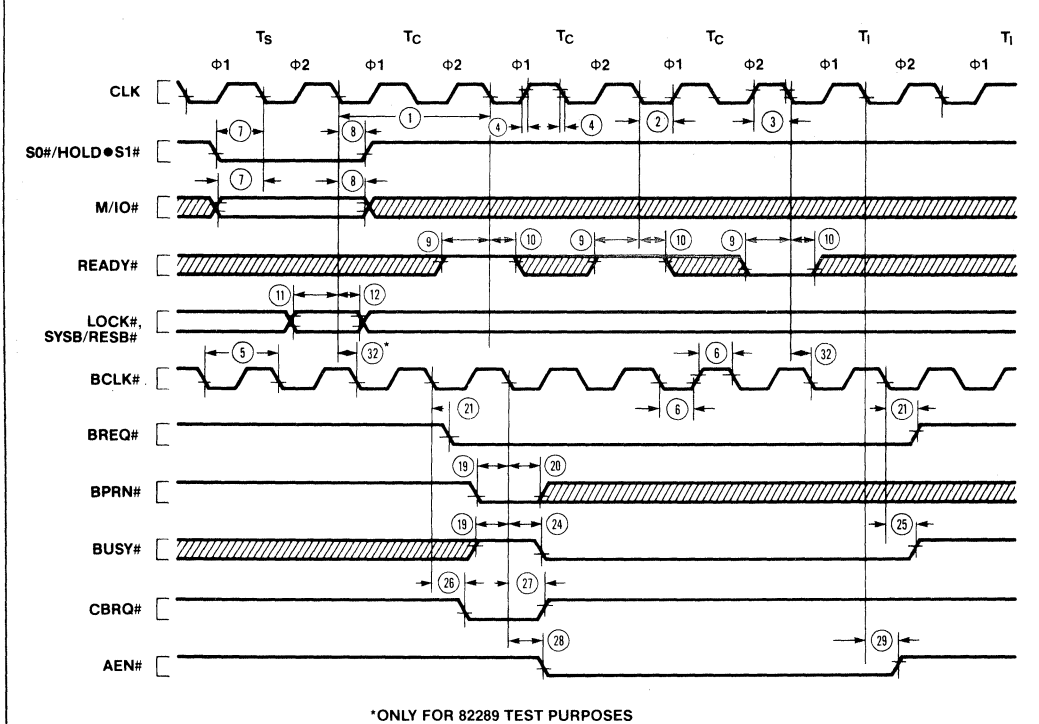


Figure 13: MULTIBUS® Acquisition and Always-Release Operation

231095-003

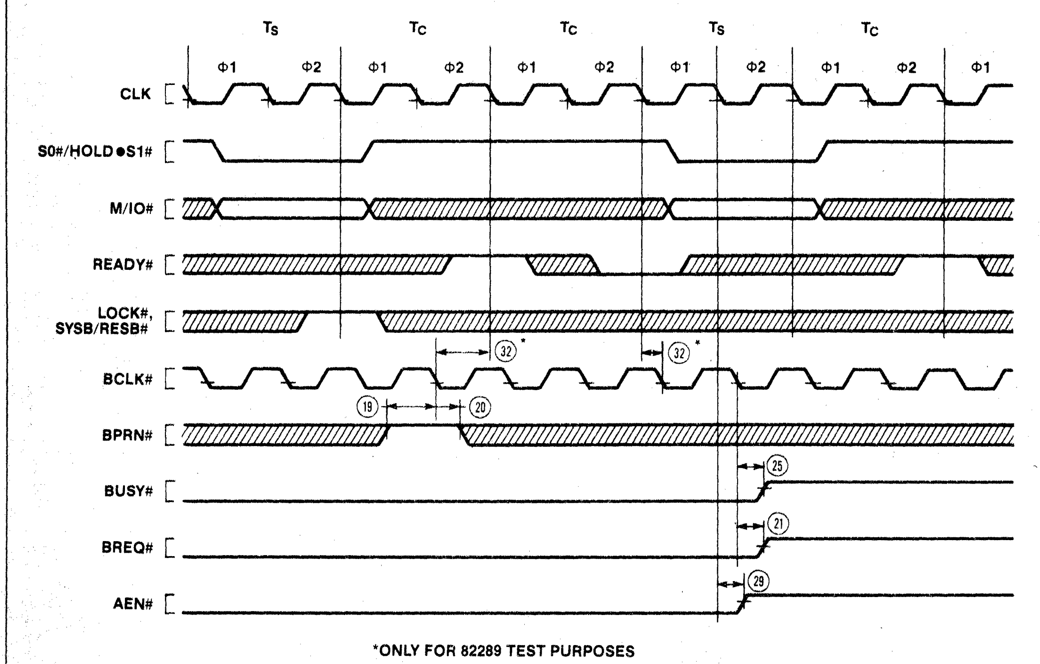


Figure 14: MULTIBUS® Release due to BPRN# Inactive

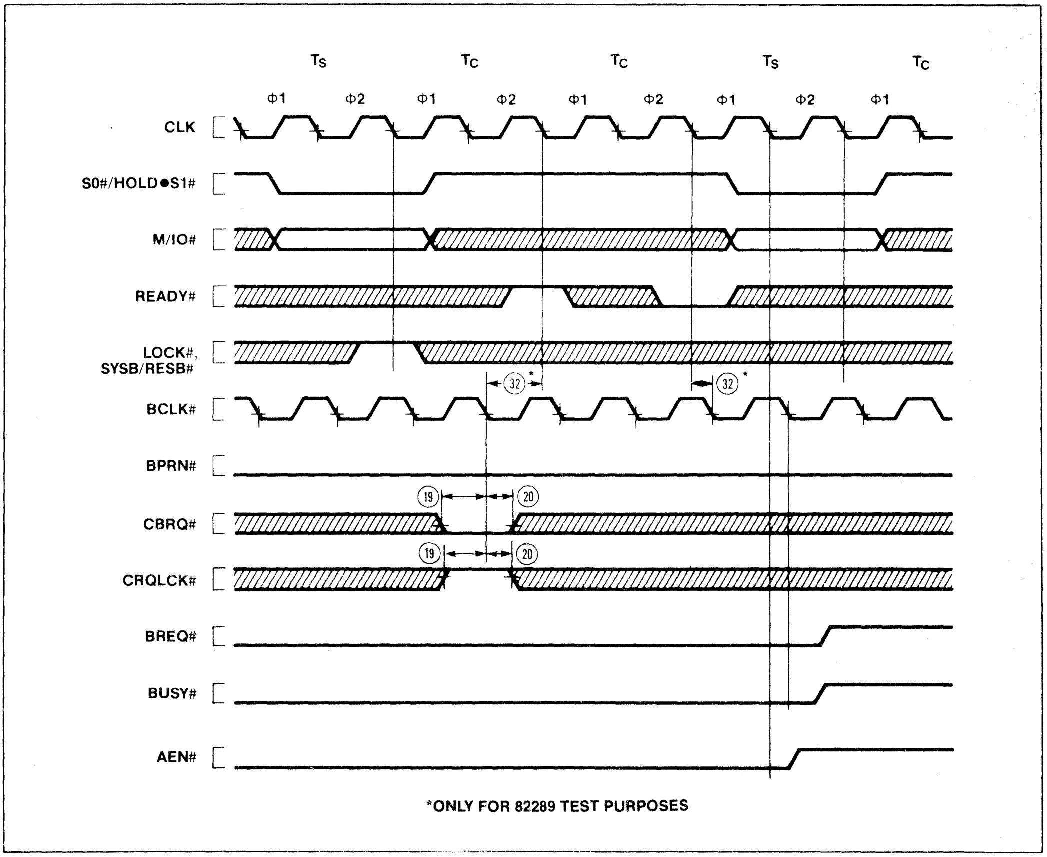


Figure 15: MULTIBUS® Release due to CBRQ# Active

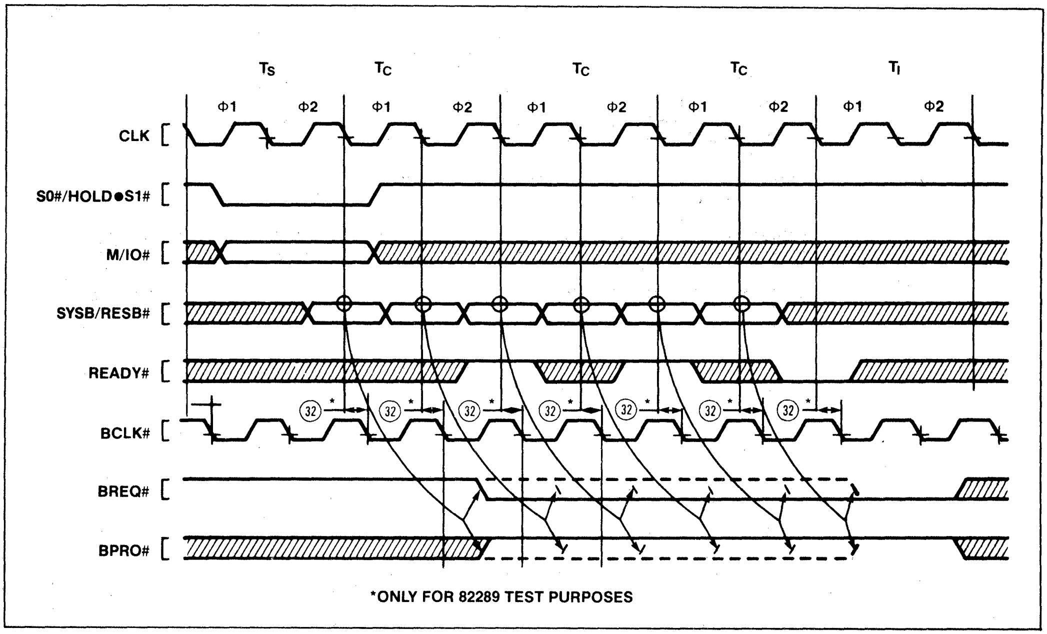


Figure 16: MULTIBUS® Acquisition During 80286 INTA Cycles

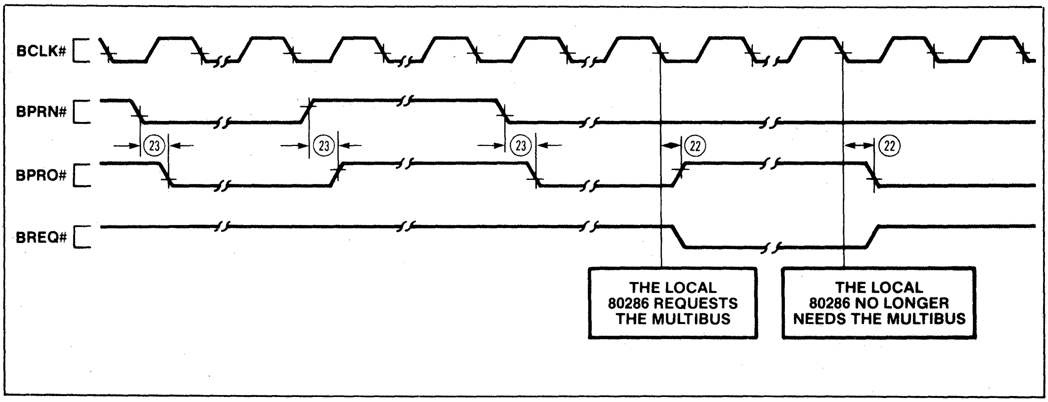


Figure 17: BPRN# to BPRO# Timing Relationship

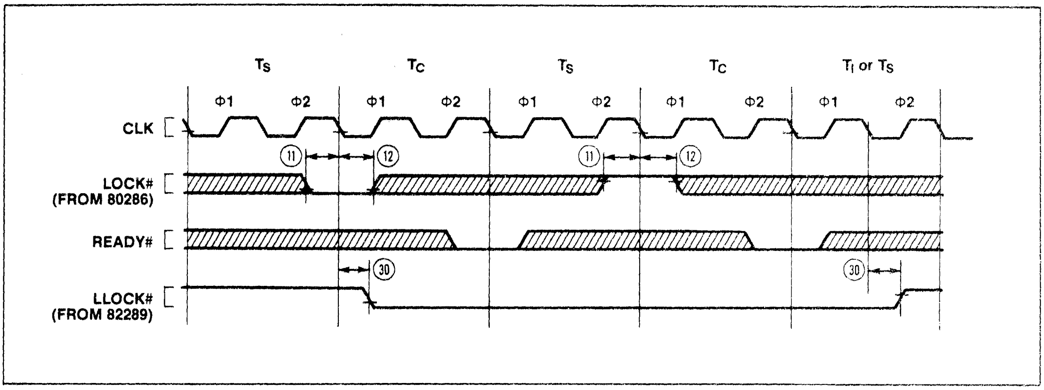


Figure 18: 80286 LOCK# and 82289 LLOCK# Relationship

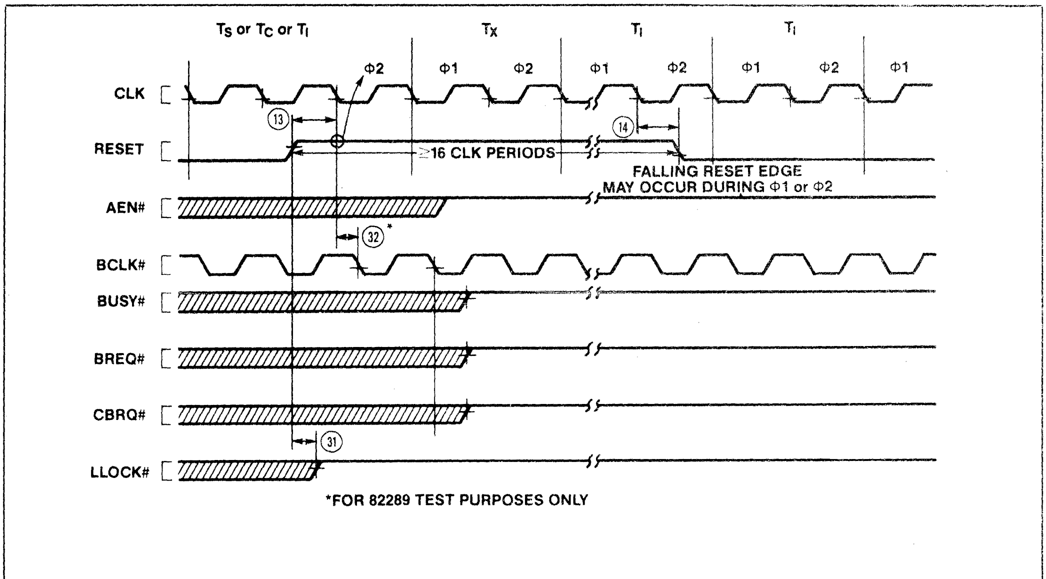


Figure 19: RESET Active Pulse

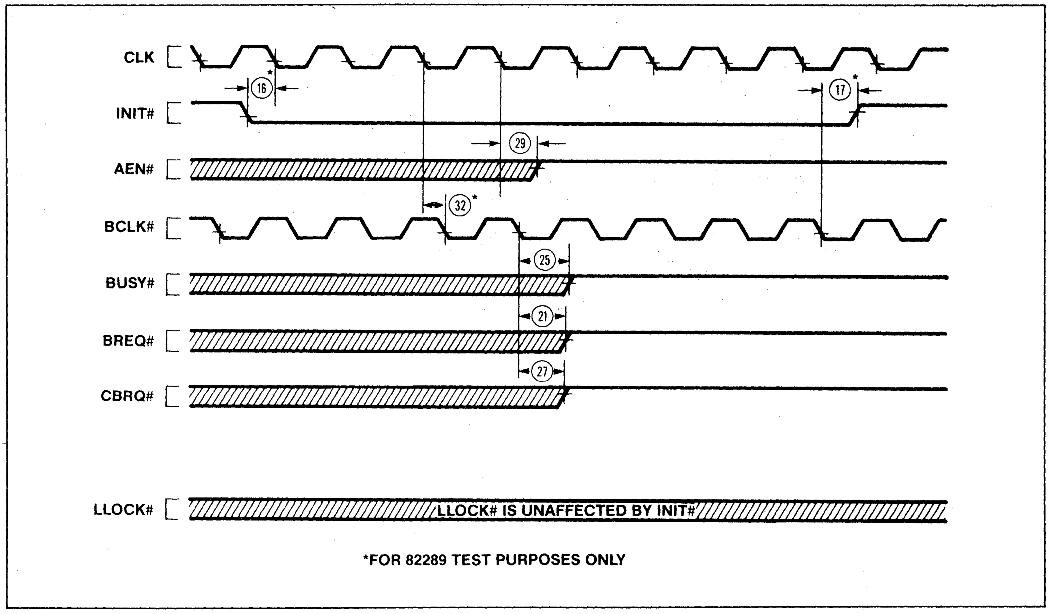


Figure 20: INIT# Active Pulse

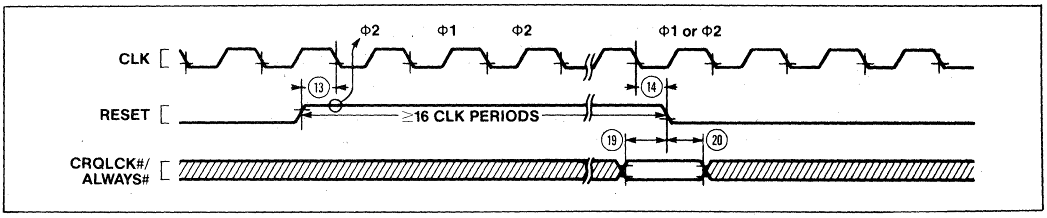


Figure 21: Programming the Always-Release/Common-Bus-Request-Release Option

80386 Microprocessors

5

ADVANCE INFORMATION

80386
HIGH PERFORMANCE MICROPROCESSOR WITH INTEGRATED MEMORY MANAGEMENT

- **Flexible 32-Bit Microprocessor**
  - **—8, 16, 32-Bit Data Types**
  - **—8 General Purpose 32-Bit Registers**
- **Very Large Address Space**
  - **—4 Gigabyte Physical**
  - **—64 Terabyte Virtual**
  - **—4 Gigabyte Maximum Segment Size**
- **Integrated Memory Management Unit**
  - **—Virtual Memory Support**
  - **—Optional On-Chip Paging**
  - **—4 Levels of Protection**
  - **—Fully Compatible with 80286**
- **Object Code Compatible with All 8086 Family Microprocessors**
- **Virtual 8086 Mode Allows Running of 8086 Software in a Protected and Paged System**
- **Hardware Debugging Support**
- **Optimized for System Performance**
  - **—Pipelined Instruction Execution**
  - **—On-Chip Address Translation Caches**
  - **—12.5 and 16 MHz Clock**
  - **—32 Megabytes/Sec Bus Bandwidth**
- **High Speed Numerics Support via 80287 and 80387 Coprocessors**
- **Complete System Development Support**
  - **—Software: C, PL/M, Assembler System Generation Tools**
  - **—Debuggers: PSCOPE, ICE™-386**
- **High Speed CHMOS III Technology**
- **132 Pin Grid Array Package**
  (See Packaging Specification, Order #231369)

The 80386 is an advanced 32-bit microprocessor designed for applications needing very high performance and optimized for multitasking operating systems. The 32-bit registers and data paths support 32-bit addresses and data types. The processor addresses up to four gigabytes of physical memory and 64 terabytes (2**46) of virtual memory. The integrated memory management and protection architecture includes address translation registers, advanced multitasking hardware and a protection mechanism to support operating systems. In addition, the 80386 allows the simultaneous running of multiple operating systems.

Instruction pipelining, on-chip address translation, a a high bus bandwidth ensure short average instruction execution times and high system throughput. The 80386 processor is capable of execution at sustained rates of between 3 and 4 million instructions per second.

The 80386 offers new testability and debugging features. Testability features include a self-test and direct access to the page translation cache. Four new breakpoint registers allow conditional or unconditional breakpoint traps on code execution or data accesses, for powerful debugging of even ROM-based systems.

Object-code compatibility with all iAPX 86 family members (8086, 8088, 80186, 80188, 80286) means the 80386 offers immediate access to the world's largest microprocesor software base.

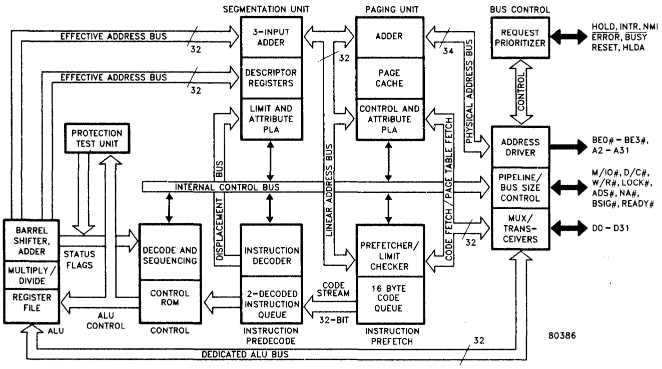


Figure 1-1. 80386 Pipelined 32-Bit Microarchitecture

Unix™ is a Trademark of AT&T Bell Labs.
MS-DOS is a Trademark of MicroSoft Corporation.

Order Number: 231630-001

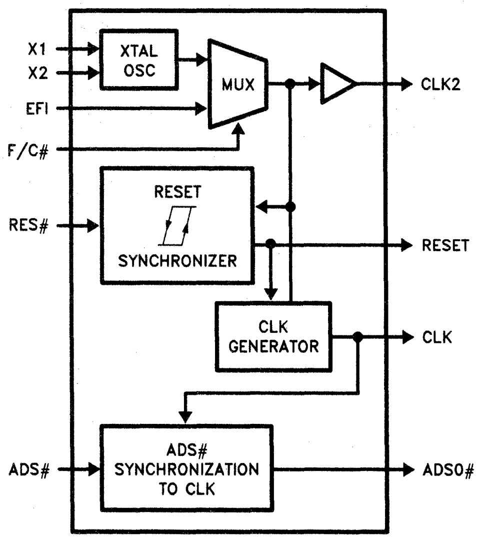
82384 CLOCK GENERATOR AND RESET INTERFACE FOR 80386 PROCESSORS

- **Generates All Clock Signals for 80386 Processors**
  - **—32 MHz CLK2 and 16 MHz CLK for 80386-16**
  - **—25 MHz CLK2 and 12.5 MHz CLK for 80386-12**
- **Generates Synchronous Reset from Schmitt-Trigger Input**
- **Generates Address Status Signal Synchronous to CLK**
- **Uses Crystal or TTL Signal for Frequency Source**
- **CHMOS III Technology**
- **Available in Two Speed Selections**
  - **—82384-16, Supporting Any 80386 Speed up to 16 MHz**
  - **—82384-12, Supporting Any 80386 Speed up to 12.5 MHz**
- **18-Pin Package**
  (See Packaging Specification, Order #231369)

INTRODUCTION

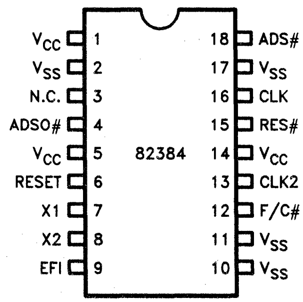
The 82384 combines a third-overtone crystal oscillator, reset synchronizing circuitry, and address status circuitry onto a single chip for easy timing and control of 80386-based systems.

The 82384 contains a clock generator/driver that provides two clock signals for the 80386-based systems. The CLK2 signal generated by the 82384 meets the 80386 CLK2 requirements, and the CLK signal indicates the 80386 processor phase. The 82384 also generates a synchronous reset signal from a schmitt-trigger reset input, and provides an Address Status signal that has guaranteed setup and hold timing with respect to the CLK output.



231659–1

Figure 1. Block Diagram



231659–2

Figure 2. Pin Configuration

October 1985
Order Number: 231659-001

Cut Along Dotted Line

80386
HIGH PERFORMANCE MICROPROCESSOR
ADDITIONAL INFORMATION

You can obtain a free copy of the most recent 80386 data sheet and related technical information by:

- Calling your local Intel sales office
- Calling toll-free (800) 538-1876
- Filling out this literature request form

Indicate areas of interest:

G27 ☐ Components: 80386 microprocessor, 82384 clock generator

G28 ☐ 386 Languages and development tools

G29 ☐ 386 single board computers

Name ______________________ Title ______________________

Company ______________________ Mailstop ______________________

Phone (____) ______________________________________

Address ______________________________________

City ______________________ State ________ ZIP ________

HPMO-308C

CARD EXPIRES 12/31/86

U.S. and Canada

Mail to: Intel Corporation
Mailstop: SC6-58
P.O. Box 58065
Santa Clara, CA 95052-8065

Customers outside the U.S. and Canada should contact the local Intel Sales Office or Distributor listed in the back of this book.

Cut Along Dotted Line

Cut Along Dotted Line

80386
HIGH PERFORMANCE MICROPROCESSOR
ADDITIONAL INFORMATION

You can obtain a free copy of the most recent 80386 data sheet and related technical information by:

- Calling your local Intel sales office
- Calling toll-free (800) 538-1876
- Filling out this literature request form

Indicate areas of interest:

G27 ☐ Components: 80386 microprocessor, 82384 clock generator

G28 ☐ 386 Languages and development tools

G29 ☐ 386 single board computers

Name ______________________ Title ______________________

Company ______________________ Mailstop ______________________

Phone (   ) ____

Address ____

City ______________________ State ________ ZIP ________

HPMO-308C

CARD EXPIRES 12/31/86

U.S. and Canada

Mail to: Intel Corporation
Mailstop: SC6-58
P.O. Box 58065
Santa Clara, CA 95052-8065

Customers outside the U.S. and Canada should contact the local Intel Sales Office or Distributor listed in the back of this book.

Cut Along Dotted Line

UNITED STATES
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

JAPAN
Intel Japan K.K.
5-6 Tokodai Toyosato-machi
Tsukuba-gun, Ibaraki-ken 300-26
Japan

FRANCE
Intel Paris
1 Rue Edison, BP 303
78054 Saint-Quentin en Yvelines
France

UNITED KINGDOM
Intel Corporation (U.K.) Ltd.
Piper's Way
Swindon
Wiltshire, England SN3 1RJ

WEST GERMANY
Intel Semiconductor GmbH
Seidlstrasse 27
D-8000 Munchen 2
West Germany

Printed in U.S.A./C421/1285/100K/RRD LD PM
Microprocessors/Peripherals

ISBN 1-555-12-001-6