The Engineering Staff of
TEXAS INSTRUMENTS INCORPORATED
Semiconductor Group

TM 990/100M MICROCOMPUTER USER'S GUIDE

AUGUST 1977

TEXAS INSTRUMENTS
INCORPORATED

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Concluded)

APPENDICES

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Concluded)

LIST OF TABLES

SECTION 1

INTRODUCTION

1.1 GENERAL

The Texas Instruments TM 990/100M is a self-contained microcomputer on a single printed-circuit board. The board's component side is shown in Figure 1-1. It contains features found on computer systems of much larger size including a Central Processing Unit (CPU) with hardware multiply and divide, programmable serial and parallel I/O lines, external interrupts, and a monitor to assist the programmer in program development and execution. Other features include (see Figure 1-2):

- TMS 9900 microprocessor based system: software is compatible with other members of the 990 family.
- 256 x 16 bits of TMS 4042-2 random-access memory (RAM) expandable on board to 512 x 16 bits.
- 1K x 16 bits of TMS 2708 erasable programmable read-only memory (EPROM) expandable on board to 2K x 16 bits. Simple jumper modifications allow substitution of large TMS 2716 EPROM's (16K bits each) for the smaller TMS 2708's (8K bits). Four TMS 2716's allow EPROM expansion to 4K x 16 bits.

NOTE

Three board configurations are available. The characteristics of each are explained in paragraph 1.4.

- Buffered address, data, and control lines for off-board memory and I/O expansion.
- 3 MHz crystal-controlled clock.
- Interfaces to 20 mA current loop or RS-232-C terminals or to twisted-pair multidrop interface (see paragraph 1.4).
- Two programmable interval timers.
- User wire-wrap area surrounded by signal access pins; area adjacent to spare onboard 40-pin connector (P3).
- PROM memory decoders allow easy reassignment of memory map configuration.

1.2 MANUAL ORGANIZATION

Section 1 covers board specifications and characteristics. A glossary in paragraph 1.5 explains terms used throughout the manual.

Section 2 of this manual shows how to install, power up, and operate the TM 990/100 microcomputer with the addition of the following:

- Power supply

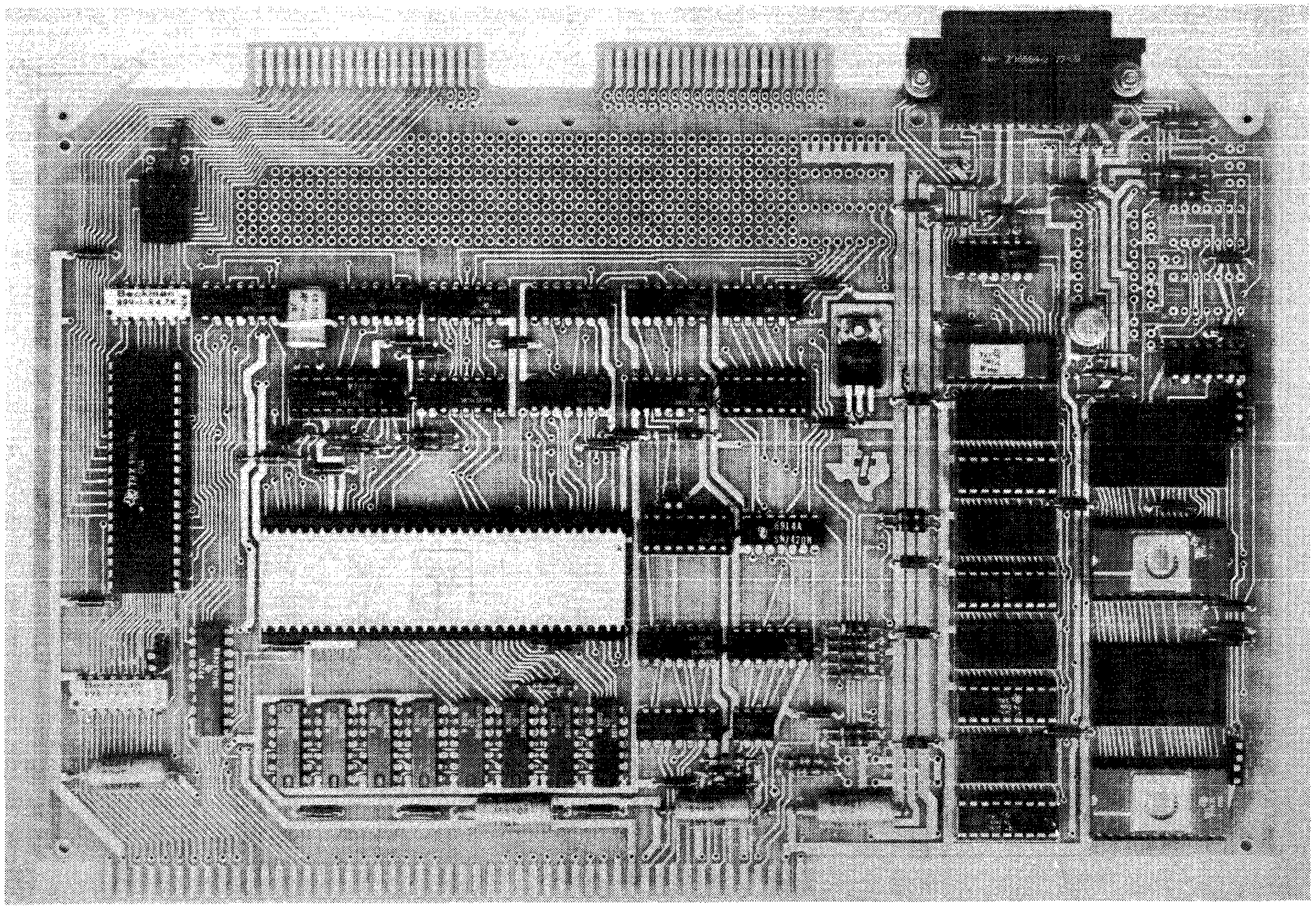


FIGURE 1-1. TM 990/100M MICROCOMPUTER

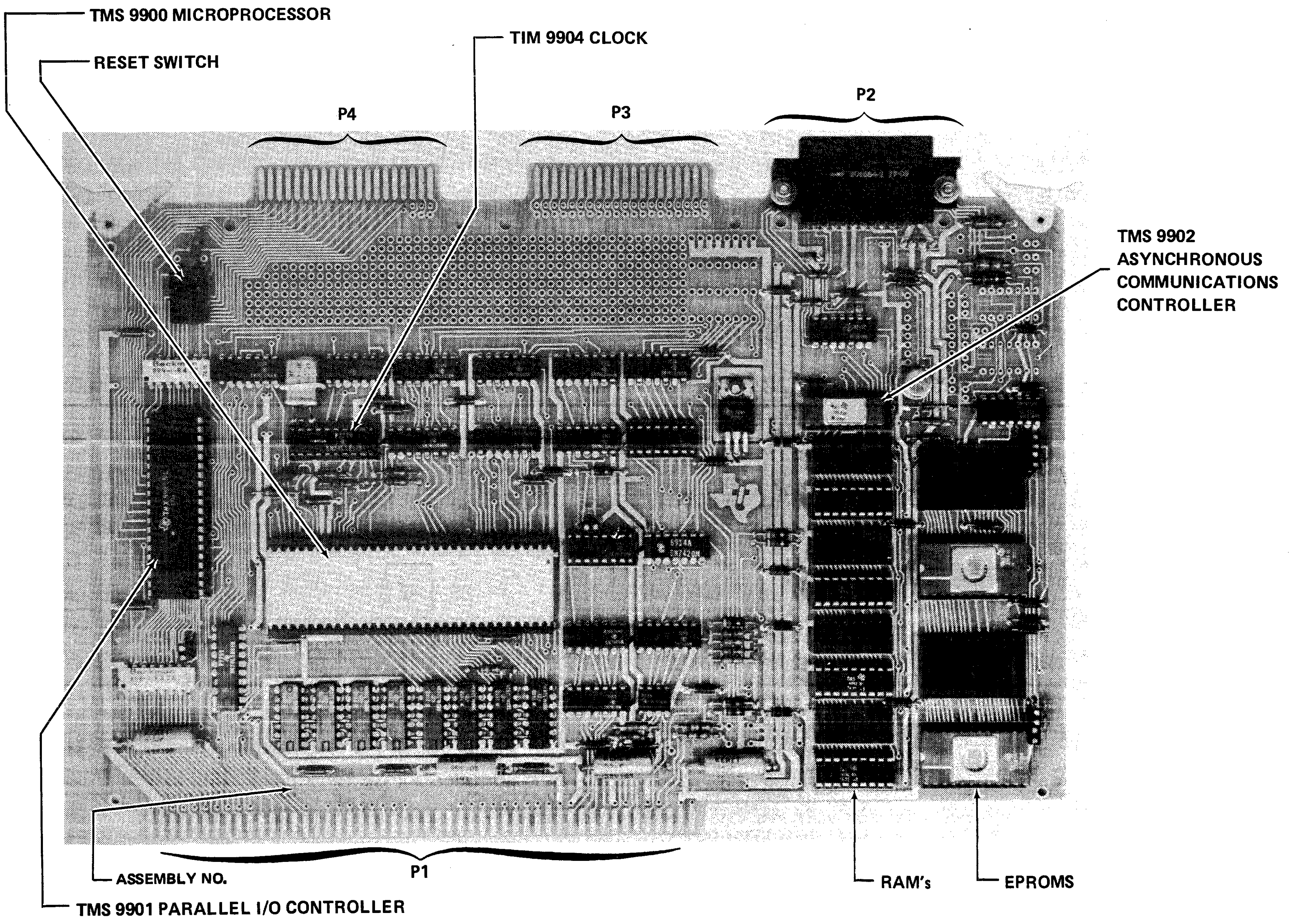


FIGURE 1-2. PRINCIPAL TM 990/100M COMPONENTS

- Data terminal (properly wired and connected)
- Connecting cables

Section 3 explains how you can communicate with the TM 990/100M using the *TIBUG* monitor (on board 999211-0001 only). This versatile monitor, complete with supervisor calls and operator communication commands facilitates the development and execution of software. Section 4 covers programming procedures including the instruction set, interrupts, extended operations (XOPs), context switching, and I/O programming.

Section 5 covers theory of operation with paragraphs keyed to schematics of specific areas of the TM 990/100M board. Section 6 contains application considerations, and Section 7 covers options including a microterminal and a line-by-line (no-label) assembler.

1.3 GENERAL SPECIFICATIONS

Power Consumption:

	+5 V	+12 V	−12 V
256 words RAM, 1K words EPROM	1.2 A	0.2 A	0.1 A
256 words RAM, 2K words EPROM	1.2 A	0.2 A	0.1 A
512 words RAM, 1K words EPROM	1.4 A	0.2 A	0.1 A

Clock rate: 3 MHz

Baud Rates (set by *TIBUG* monitor):
110 baud, 300 baud, 1200 baud, 2400 baud

Memory Size:

RAM (TMS 4042-2's), 256 x 16 bits expandable on-board to 512 x 16 bits

EPROM (TMS 2708's), 1K x 16 bits expandable on-board to 2K x 16 bits

Optional EPROM (TMS 2716's), 2K x 16 bits expandable to 4K x 16 bits

Board Dimensions: See Figure 1-3.

1.4 BOARD CHARACTERISTICS

Different models of the TMS 990/100M microcomputer and identified by different assembly numbers. This number is in the lower left as shown in Figure 1-2. The different aspects of these boards as shipped from the factory are listed in Table 1-1.

1.5 GLOSSARY

The following are definitions of terms used with the TM 990/100M. Applicable areas in this manual are in parentheses.

Absolute Address: The actual memory address in quantity of bytes. Memory addressing is usually represented in hexadecimal from 0000_{16} to $FFFF_{16}$ for the TM 990/100M.

Alphanumeric Character: Letters, numbers, and associated symbols.

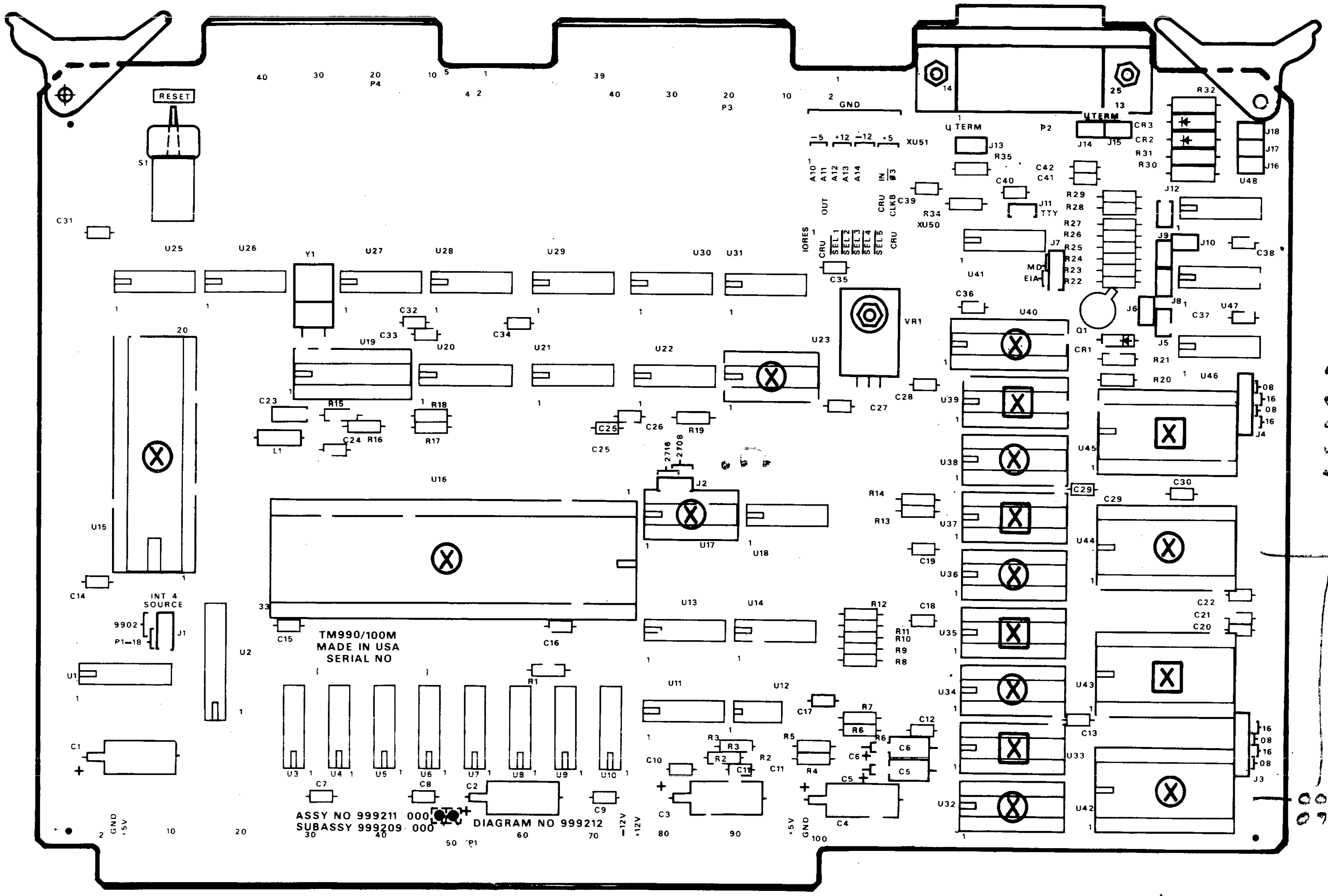


FIGURE 1-3. TM 990/100M BOARD DIMENSIONS (IN INCHES)

TABLE 1-1. BOARD ASSEMBLY CHARACTERISTICS

ASSEMBLY NO.	I/O INTERFACE TYPES	EPROM*	RAM
999211-0001	RS-232-C (EIA) or Current Loop	1K x 16 bits**	256 x 16 bits**
999211-0002	Multidrop or RS-232-C only	1K x 16 bits**	256 x 16 bits**
999211-0003	Multidrop or RS-232-C only	2K x 16 bits***	512 x 16 bits***

*Assembly 999211-0001 EPROM's contain *TIBUG* monitor; assemblies 999211-0002 and -0003 EPROM's are not programmed.
**Two 2708 EPROM's and two 4042 RAM's.
***Four 2708 EPROM's and four 4042 RAM's.

ASCII Code: A seven-bit code used to represent alphanumberic characters and control (Appendix C).

Assembler: Program that interprets assembly language source statements into object code.

Assembly Language: Mnemonics which can be interpreted by an assembler and translated into an object program (paragraph 4.6).

Bit: The smallest part of a word; it has a value of either a 1 or 0.

Breakpoint: Memory address where a program is intentionally halted. This is a program debugging tool.

Byte: Eight bits or half a word.

Carry: A carry occurs when the most-significant bit is carried out in an arithmetic operation (i.e., resultant cannot be contained in only 16 bits), (paragraph 4.3.3.4).

Central Processing Unit (CPU): The "heart" of the computer: responsibilities include instruction access and interpretation, arithmetic functions, I/O memory access. The TMS 9900 is the CPU of the TM 990/100M.

Chad: Dot-like paper particles resulting from the punching of paper tape.

Command Scanner: A given set of instructions in the *TIBUG* monitor which takes the user's input from the terminal and searches a table for the proper code to execute.

Context Switch: Change in program execution environment, includes new program counter (PC) value and new register file.

CRU (Communications Register Unit): The TMS 9900's general purpose, command-driven input/output interface. The CRU provides up to 4096 directly addressable input and output bits (paragraph 4.8).

Effective Address: Memory address resulting from interpretation of an instruction, required for execution of that instruction.

EPROM: See Read Only Memory.

Hexadecimal: Numerical notation in the base 16 (Appendix D).

Immediate Addressing: An immediate or absolute value (16-bits) is part of the instruction (second word of instruction).

Indexed Addressing: The effective address is the sum of the contents of an index register and an absolute (or symbolic) address (paragraph 4.5.3.5).

Indirect Addressing: The effective address is the contents of a register (paragraph 4.5.3.2).

Interrupt: Context switch in which new program counter (PC) and workspace pointer (WP) values are obtained from one of 16 interrupt traps in memory addresses 0000_{16} to $003E_{16}$ (paragraph 4.9).

I/O: The input/output lines are the signals which connect an external device to the data lines of the TMS 9990.

Least Significant Bit (LSB): Bit having the smallest value (smallest power of base 2); represented by the right-most bit.

Link: The process by which two or more object code modules are combined into one, with cross-referenced label address locations being resolved.

Loader: Program that places one or more absolute or relocatable object programs into memory (Appendix G).

Machine Language: Binary code that can be interpreted by the CPU (Table 4-4).

Monitor: A program that assists in the real-time aspects of program execution such as operator command interpretation and supervisor call execution. Sometimes called supervisor (Section 3).

Most Significant Bit (MSB): Bit having the most value; the left-most bit representing the highest power of base 2. This bit is used to show sign with a 1 indicating negative and a 0 indicating positive.

Object Program: The hexadecimal interpretations of source code output by an assembler program. This is the code executed when loaded into memory.

One's Complement: Binary representation of a number in which the negative of the number is the complement or inverse of the positive number (all ones become zeroes, vice versa). The MSB is one for negative numbers and zero for positive. Two representations exist for zero: all ones or all zeroes.

Op Code: Binary operation code interpreted by the CPU to execute the instruction (paragraph 4.5.1).

Overflow: An overflow occurs when the result of an arithmetic operation cannot be represented in two's complement (i.e., in 15 bits plus sign bit), (paragraph 4.3.3.5).

Parity: Means for checking validity of a series of bits, usually a byte. Odd parity means an odd number of one bits; even parity means an even number of one bits. A parity bit is set to make all bytes conform to the selected parity. If the parity is not as anticipated, an error flag can be set by software. The parity jump instruction can be used to determine parity (paragraph 4.3.3.6).

Program Counter (PC): Hardware register that points to the next instruction to be executed or next word to be interpreted (paragraph 4.3.1).

PROM: See Read Only Memory.

Random Access Memory (RAM): Memory that can be written to as well as read from (vs. ROM).

Read Only Memory (ROM): Memory that can only be read from (can't change contents). Some can be programmed (PROM) using a PROM burner. Some PROM's can be erased (EPROM's) by exposure to ultraviolet light.

Source Program: Programs written in menmonics that can be translated into machine language (by an assembler).

Status Register (ST): Hardware register that reflects the outcome of a previous instruction and the current interrupt mask (paragraph 4.3.3).

Supervisor: See Monitor

Utilities: A unique set of instructions used by different parts of the program to perform the same function. In the case of *TIBUG*, the utilities are the I/O XOP's (paragraph 3.3).

Word: Sixteen bits or two bytes.

Workspace Register File: Sixteen words, designated registers 0 to 15, located in RAM for use by the executing program (paragraph 4.4).

Workspace Pointer (WP): Hardware register that contains the memory address of the beginning (register 0) of the workspace register file (paragraph 4.3.2).

1.6 APPLICABLE DOCUMENTS

The following is a list of documents that provide supplementary information for the TM 990/100M user.

- *TMS 9900 Microprocessor Data Manual*
- *TMS 9901 Programmable Systems Interface Data Manual*
- *TMS 9902 Asynchronous Communication Controller (Data Manual)*
- *Model 990 Computer, TMS 9900 Microprocessor Assembly Language Programmer's Guide (P/N 943441-9701)*
- *TM 990/301 Microterminal*
- *TM 990/401 TIBUG Monitor Listing*
- *TM 990/402 Line-By-Line Assembler*
- *TM 990/402L Line-By-Line Assembler Listing*

SECTION 2

INSTALLATION AND OPERATION

2.1 GENERAL

This section explains procedures for unpacking and setting up the TM 990/100M board for operation.

2.2 REQUIRED EQUIPMENT

(1) Volt-ohmmeter

(2) Soldering iron, electrical solder

(3) 24 AWG insulated stranded wire

(4) 18 AWG insulated stranded wire

(5) Connectors

- 100-pin, 0.125 in. C-C, wire-wrap PCB edge connector such as:
  - TI H321150
  - Amphenol 225-804-50
  - Viking 3VH50/9N05
  - Elco 00-6064-100-061-001
- 40-pin, 0.1 in. C-C, wire-wrap PCB edge connector such as:
  - TI H311120
  - Viking 3VH20/IJND5
- 25-pin RS-232 style (plug)
  - ITT DB25P
  - TRW CINCH DB25P

(6) Power Supplies

Voltage	**Reg.**	**Current**
+5 V	±3%	1.3 A
−12 V	±3%	0.2 A
+12 V	±3%	0.1 A

(7) Terminal such as:

- Texas Instruments 743 KSR or 733 KSR/ASR (see Appendix B)
- Teletype Model 3320/5JE (see Appendix A). This current-loop terminal is useable with board assembly 999211-0001 only
- RS-232-C compatible terminal (see Appendix B).

2.3 UNPACKING

Take the TM 990/100M board from its carton and remove the protective wrapping.

Check the board for any abnormalities that could have occurred in shipping. Report any discrepancies to your supplier.

2.4 POWER AND TERMINAL HOOKUP

These procedures assume that user has the following configuration:

- TM 990/100M board with two TMS 2708 erasable, programmable read-only memories (EPROM's).
- Texas Instruments Model 743 KSR terminal.

It is also assumed that jumper configuration is as shipped by the factory (J1, J2, J3, and J4 installed). See Figure 7-2.

For other memory configurations, see paragraph 7.2 for applicable jumper connections.

For other terminals, contact the manufacturer for correct wiring. Hookup to a Teletype model 3320/SJE is explained in Appendix A. Hookup for other RS-232-C compatible terminals is explained in Appendix B.

CAUTION

Be very cautious to avoid applying incorrect voltage levels to the TM 990/100M. Texas Instruments assumes no responsibility for damage caused by improper wiring or voltage application by the user.

2.4.1 POWER SUPPLY HOOKUP

Figure 2-1 shows how to connect voltage to the TM 990/100M through connector P1. Be careful to use the correct pins as numbered on the board; these pin numbers may not correspond to the numbers on the particular edge connector used.

The table in Figure 2-1 shows suggested color coding for the power supply plugs. To prevent incorrect connection, label the top side of the edge connector "TOP" and the bottom "TURN OVER."

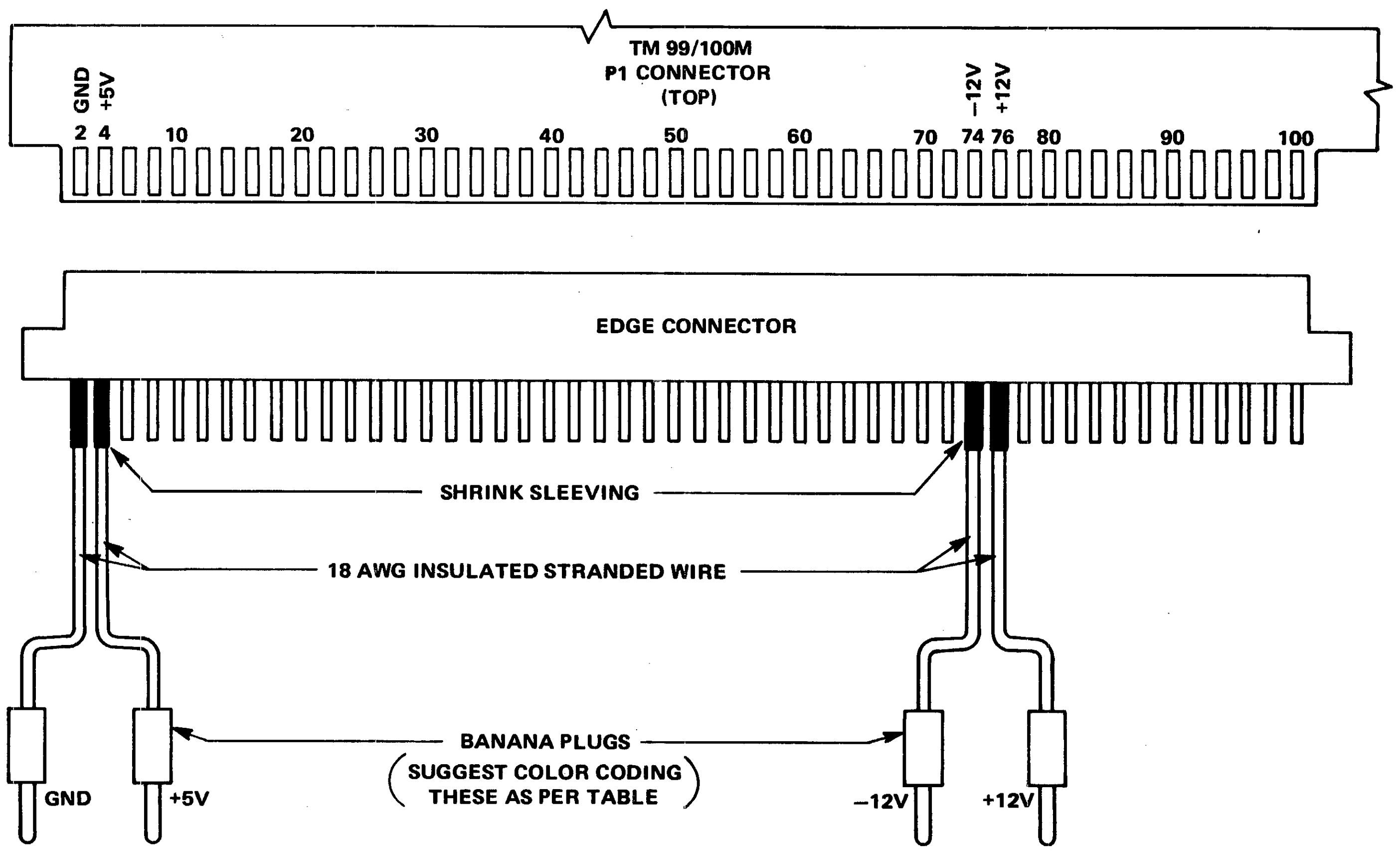
2.4.2 TERMINAL HOOKUP

Figure 2-2 shows how to connect the TM 990/100M to the 743 KSR terminal through connector P2. A DEI5S connector attaches to the terminal; a DB25P connector attaches to P2 on the board. Point-to-point connections between the connectors are shown in the table.

Because this is an RS-232-C type terminal, make sure that jumper J11 is removed and that jumper J7 is in the EIA position (Figure 7-2).

2.5 OPERATION

(1) Verify that all wiring has been correctly connected.



VOLTAGE	P1 PIN*	SUGGESTED PLUG COLORS
+5V	3, 4	RED
+12V	75, 76	BLUE
−12V	73, 74	GREEN
GND	1, 2	BLACK

*ON BOARD, ODD-NUMBERED PADS ARE DIRECTLY BENEATH EVEN-NUMBERED PADS.

A0001417

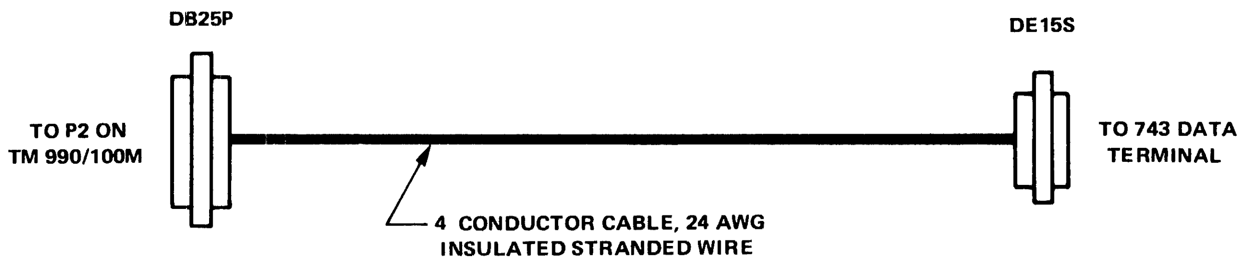
FIGURE 2-1. POWER SUPPLY HOOKUP

CAUTION

Before connecting the power supply to P1, use a volt-ohmmeter to verify that correct voltages are present as shown in Figure 2-2.

(2) Set the 743 KSR data terminal switches to the following:

- LOW SPEED switch to high speed (30 characters per second).
- HALF DUP switch to full duplex.
- ON LINE switch to ON LINE.



CONNECTIONS		
PIN ON DE15S	PIN ON DB25P	SIGNAL
13	2	XMIT
12	3	RECV
11	8	DCD
1	7	GND

A0001418

FIGURE 2-2. 743 KSR TERMINAL HOOKUP

(3) Apply power to board and data terminal.

(4) Press the RESET switch on the board (see Figure 1-2).

(5) Press the "A" key on the terminal.

(6) The *TIBUG* monitor (assembly 999211-0001 only) will be called up and print a message on the terminal. Following the message, a question mark will be printed on a new line. This is a request to input a command to the *TIBUG* command scanner. Commands are explained in detail in Section 3 and assembly language is presented in Section 4.

NOTE

If control is lost during operation, return control back to monitor by repeating steps (4) and (5).

2.6 SAMPLE PROGRAMS

2.6.1 SAMPLE PROGRAM 1

The following is a sample program you can input using the *TIBUG* commands M (paragraph 3.2.8), R (paragraph 3.2.9), and E (paragraph 3.2.4). (*TIBUG* is on assembly 999211-0001 only).

(1) Enter the M command with a hexadecimal address of FD00.

(2) Enter the following values into memory beginning at hexadecimal address FD00 by using the space bar with the M command as described in paragraph 3.2.8.

LOCATION	ENTER VALUE	ASSEMBLY LANGUAGE MNEMONICS
FE00	2FA0	XOP @>FE08, 14
FE02	FE08	
FE04	0420	BLWP @>FFFC
FE06	FFFC	
FE08	4849	TEXT 'HI'
FE0A	0A0D	DATA>0A0D
FE0C	0700	DATA>0700

Exit the M command with a carriage return. The monitor will print a question mark.

(3) Use the R command to set the value 'FD00' into the P register (Program Counter).

(4) Use the E command to execute the program.

(5) The message HI will print on the printer, followed by a line feed, carriage return, and bell. Your terminal printout should look like the following:

```
?M FE00
FE00=2FA0  2FA0
FE02=FE08  FE08
FE04=0460  0460
FE06=0080  0080
FE08=4849  4849
FE0A=0A0D  0A0D
FE0C=0700  0700
?R
W=0B80
P=FE00   FE00
?E HI
```

You can re-execute your program by repeating steps (3) and (4).

2.6.2 SAMPLE PROGRAM 2

Using steps 1 to 5 in pragraph 2.6.1, enter and execute the following program which has been assembled by the optional TM 990/402 Line-By-Line Assembler.

```
FE00 2FA0 XOP @>FE08,14
FE02 FE08
FE04 0460 B @>0080
FE06 0080
FE08 434F $CONGRATULATIONS. YOUR PROGRAM WORKS!
FE0A 4E47
FE0C 5241
```

```
FE0E 5455
FE10 4C41
FE12 5449
FE14 4F4E
FE16 532E
FE18 2059
FE1A 4F55
FE1C 5220
FE1E 5052
FE20 4F47
FE22 5241
FE24 4D20
FE26 574F
FE28 524B
FE2A 5321
FE2C 0707  +>0707
FE2E 0700  +>0700
```

You can re-execute this program by repeating steps (3) and (4) in paragraph 2.6.1.

SECTION 3

TIBUG INTERACTIVE DEBUG MONITOR

3.1 GENERAL

TIBUG is debug monitor which provides an interactive interface between the user and the TM 990/100M. It is supplied by the factory on assembly 999211-0001 only and is available as an option, supplied on two 2708 EPROM's.

TIBUG occupies EPROM memory space from memory address (M.A.) 0080_{16} as shown in Figure 3-1. *TIBUG* uses four workspaces in 40 words of RAM memory. Also in this reserved RAM area are the restart vectors which initialize the monitor following single step execution of instructions.

The *TIBUG* monitor provides seven software routines that accomplish special tasks. These routines, called in user programs by the XOP machine instruction, perform tasks, such as writing characters to a terminal. XOP utility instructions are discussed in detail in paragraph 4.6.9.

All communication with *TIBUG* is through a 20 mA current loop or RS-232-C device. *TIBUG* is initialized as follows:

- Press the RESET pushbutton (Figure 1-2). The monitor is called up through interrupt trap 0.
- Enter the character 'A' at the terminal. *TIBUG* uses this input to measure the width of the start bit and set the TMS 9902 Asynchronous Communication Controller (ACC) to the correct baud rate.
- *TIBUG* prints an initialization message on the terminal. On the next line it prints a question mark indicating that the command scanner is available to interpret terminal inputs.
- Enter one of the commands as explained in paragraph 3.2.

3.2 *TIBUG* COMMANDS

TIBUG commands are listed in Table 3-1.

TABLE 3-1. *TIBUG* COMMANDS

INPUT	RESULTS	PARAGRAPH
B	Execute under Breakpoint	3.2.1
C	CRU Inspect/Change	3.2.2
D	Dump Memory to Cassette/Paper Tape	3.2.3
E	Execute	3.2.4
F	Find Word/Byte in Memory	3.2.5
H	Hex Arithmetic	3.2.6
L	Load Memory from Cassette/Paper Tape	3.2.7
M	Memory Inspect/Change	3.2.8
R	Inspect/Change User WP, PC, and ST Registers	3.2.9
S	Execute in Step Mode	3.2.10
T	1200 Baud Terminal	3.2.11
W	Inspect/Change Current User Workspace	3.2.12

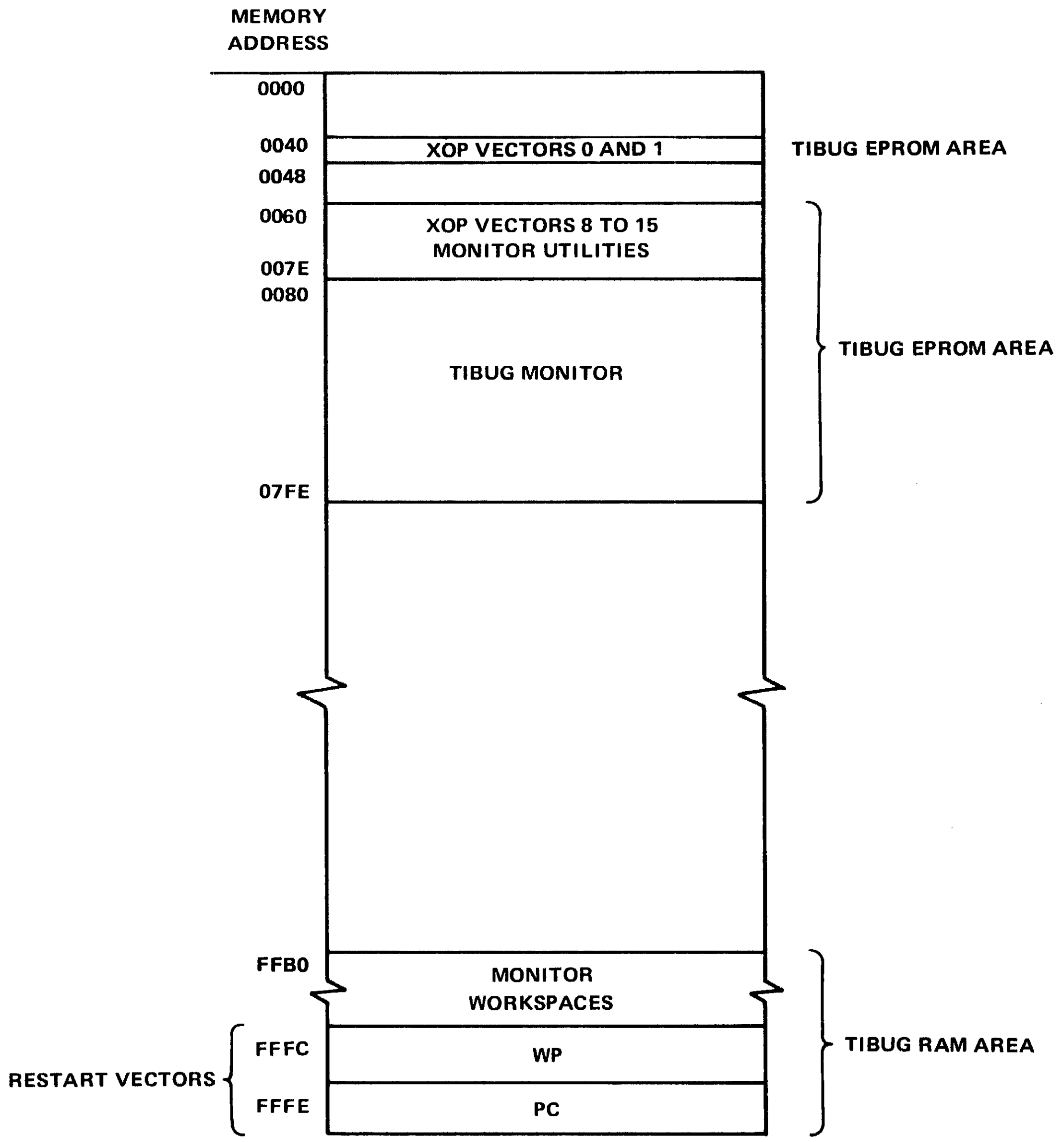


FIGURE 3-1. MEMORY REQUIREMENTS FOR *TIBUG*

Conventions used to define command syntax in this paragraph are listed in Table 3-2.

TABLE 3-2. COMMAND SYNTAX CONVENTIONS

CONVENTION SYMBOL	EXPLANATION
<>	Items to be supplied by the user. The term within the angle brackets is a generic term.
[ ]	Optional Item – May be included or omitted at the user's discretion. Items not included in brackets are required.
{ }	One of several optional items must be chosen.
(CR)	Carriage Return
Λ	Space Bar
LF	Line Feed
R or Rn	Register (n = 0 to 15)
WP	Current User Workspace Pointer contents
PC	Current User Program Counter contents
ST	Current User Status Register contents

NOTE

Except where indicated otherwise, no space is necessary between the parts of these commands. All numeric input is assumed to be hexadecimal; the last four digits input will be the value used. Thus a mistaken numerical input can be corrected by merely making the last four digits the correct value. If fewer than four digits are input, they are right justified.

3.2.1 EXECUTE UNDER BREAKPOINT (B)

3.2.1.1 Syntax

B <address> < (CR) >

3.2.1.2 Description

This command is used to execute instructions from one memory address to another (the stopping address is the breakpoint). When execution is complete, WP, PC, and ST register contents are displayed and control is returned back to the monitor command scanner. Program execution begins at the address in the PC (set by using the R command). Execution terminates at the address specified in the B command, and a banner is output showing the contents of the hardware WP, PC, and ST registers in that order.

The address specified must be in RAM and must be the address of an instruction. The breakpoint is controlled by a software interrupt, XOP 15.

If no address is specified, the B command defaults to an E command, where execution continues with no halting point specified.

EXAMPLE:

```
?B FC06
BP     FFB0     FC06     E400
?
```

3.2.2 CRU INSPECT/CHANGE (C)

3.2.2.1 Syntax

C < base address > { $\overset{\wedge}{,}$ }< count > < (CR) >

3.2.2.2 Description

The Communication Register Unit (CRU) input bits from "base address" to ("base address" + "count" −1) are displayed right justified in a 16-bit hexadecimal representation. "Base address" is a 12-bit value in bits 3 to 14 which is the actual CRU address; this is the same as the contents of register 12 as used by the CRU instructions (paragraph 4.7). Up to 16 CRU bits may be displayed. The corresponding CRU output bits may be altered following input bit display by keying in desired hexadecimal data, right justified. A carriage return following data output forces a return to the command scanner. A minus sign (−) or a space causes the same CRU input bits to be displayed again. Defaults for "base address" and "count" are 0_{16} and 10_{16} respectively.

EXAMPLES:

(1) Examine eight CRU input bits. Base address is 20_{16}.

```
?C 20,8
0020=00FF<—— CARRIAGE RETURN ENTERED
?
```

(2) Set value of eight CRU output bits at base address 20_{16}; new value is 02_{16}.

```
                  ———— CHANGE 00FF TO 0002
?C 20,8
0020=00FF   2<—— 2 FOLLOWED BY CARRIAGE RETURN
?
```

(3) Check changes in CRU input bit 0.

```
?C 0,1
0000=0001   -
0000=0001   -
0000=0001   -  } MINUS SIGN ENTERED
0000=0001   -
0000=00FF   -
0000=0001 <———— CARRIAGE RETURN ENTERED
?
```

(4) Check to see if the TMS 9901 is in the interrupt mode (zero) or clock mode (one);

```
?C 100
0100=FFFE<———— CARRIAGE RETURN ENTERED
```

(5) Check the contents of the clock register on the TMS 9901 (bits 1 to 14).

```
?C 101 14
0101=000E
?
```

3.2.3 DUMP MEMORY TO CASSETTE/PAPER TAPE (D)

3.2.3.1 Syntax

MONITOR PROMPT

D < start address > $\{\overset{\wedge}{,}\}$< stop address > $\{\overset{\wedge}{,}\}$< entry address > $\{\overset{\wedge}{,}\}$IDT = < name > < ∧ >

3.2.3.2 Description

Memory is dumped from "start address" to "stop address." "Entry address" is the address in memory where it is desired to begin program execution. After entering a space or comma following the entry address, the monitor responds with an "IDT=" prompt asking for an input of up to eight characters that will identify the proqram. This program ID will be output when the program is loaded into memory using the *TIBUG* loader, code will be dumped as non-relocatable data in 990 object record format with absolute load ("start address") and entry addresses specified. Object record format is explained in Appendix G.

After entering the D command, the monitor will respond with "READY Y/N" and wait for a Y keyboard entry indicating that the receiving device is ready. This allows the user to verify switch settings, etc., before proceeding with the dump.

3.2.3.3 Dump to Cassette Example

The terminal is assumed to be a Texas Instruments 733 ASR or equivalent. The terminal must have automatic device control (ADC). This means that the terminal recognizes the four tape control characters DC1, DC2, DC3, and DC4.

The following procedure is carried out prior to answering the "READY Y/N" query (Figure 3-2):

(1) Load a cassette in the left (No. 1) transport on the 733 ASR.

(2) Place the transport in the "RECORD" mode.

(3) Rewind the cassette.

(4) Load the cassette. If the cassette does not load, it may be write protected. The write protect hole is on the bottom right side of the cassette (Figure 3-3). Cover it with the tab provided with the cassette. Now repeat steps 1 through 4.

(5) The KEYBOARD, PLAYBACK, RECORD, and PRINTER LOCAL/OFF/LINE switches must be in the LINE position.

(6) Place the TAPE FORMAT switch in the LINE position.

(7) Answer the "READY Y/N" query with a "Y"; the "Y" will not be echoed.

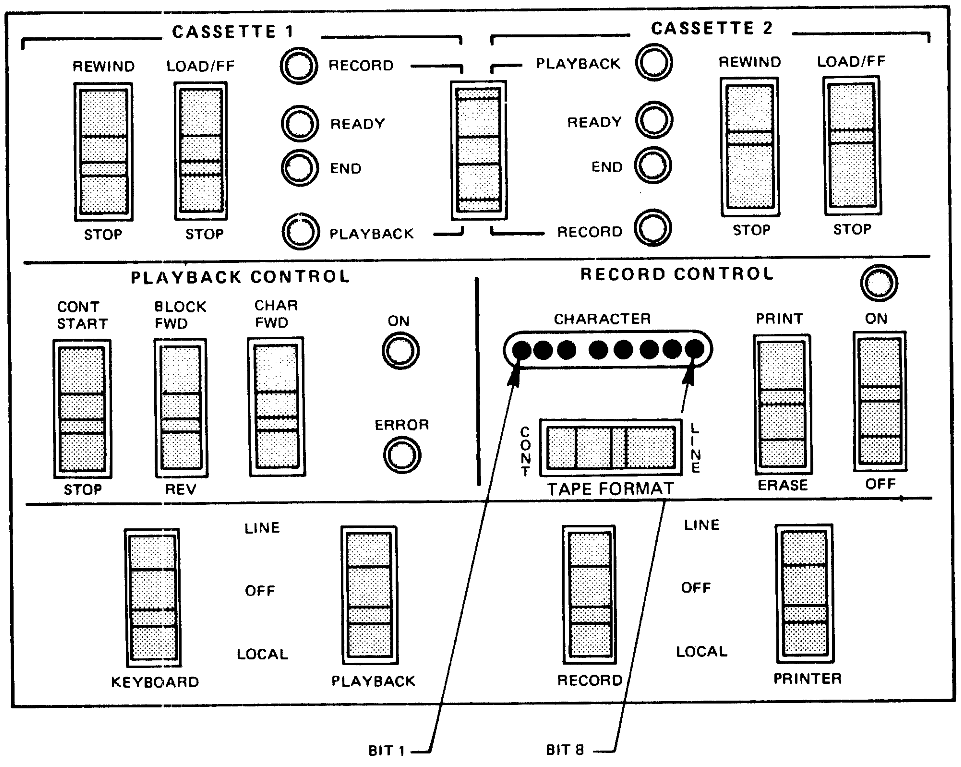
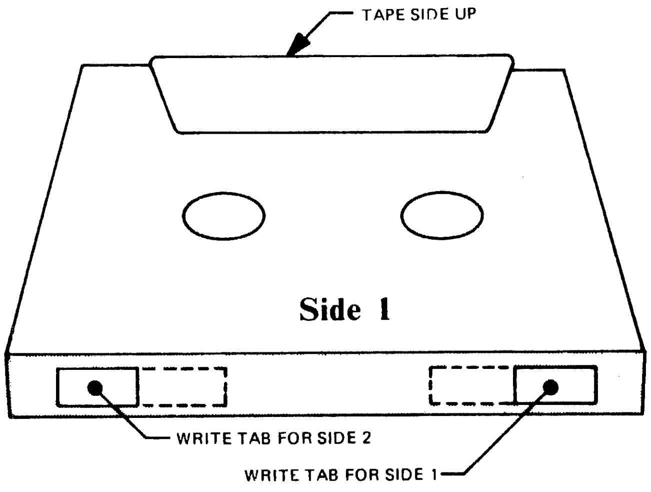


FIGURE 3-2. 733 ASR MODULE ASSEMBLY (UPPER UNIT) SWITCH PANEL



A0001419

FIGURE 3-3. TAPE TABS

3.2.3.4 Dump to Paper Tape

The terminal is assumed to be an ASR 33 teletypewriter. The following steps should be completed carefully to avoid punching stray characters:

(1) Enter the command as described in paragraph 3.2.3.1. Do not answer the "READY Y/N" query yet.

(2) Change the teletype mode from ON LINE to LOCAL.

(3) Turn on the paper tape punch and press the RUBOUT key several times, placing RUBOUTS at the beginning of the tape for correct-reading/program-loading.

(4) Turn off the paper tape punch, and reset the teletype mode to LINE. (This is necessary to prevent punching stray characters).

(5) Turn on the punch and answer the "READY Y/N" query with "Y". The Y will not be echoed.

(6) Punching will begin. Each file is followed by 60 rubout characters. When these characters appear (identified by the constant punching of all holes) the punch must be turned off.

3.2.4 EXECUTE COMMAND (E)

3.2.4.1 Syntax

E

3.2.4.2 Description

The E command causes task execution to begin at current values in the Workspace Pointer and Program Counter.

Example: E

3.2.5 FIND COMMAND (F)

3.2.5.1 Syntax

$\text{F} < \text{start address} > \left\{ \begin{smallmatrix} \wedge \\ , \end{smallmatrix} \right\} < \text{stop address} > \left\{ \begin{smallmatrix} \wedge \\ , \end{smallmatrix} \right\} < \text{value} > \left\{ \begin{smallmatrix} - \\ (\text{CR}) \end{smallmatrix} \right\}$

3.2.5.2 Description

The contents of memory locations from "start address" to "stop address" are compared to "value". The memory addresses whose contents equal "value" are printed out. Default value for start address is 0. The default for "stop address" is 0. The default for "value" is 0.

If the termination character of "value" is a minus sign, the search will be from "start address" to "stop address" for the right byte in "value". If the termination character is a carriage return, the search will be a word mode search.

EXAMPLE:

```
?F 0,20 FFFF ◄──────── CARRIAGE RETURN ENTERED
0006
000C
0012
0016
?F 0 20 FF- ◄───────── MINUS SIGN ENTERED
0006
0007
000C
000D
0012
0013
0016
0017
?
```

3.2.6 HEXADECIMAL ARITHMETIC (H)

3.2.6.1 Syntax

H < number 1 > { $^\Lambda$ }< number 2 >< (CR) >

3.2.6.2 Description

The sum and difference of two hexadecimal numbers are output.

EXAMPLE:

```
?H 200,100 ◄──────────── CARRIAGE RETURN ENTERED
H1+H2=0300   H1-H2=0100
?
```

3.2.7 LOAD MEMORY FROM CASSETTE OR PAPER TAPE (L)

3.2.7.1 Syntax

L < bias >< (CR) >

3.2.7.2 Description

Data in 990 object record format (defined in Appendix G) is loaded from paper tape or cassette into memory. Bias is the relocation bias (starting address in RAM). Its default is 0_{16}. Both relocatable and absolute data may be loaded into memory with the L command. After the data is loaded, the module identifier (see tag 0 in Appendix G) is printed on the next line.

3.2.7.3 Loading From Texas Instruments 733 ASR Terminal Cassette

The 733 ASR must be equipped with automatic device control (ADC). The following procedure is carried out prior to executing the L command:

(1) Insert the cassette in one of the two transports on the 733 ASR (cassette 1 in Figure 3-2).

(2) Place the transport in the playback mode.

(3) Rewind the cassette.

(4) Load the cassette.

(5) Set the KEYBOARD, PLAYBACK, RECORD, and PRINTER LOCAL/LINE switches to LINE.

(6) Set the TAPE FORMAT switch to LINE.

(7) Loading will be at 1200 baud; thus the T command must be entered (paragraph 3.2.11).

Execute the L command.

3.2.7.4 Loading From Paper Tape (ASR33 Teletype)

Prior to executing the L command, place the paper tape in the reader and position the tape so the reader mechanism is in the null field prior to the file to be loaded. Enter the load command. If the ASR33 has ADC (automatic device control), the reader will begin to read from the tape. If the ASR33 does not have ADC, turn on the reader, and loading will begin.

Each file is terminated with 60 rubouts. When the reader reaches this area of the tape, turn it off. The loader will then pass control to the command scanner.

The user program counter (P) is loaded with the entry address if a 1 tag or a 2 tag is found on the tape.

EXAMPLE:

```
?L   0000<----------- CARRIAGE RETURN ENTERED
PROGRAM <----------- PROGRAM ID FROM TAPE
?
```

3.2.8 MEMORY INSPECT/CHANGE, MEMORY DUMP (M)

3.2.8.1 Syntax

- Memory Inspect/Change Syntax

  M < address > < (CR) >

- Memory Dump Syntax

  M < start address > $\left\{ \begin{matrix} \wedge \\ , \end{matrix} \right\}$< stop address > < (CR) >

3.2.8.2 Description

Memory inspect/change "opens" a memory location, displays it, and gives the option of changing the data in the location. The termination character causes the following:

- If a carriage return, control is returned to the command scanner.

- If a space, the next memory location is opened and displayed.
- If a minus sign, the previous memory location is opened and displayed.

If a hexadecimal value is entered before the termination character, the displayed memory location is updated to the value entered.

Memory dump directs a display of memory contents from "start address" to "stop address". Each line of output consists of the address of the first data word output followed by eight data words. Memory dump can be terminated at any time by typing any character on the keyboard.

EXAMPLES:

(1)

```
?M FE00←————————— CARRIAGE RETURN ENTERED
FE00=FF0F
FE02=0012  FFFF←—— NEW CONTENTS ENTERED
FE04=0311  -←———— MINUS SIGN ENTERED
FE02=FFFF ←—————— NEW CONTENTS
FE04=0311
FE06=0032  EEAA←—— CARRAGE RETURN ENTERED
?
```

(2)

```
?M 20 30
0020=0020  0030  0000  0005   0030  0D00  0000  0024
0030=0001
?
```

3.2.9 INSPECT/CHANGE USER WP, PC, AND ST REGISTERS (R)

3.2.9.1 Syntax

R < (CR) >

3.2.9.2 Description

The user workspace pointer (WP), program counter (PC), and status register (ST) are inspected and changed with the R command. The output letters WP, PC, and ST identify the values of the three principal hardware registers passed to the TMS 9900 microprocessor when a B, E, or S command is entered. WP points to the workspace register area, PC points to the next instruction to be executed (Program Counter), and ST is the Status Register contents.

The termination character causes the following:

- A carriage return causes control to return to the command scanner.
- A space causes the next register to be opened.

Order of display is W, P, S.

EXAMPLES:

(1)

```
?R
W=0020  100 <—— SPACE ENTERED
P=0000  200 <——CARRIAGE RETURN ENTERED
?
```

(2)

```
?R
W=0100<———————— SPACE ENTERED
P=0200<———————/
S=0000<———————— SPACE OR CARRIAGE RETURN ENTERED
?
```

3.2.10 EXECUTE IN SINGLE STEP MODE (S)

3.2.10.1 Syntax

S

3.2.10.2 Description.

Each time the S command is entered, a single instruction is executed at the address in the Program Counter, then the contents of the Program Counter, Workspace Pointer, and Status Register (after execution) are printed out. Successive instructions can be executed by repeated S commands. Essentially, this command executes one instruction then returns control to the monitor.

EXAMPLE:

```
?R
W=FFC6       }    /— SPACES ENTERED
P=FE10  FE00 }<——         ———————————— WORKSPACE POINTER
S=260A       }        /           ———— PROGRAM COUNTER
?S        FFC6<——  FE02<——     860A<— STATUS REGISTER
?S        FFC6     FE04        860A
?S        FFC6     FE08        860A
?S        FFC6     FE0C        860A
?
```

NOTE

Incorrect results are obtained when the S instruction causes execution of an XOP instruction (see paragraph 4.6.9) in a user program. To avoid these problems the B command should be used to execute any XOP's in a program (rather than the S command).

3.2.11 TI 733 ASR BAUD RATE (T)

3.2.11.1 Syntax

T

3.2.11.2 Description

The T command is used to alert *TIBUG* that the terminal being used is a 1200 baud terminal which is not a Texas Instrument's 733 ASR (e.g., a 1200 baud CRT). To revoke the T command, enter it again.

3.2.11.3 Use

T is used only when operating with a true 1200 baud peripheral device. Note that T is never used when operating at other baud rates.

In *TIBUG* the baud rate is set by measuring the width of the character 'A' input from a terminal. When an 'A' of 1200 baud width is measured, *TIBUG* is set up to automatically insert three nulls for every character output to the terminal. These nulls are inserted to allow correct operation of the TM 990/100M with Texas Instruments 733ASR data terminals.

3.2.12 INSPECT/CHANGE USER WORKSPACE (W)

3.2.12.1 Syntax

W [REGISTER NUMBER] < (CR) >

3.2.12.2 Description

The W command is used to display the contents of all workspace registers or display one register at a time while allowing the user to change the register contents. The workspace begins at the address given by the Workspace Pointer.

The W command, followed by a carriage return, causes the contents of the entire workspace to be printed. Control is then passed to the command scanner.

The W command followed by a register number in hexadecimal and a carriage return causes the display of the specified register's contents. The user may then enter a new value into the register by entering a hexadecimal value. The following are termination characters whether or not a new value is entered:

- A space causes display of the next register.
- A minus sign causes display of the previous register.
- A carriage return gives control to the command scanner.

EXAMPLES:

(1)

```
?W ◄──────────────── CARRIAGE RETURN ENTERED
R0=F942  R1=0084  R2=FA2A  R3=0020  R4=FB5E  R5=0098  R6=1300  R7=0084
R8=FAA0  R9=3600  RA=0EA6  RB=0000  RC=01C0  RD=0084  RE=FA30  RF=C600
?
```

(2)

```
?W 2 ◄──────────── CARRIAGE RETURN ENTERED
R2=0284  3456 ⎫
R3=001B  100  ⎪
R4=1608       ⎬ SPACE ENTERED
R5=0460  800F ⎭
R6=F800  0 ◄────── CARRIAGE RETURN ENTERED
```

3.3 USER ACCESSIBLE UTILITIES

TIBUG contains seven utility subroutines that perform I/O functions as listed in Table 3-3. These subroutines are called through the XOP (extended operation) assembly language instruction. This instruction is covered in detail in paragraph 4.6.9.

TABLE 3-3. USER ACCESSIBLE UTILITIES

XOP	FUNCTION	PARAGRAPH
8	Write 1 Hexadecimal Character to Terminal	3.3.1
9	Read Hexadecimal Word from Terminal	3.3.2
10	Write 4 Hexadecimal Characters to Terminal	3.3.3
11	Echo Character	3.3.4
12	Write 1 Character to Terminal	3.3.5
13	Read 1 Character from Terminal	3.3.6
14	Write Message to Terminal	3.3.7

NOTE
All characters are in ASCII code.

NOTE
Most of the XOP format examples herein use a register for the source address; however, all XOP's can also use a symbolic memory address or any of the addressing forms available for the XOP instruction.

3.3.1 WRITE ONE HEXADECIMAL CHARACTER TO TERMINAL (XOP 8)

Format: XOP Rn,8

The least significant four bits of user register Rn are converted to their ASCII coded hexadecimal equivalent (0 to F) and output on the terminal. Control returns to the instruction following the extended operation.

EXAMPLE:

Assume user register 5 contains $203C_{16}$. The assembly language (A.L.) and machine language (M.L.) values are shown below.

A.L. **XOP** **R5,8** **SEND 4 LSB'S OF R5 TO TERMINAL**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
M.L.	0	0	1	0	1	1	1	0	0	0	0	0	0	1	0	1	**>2E05**

Terminal Output: C

3.3.2 READ HEXADECIMAL WORD FROM TERMINAL (XOP 9)

```
Format:   XOP     Rn,9
          DATA    NULL         ADDRESS OF CONTINUED EXECUTION IF
                               NULL IS ENTERED
          DATA    ERROR        ADDRESS OF CONTINUED EXECUTION IF
                               NON-HEX NO. ENTERED
          (NEXT INSTRUCTION)   EXECUTION CONTINUED HERE IF VALID HEX
                               NUMBER AND TERMINATOR ENTERED
```

Binary representation of the last four hexadecimal digits input from the terminal is accumulated in user register Rn. The termination character is returned in register Rn+1. Valid termination characters are space, minus, comma, and a carriage return. Return to the calling task is as follows:

- If a valid termination character is the only input, return is to the memory address contained in the next word following the XOP instruction (NULL above).
- If a non-hexadecimal character or an invalid termination character is input, control returns to the memory address contained in the second word following the XOP instruction (ERROR above).
- If a hexadecimal string followed by a valid termination character is input, control returns to the word following the DATA ERROR statement above.

EXAMPLE:

```
A.L.    XOP     R6,9     READ HEXADECIMAL WORD INTO R6
        DATA    >FFC0    RETURN ADDRESS, IF NO NUMBER
        DATA    >FFC6    RETURN ADDRESS, IF ERROR
```

M.L. / M.A.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
FFB0	0	0	1	0	1	1	1	0	0	1	0	0	0	1	1	0	>2E46
FFB2	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	>FFC0
FFB4	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	0	>FFC6

If the valid hexadecimal character string 12C is input from the terminal followed by a carriage return, control returns to memory address (M.A.) $FFB6_{16}$ with register 6 containing $012C_{16}$ and register 7 containing $000D_{16}$.

If the hexadecimal character string 12C is input from the terminal followed by an ASCII plus (+) sign, control returns to location $FFC6_{16}$. Registers 6 and 7 are returned to the calling program without being altered. "+" is an invalid termination character.

If the only input from the terminal is a carriage return, register 6 is returned unaltered while register 7 contains $000D_{16}$. Control is returned to address $FFC0_{16}$.

3.3.3 WRITE FOUR HEXADECIMAL CHARACTERS TO TERMINAL (XOP 10)

```
Format:   XOP   Rn,10
```

The four-digit hexadecimal representation of the contents of user register Rn is output to the terminal. Control returns to the instruction following the XOP call.

EXAMPLE:

Assume register 1 contains $2C46_{16}$.

A.L. XOP R1,10 WRITE HEX NUMBER

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
M.L.	0	0	1	0	1	1	1	0	1	0	0	0	0	0	0	1	>2E81

Terminal Output: 2C46

3.3.4 ECHO CHARACTER (XOP 11)

Format: XOP Rn, 11

This is a combination of XOP's 13 (read character) and 14 (write character). A character in ASCII code is read from the terminal, placed in the left byte of Rn, then written (echoed back) to the terminal. Control returns to the instruction following the XOP after a character is read and written. By using a code to determine a character string termination, a series of characters can be echoed and stored at a particular address:

```
CLR     R2          CLEAR R2
LI      R1,>FE00    SET STORAGE ADDRESS
XOP     R2, 11      ECHO USING R2
CI      R2,>0D00    WAS CHARACTER A CR?
JEQ     $+6         YES, EXIT ROUTINE
MOVB    R2,*R1+     NO, MOVE CHAR TO STORAGE
JMP     $-10        REPEAT XOP
```

NOTE
The parity bit must be reset so that >OD = CR.

3.3.5 WRITE ONE CHARACTER TO TERMINAL (XOP 12)

Format: XOP Rn,12

The ASCII character in the left byte of user register Rn is output to the terminal. The right byte of Rn is ignored. Control is returned to the instruction following the call.

3.3.6 READ ONE CHARACTER FROM TERMINAL (XOP 13)

Format: XOP Rn,13

The ASCII representation of the character input from the terminal is placed in the left byte of user register Rn. The right byte of register Rn is zeroed. When this utility is called, control is returned to the instruction following the call only after a character is input.

3.3.7 WRITE MESSAGE TO TERMINAL (XOP 14)

Format: XOP @MESSAGE,14

MESSAGE is the symbolic address of the first character of the ASCII character string to be output. The string must be terminated with a byte containing binary zeroes. After the character string is output, control is returned to the first instruction following the call.

Assuming the following program:

```
MEMORY
ADDRESS         OP CODE          A.L. MNEMONIC
 (Hex)           (Hex)

 FE00            2FA0             XOP @>FEE0,14
 FE02            FEE0
 FE04
  .               .
  .               .
  .               .
 FEE0            5445             TEXT 'TEST'
 FEE2            5354
 FEE4            00               BYTE 0
```

During the execution of this XOP, the character string 'TEST' is output on the terminal and control is then returned to the instruction at location $FE04_{16}$. TEXT is an assembler directive to transcribe characters into ASCII code.

3.4 *TIBUG* ERROR MESSAGES

Several error messages have been included in the *TIBUG* monitor to alert the user to incorrect operation. In the event of an error, the word 'ERROR' is output followed by a single digit representing the error number.

Table 3-4 outlines the possible error conditions

TABLE 3-4. *TIBUG* ERROR MESSAGES

ERROR	CONDITION
0	Invalid tag detected by the loader.
1	Checksum error detected by the loader.
2	Invalid termination character detected.
3	Null input field detected by the dump routine.
4	Invalid command entered.

In the event of errors 0 or 1, the program load process is terminated. If the program is being input from a 733 ASR, possible causes of the errors are a faulty cassette tape or dirty read heads in the tape transport. If the terminal device is an ASR33, chad may be caught in a punched hole in the paper tape. In either case repeat the load procedure.

In the event of error 2, the command is terminated. Reissue the command and parameters with a valid termination character.

Error 3 is the result of the user inputting a null field for either the start address, stop address, or the entry address to the dump routine. It also occurs if the ending address is less than the beginning address. The dump command is terminated. To correct the error, reissue the dump command and input all necessary parameters.

SECTION 4

PROGRAMMING THE TM 990/100M

4.1 GENERAL

This section covers the instruction set used with the TM 990/100M including assembly language and machine language. This instruction set is compatible with other members of the 990 family.

Other topics include:

- Hardware and software registers (paragraphs 4.3 and 4.4).
- CRU addressing (paragraph 4.7)
- Interrupts (paragraph 4.10)

The TM 990/100M microcomputer is designed for use by a variety of users with varying technical backgrounds and available support equipment. Because a TM 990/100M user has the capability of writing his programs in machine language and entering them into memory using the *TIBUG* monitor, emphasis is on binary/hexadecimal representations of assembly language statements. The assembly language described herein can be assembled on a 990 family assembler. If an assembler is used, this section assumes that the user will be aware of all prerequisites for using the particular assembler.

It is also presumed that all users learning this instruction set have a working knowledge in:

- ASCII coded character set (described in Appendix C).
- Decimal/hexadecimal, binary number system (described in Appendix D).

Further information on the 990 assembly language is provided in the *Model 990 Computer/TMS 9900 Microprocessor Assembly Language Programmer's Guide* (P/N 943441-9701).

4.2 USER MEMORY

Figure 4-1 shows the user RAM space in memory available for execution of user programs. Note that the memory address value is the number of bytes beginning at 0000; thus, all word addresses are even values from 0000 to FFFE_{16}.

Programs in EPROM's can be read by the processor and executed; however, EPROM memory cannot be modified (written to). Therefore, workspace register areas are in RAM where their values can be modified. Restart vectors and *TIBUG* workspaces utilize the last 40 words of RAM memory space as shown in Figure 4-1.

4.3 HARDWARE REGISTERS

The TM 990/100M uses three major hardware registers in executing the instruction set: Program Counter (PC), Workspace Pointer (WP), and Status Register (ST).

4.3.1 PROGRAM COUNTER (PC)

This register contains the memory address of the next instruction to be executed. After an instruction image is read in for interpretation by the processor, the PC is incremented by two so that it "points" to the next sequential memory word.

4.3.2 WORKSPACE POINTER (WP)

This register contains the memory address of the register file currently being used by the program under execution. This workspace consists of 16 contiguous memory words designated registers 0 to 15. The WP points to register 0. Paragraph 4.4 explains a workspace in detail.

4.3.3 STATUS REGISTER (ST)

The Status Register contains relevant information on preceding instructions and current interrupt level. Included are:

- Results of logical and two's complement comparisons (many instructions automatically compare the results to zero).

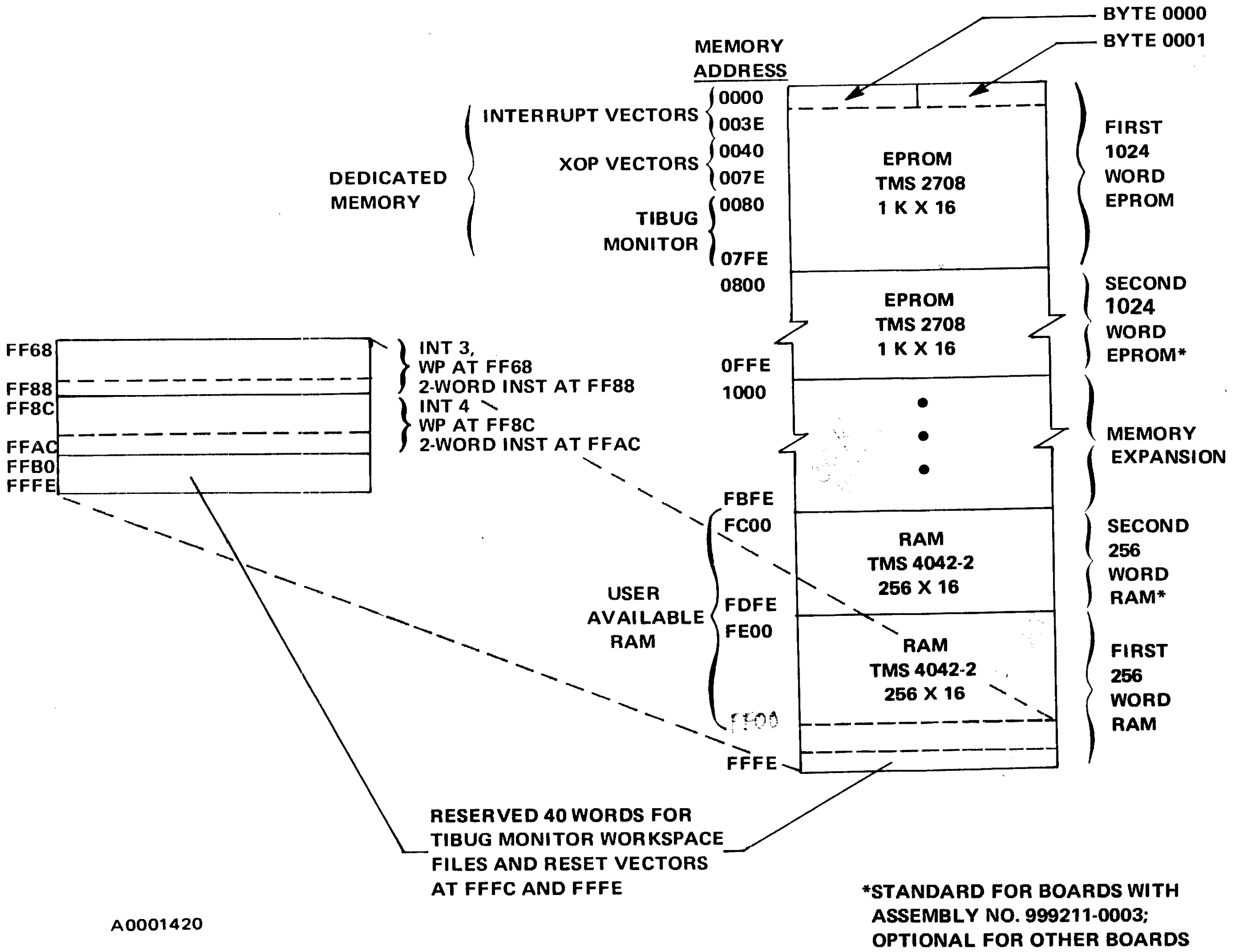
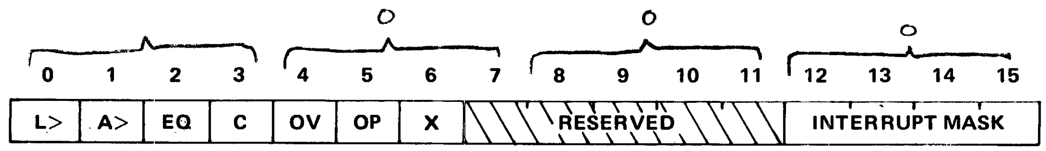


FIGURE 4-1. MEMORY MAP

- Carry and overflow.
- Odd parity found (byte instructions only).
- XOP being executed.
- Lowest priority interrupt level that will be currently recognized by the processor.

The Status Register is shown in Figure 4-2.



L>	LOGICALLY GREATER THAN	OV	OVERFLOW
A>	ARITHMETICALLY GREATER THAN	OP	ODD PARITY
EQ	EQUAL	X	XOP BEING EXECUTED
C	CARRY		

6

A0001421

FIGURE 4-2. STATUS REGISTER

4.3.3.1 Logical Greater Than

This bit contains the result of a comparison of words or bytes as unsigned binary numbers. The most significant bit (MSB) of words being logically compared represents 2^{15} (32,768), and the MSB of bytes being logically compared represents 2^7 (128).

4.3.3.2 Arithmetic Greater Than

The arithmetic greater than bit contains the result of a comparison of words or bytes as two's complement numbers. In this comparison, the MSB of words or bytes being compared represents the sign of the number, zero for positive, or one for negative.

4.3.3.3 Equal

The equal bit is set when the words or bytes being compared are equal.

4.3.3.4 Carry

The carry bit is set by a carry out of the MSB of a word or byte (sign bit) during arithmetic operations. The carry bit is used by the shift operations to store the value of the last bit shifted out of the workspace register being shifted.

4.3.3.5 Overflow

The overflow bit is set when the result of an arithmetic operation is too large or too small to be correctly represented in two's complement (arithmetic) representation. In addition operations, overflow is set when the MSB's of the operands are equal and the MSB of the result is not equal to the MSB of the destination operand. In subtraction operations, the overflow bit is set when the MSB's of the operands are not equal, and the MSB of the result is not equal to the MSB of the destination operand. For a divide operation, the overflow bit is set when the most significant sixteen bits of the dividend (a 32-bit value) are greater than or equal to the divisor. For an arithmetic left shift, the overflow bit is set if the MSB of the workspace register being shifted changes value. For the absolute value and negate instructions, the overflow bit is set when the source operand is the maximum negative value, 8000_{16}.

4.3.3.6 Odd Parity

The odd parity bit is set in byte operations when the parity of the result is odd, and is reset when the parity is even. The parity of a byte is odd when the number of bits having a value of one is odd; when the number of bits having a value of one is even, the parity of the byte is even.

4.3.3.7 Extended Operation

The extended operation bit of the Status Register is set to one when a software implemented extended operation (XOP) is initiated.

4.3.3.8 Status Bit Summary

Table 4-1 lists the instruction set and the status bits affected by each instruction.

4.4 SOFTWARE REGISTERS

Registers used by programs are contained in memory. This speeds up context-switch time because the content of only one register (WP hardware register) needs to be saved instead of the entire register file. The WP, PC, and ST register contents are saved in a context switch.

A workspace is a contiguous 16 word area; its memory location can be designated by placing a value in the WP register through software or a keyboard monitor command. A program can use one or several workspace areas, depending upon register requirements.

More than three-fourths of the instructions can address the workspace register file; all shift instructions and most immediate operand instructions use workspace registers exclusively.

Figure 4-3 is an example of a workspace file in high-order memory (RAM). A workspace in ROM would be ineffective since it could not be written into. Note that several registers are used by particular instructions.

TABLE 4-1. STATUS BITS AFFECTED BY INSTRUCTIONS

MNEMONIC	L>	A>	EQ	C	OV	OP	X
A	X	X	X	X	X	—	—
AB	X	X	X	X	X	X	—
ABS	X	X	X	X	X	—	—
AI	X	X	X	X	X	—	—
ANDI	X	X	X	—	—	—	—
B	—	—	—	—	—	—	—
BL	—	—	—	—	—	—	—
BLWP	—	—	—	—	—	—	—
C	X	X	X	—	—	—	—
CB	X	X	X	—	—	X	—
CI	X	X	X	—	—	—	—
CLR	—	—	—	—	—	—	—
COC	—	—	X	—	—	—	—
CZC	—	—	X	—	—	—	—
DEC	X	X	X	X	X	—	—
DECT	X	X	X	X	X	—	—
DIV	—	—	—	—	X	—	—
IDLE	—	—	—	—	—	—	—
INC	X	X	X	X	X	—	—
INCT	X	X	X	X	X	—	—
INV	X	X	X	—	—	—	—
JEQ	—	—	—	—	—	—	—
JGT	—	—	—	—	—	—	—
JH	—	—	—	—	—	—	—
JHE	—	—	—	—	—	—	—
JL	—	—	—	—	—	—	—
JLE	—	—	—	—	—	—	—
JLT	—	—	—	—	—	—	—
JMP	—	—	—	—	—	—	—
JNC	—	—	—	—	—	—	—
JNE	—	—	—	—	—	—	—
JNO	—	—	—	—	—	—	—
JOC	—	—	—	—	—	—	—
JOP	—	—	—	—	—	—	—
LDCR	X	X	X	—	—	1	—
LI	X	X	X	—	—	—	—
LIMI	—	—	—	—	—	—	—
LREX	—	—	—	—	—	—	—
LWPI	—	—	—	—	—	—	—
MOV	X	X	X	—	—	—	—
MOVB	X	X	X	—	—	X	—
MPY	—	—	—	—	—	—	—
NEG	X	X	X	X	X	—	—
ORI	X	X	X	—	—	—	—
RSET	—	—	—	—	—	—	—
RTWP	X	X	X	X	X	X	X
S	X	X	X	X	X	—	—
SB	X	X	X	X	X	X	—
SBO	—	—	—	—	—	—	—
SBZ	—	—	—	—	—	—	—
SETO	—	—	—	—	—	—	—
SLA	X	X	X	X	X	—	—
SOC	X	X	X	—	—	—	—
SOCB	X	X	X	—	—	X	—
SRA	X	X	X	X	—	—	—
SRC	X	X	X	X	—	—	—
SRL	X	X	X	X	—	—	—
STCR	X	X	X	—	—	1	—
STST	—	—	—	—	—	—	—
STWP	—	—	—	—	—	—	—
SWPB	—	—	—	—	—	—	—
SZC	X	X	X	—	—	—	—
SZCB	X	X	X	—	—	X	—
TB	—	—	X	—	—	—	—
X	2	2	2	2	2	2	2
XOP	2	2	2	2	2	2	2
XOR	X	X	X	—	—	—	—

NOTES

1. When an LDCR or STCR instruction transfers eight bits or less, the OP bit is set or reset as in byte instructions. Otherwise these instructions do not affect the OP bit.
2. The X instruction does not affect any status bit; the instruction executed by the X instruction sets status bits normally for that instruction. When an XOP instruction is implemented by software, the XOP bit is set, and the subroutine sets status bits normally.

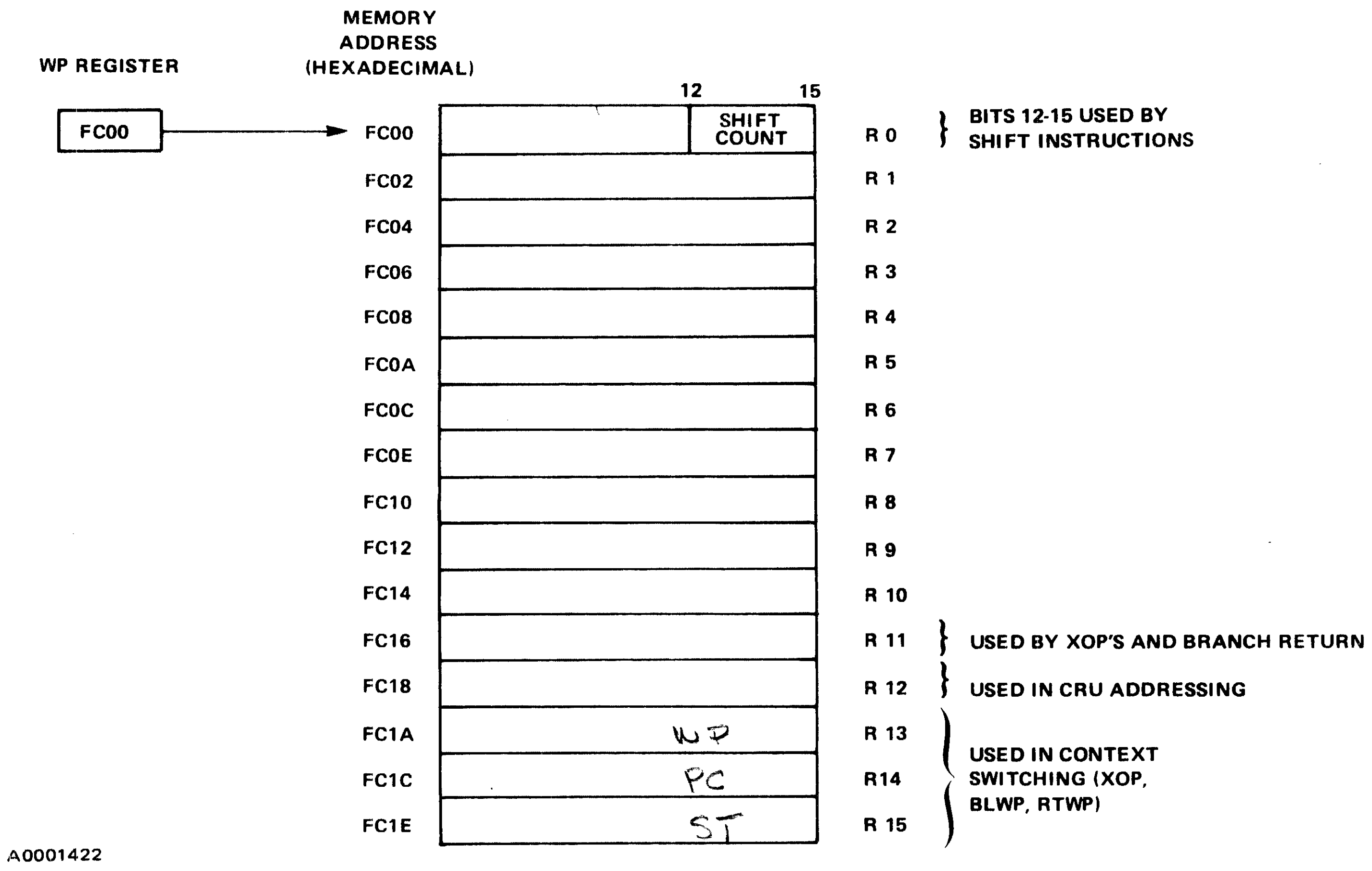


FIGURE 4-3. WORKSPACE EXAMPLE

4.5 INSTRUCTION FORMATS AND ADDRESSING MODES

The instructions used by the TM 990/100M are contained in 16-bit memory words and require one, two, or three words for full definition. The first word (or the single word) of an instruction will describe the purpose of the instruction while the succeeding one or two words will be numbers that are referenced by the initial instruction word. A word describing an instruction is interpreted by the Central Processing Unit (CPU) by decoding the various fields within the 16 bits. These fields are shown in Figure 4-4 for the 9900 instruction set which is also categorized into nine instruction formats as shown in the figure.

In order to construct instructions in machine language, the programmer must have a knowledge of the fields and formats of the instructions. This knowledge is often very important in debugging operations because it allows the programmer to change bits within an instruction in order to solve an execution problem.

The fields within an instruction word contain the following information (see Figure 4-4):

- Op code which identifies the desired operation to be accomplished when this instruction is executed.
- B code which identifies whether the instruction will affect a full 16-bit word in memory or an 8-bit byte. A one indicates a byte will be addressed, while a zero indicates a word will be addressed.

FORMAT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	GENERAL USE
1	OP CODE			B	T_D		DR				T_S		SR				ARITHMETIC
2	OP CODE								SIGNED DISPLACEMENT								JUMP
3	OP CODE						WR				T_S		SR				LOGICAL
4	OP CODE						C				T_S		SR				CRU
5	OP CODE								C				R				SHIFT
6	OP CODE										T_S		SR				PROGRAM
7	OP CODE											NOT USED					CONTROL
8	OP CODE											N	R				IMMEDIATE
9	OP CODE						DR				T_S		SR				MPY, DIV, XOP

OP CODE	OPERATION CODE
B	BYTE INDICATOR (1=BYTE)
T_D	DESTINATION ADDRESS TYPE*
DR	DESTINATION REGISTER
T_S	SOURCE ADDRESS TYPE*
SR	SOURCE REGISTER
C	CRU TRANSFER COUNT OR SHIFT COUNT
R	REGISTER
N	NOT USED

*T_D OR T_S	ADDRESS MODE TYPE
00	DIRECT REGISTER
01	INDIRECT REGISTER
10	PROGRAM COUNTER RELATIVE, NOT INDEXED (SR OR DR = 0)
	PROGRAM COUNTER RELATIVE + INDEX REGISTER (SR OR DR>0)
11	INDIRECT REGISTER, AUTOINCREMENT REGISTER

A0001423

FIGURE 4-4. TM 990/100M INSTRUCTION FORMATS

- T fields identified by T_D for the destination T field and T_S for the source T field. The T field is a two-bit code which identifies which of five different addressing modes will be used (direct register, indirect register, memory address, memory address indexed, and indirect register autoincremented). These modes are described in detail in paragraphs 4.5.1 through 4.5.5. The source T field is the code for the source address and the destination T field is the code for the destination address. As shown in Figure 4-4, only five instruction formats use a T field.

- Source and destination register fields which contain the number of the register affected (0 through 15).

- Displacement fields that contain a bias to be added to the program counter in program counter relative addressing. This form of addressing is further described in paragraph 4.5.7.

- Fields that contain counts for indicating the number of bits that will be shifted in a shift instruction or the number of Communication Register Unit (CRU) bits that will be addressed in a CRU instruction.

4.5.1 DIRECT REGISTER ADDRESSING ($T=00_2$)

In direct register addressing, execution involves data contained within one of the 16 workspace registers. In the first example in Figure 4-5, both the source and destination operands are registers as noted in the assembly language example at the top of the figure. Both T fields contain 00_2 to denote direct register addressing and their associated register fields contain the binary value of the number of the register affected. The 110_2 in the op code field identifies this instruction as a move instruction. Since the B field contains a zero, the data moved will be the full 16 bits of the register (a byte instruction addressing a register would address the left byte of the register). The instruction specifies moving the contents of register 1 to register 4, thus changing the contents of register 4 to the same value as in register 1. Note that the assembly language statement is constructed so that the source register is the first item in the operand while the destination register is the second item in the operand. This order is reversed in the machine language construction with the destination register and its T field first and the source register and its T field second.

4.5.2 INDIRECT REGISTER ADDRESSING ($T=01_2$)

In indirect register addressing, the register does not contain the data to be affected by the instruction; instead, the register contains the address within memory of where that data is stored. For example, the instruction in Figure 4-6 specifies to move the contents of register 1 to the address which is contained in register 4 (indirect register 4). Instead of moving the value in register 1 to register 4 as was the case in Figure 4-5, the CPU must first read in the 16-bit value in register 4 and use that value as a memory address at which location the contents of register 1 will be stored. In the example, register 4 contains the value $FD00_{16}$. This instruction stores the value in register 1 into memory address (MA) $FD00_{16}$.

In direct register addressing, the contents of a register are addressed. In indirect register addressing, the CPU goes to the register to find out what memory location to address. This form of addressing is especially suited for repeating an instruction while accessing successive memory addresses. For example, if you wished to add a series of numbers in 100 consecutive memory locations, you could place the address of the first number in a register, and execute an add indirect through that register, causing the contents of the first memory address (source operand) to be added to another register or memory address (destination operand). Then you could increment the contents of the register containing the address of the number, loop back to the add instruction, and repeat the add, only this time you will be adding the contents of the next memory address to the accumulator (destination operand). This way a whole string of data can be summed using a minimum of instructions. Of course, you would have to include control instructions that would signal when the entire list of 100 addresses have been added, but there are obvious advantages in speed of operation, better utilization of memory space, and ease in programming.

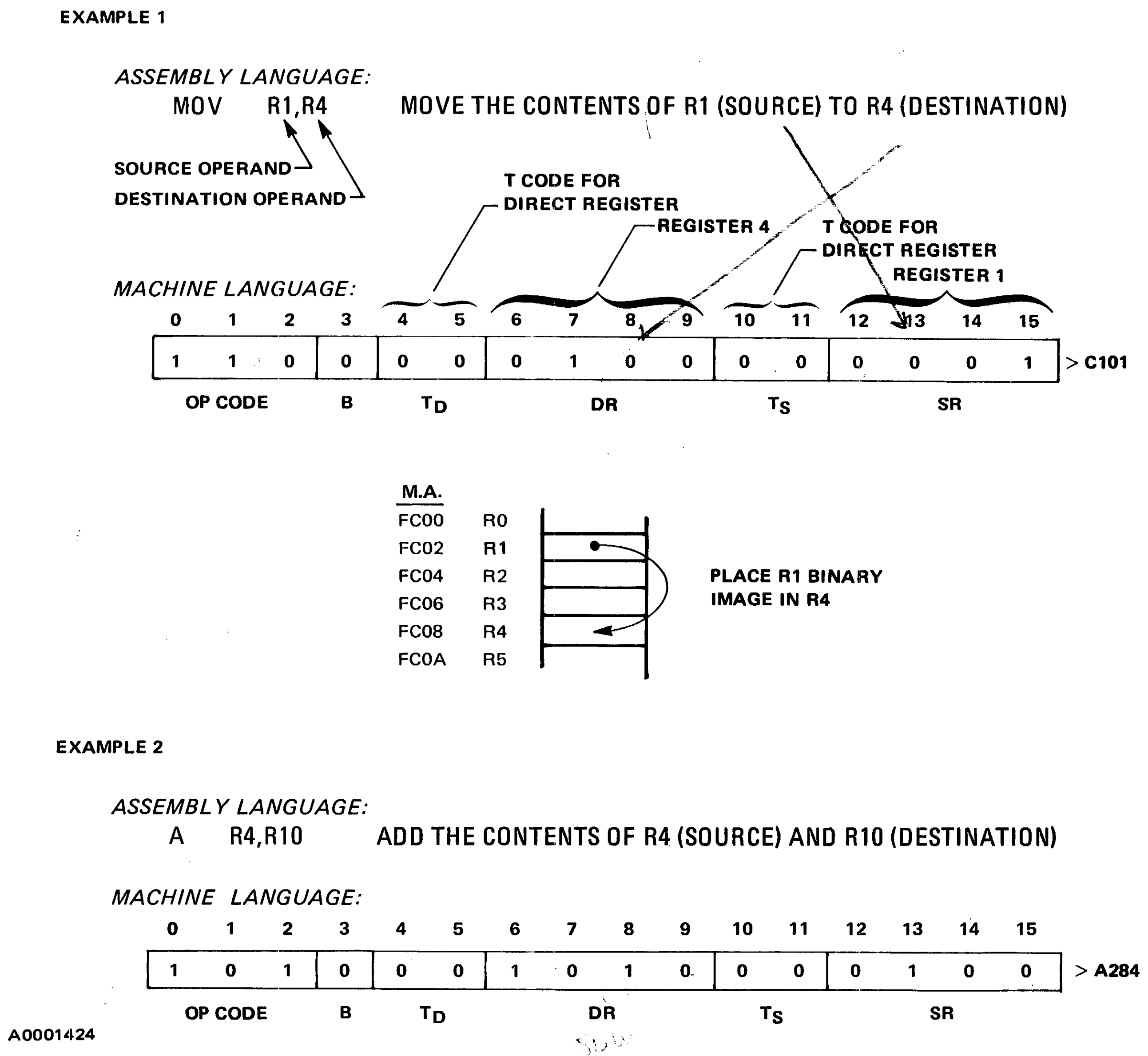


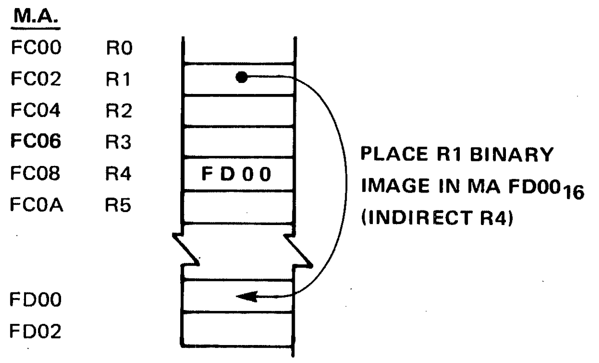
FIGURE 4-5. DIRECT REGISTER ADDRESSING EXAMPLE

ASSEMBLY LANGUAGE:

MOV R1,*R4 MOVE THE CONTENTS OF RI (SOURCE) TO ADDRESS IN R4 (DESTINATION)

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	0	0	1	0	1	0	0	0	0	0	0	0	1	>C501
OP CODE			B	T_D		DR				T_S		SR				



A0001425

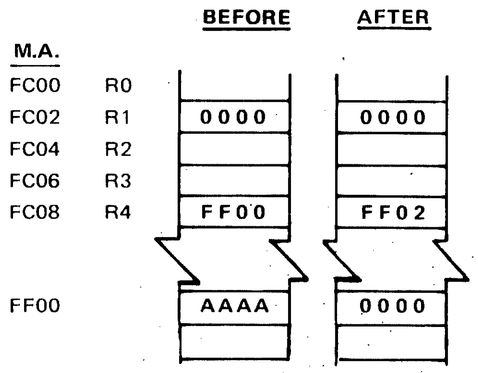
FIGURE 4-6. INDIRECT REGISTER ADDRESSING EXAMPLE

ASSEMBLY LANGUAGE:

MOV R1,*R4+ MOVE THE CONTENTS OF RI TO ADDRESS CONTAINED IN R4, INCREMENT ADDRESS BY 2

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	0	1	1	0	1	0	0	0	0	0	0	0	1	> CD01
OP CODE			B	T_D		DR				T_S		SR				



A0001427

FIGURE 4-7. INDIRECT REGISTER AUTOINCREMENT ADDRESSING EXAMPLE

4.5.3 INDIRECT REGISTER AUTOINCREMENT ADDRESSING ($T=11_2$)

Indirect register autoincrement addressing is the same as indirect register addressing (paragraph 4.5.2) except for an additional feature – automatic incrementation of the register. This saves the requirement of adding an increment (by one or two) instruction to increment the register being used in the indirect mode. The increment will be a value of one for byte instructions (e.g., add byte or AB) or a value of two for full word instructions (e.g., add word or A).

In assembly language, the register number is preceded by an asterisk (*) and followed by a plus sign (+) as shown in Figure 4-7. Note in the figure that the contents of register 4 was incremented by two since the instruction was a move word (vs. byte) instruction. If the example used a move byte instruction, the contents of the register would be incremented by one so that successive bytes would be addressed (the 16-bit word addresses in memory are always even numbers or multiples of two since each contains two bytes). Bytes are also addressed by various instructions of the 990 instruction set.

Note that only a register can contain the indirect address.

4.5.4 SYMBOLIC MEMORY ADDRESSING, NOT INDEXED ($T=10_2$)

This mode does not use a register as an address or as a container of an address. Instead, the address is a 16-bit value stored in the second or third word of the instruction. The SR or DR fields will be all zeroes as shown for the destination register field in the first example of Figure 4-8. When the T field contains 10_2, the CPU retrieves the contents of the next memory location and uses these contents as the effective address. In assembly language, a symbolic address is preceded by an at sign (@) to differentiate a numerical memory address from a register number. All alphanumeric labels must be preceded by an @ sign; numerical values preceded by an @ sign will be assembled as an absolute address (the TM 990/402 Line-By-Line Assembler does not recognize alphanumeric symbols but does recognize absolute memory addresses).

In the second example in Figure 4-8, both the source and destination operands are symbolic memory addresses. In this case, the source address is the first word following the instruction and the destination is the second word following the instruction in machine language.

4.5.5 SYMBOLIC MEMORY ADDRESSING, INDEXED ($T=10_2$)

Note that the T field for indexed as well as non-indexed symbolic addressing is the same (10_2). In order to differentiate between the two different modes, the associated SR or DR field is interrogated; if this field is all zeroes (0000_2), non-indexed addressing is specified; if the SR or DR field is greater than zero, indexing is specified and the non-zero value is the index register number. As a result, register 0 cannot be used as an index register.

In assembly language, the symbolic address is followed by the number of the index register in parentheses. In the example in Figure 4-9, the source operand is non-indexed symbolic memory addressing while the destination operand is indexed symbolic memory addressing. In this case, the destination effective address is the sum of the $FF02_{16}$ value in the source memory address word plus the value in the index register (0004_{16}). The effective address in this case is $FF06_{16}$ as shown by the addition in the left part of the figure.

Note that only symbolic addressing can be indexed.

EXAMPLE 1

ASSEMBLY LANGUAGE:

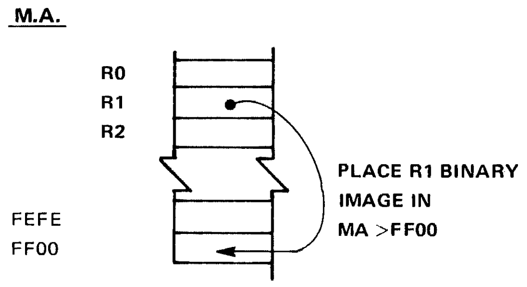
MOV R1,@>FF00 **MOVE THE CONTENTS OF RI TO ADDRESS >FF00**

NOTE

The > sign indicates hexidecimal representation.

MACHINE LANGUAGE:

	OP CODE			B	T_D		DR				T_S		SR				
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1st WORD	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	> C801
2nd WORD	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	> FF00



EXAMPLE 2

ASSEMBLY LANGUAGE:

MOV @>FF0A,@>FF08 **MOVE THE CONTENTS OF >FF0A TO >FF08**

MACHINE LANGUAGE:

	OP CODE			B	T_D		DR				T_S		SR				
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1st WORD	1	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	>C820
2nd WORD	1	1	1	1	1	1	1	1	0	0	0	0	1	0	1	0	>FF0A (SOURCE)
3rd WORD	1	1	1	1	1	1	1	1	0	0	0	0	1	0	0	0	>FF08 (DESTINATION)

M.A.	BEFORE	AFTER
FF08	FFFF	0000
FF0A	0000	0000

A0001428

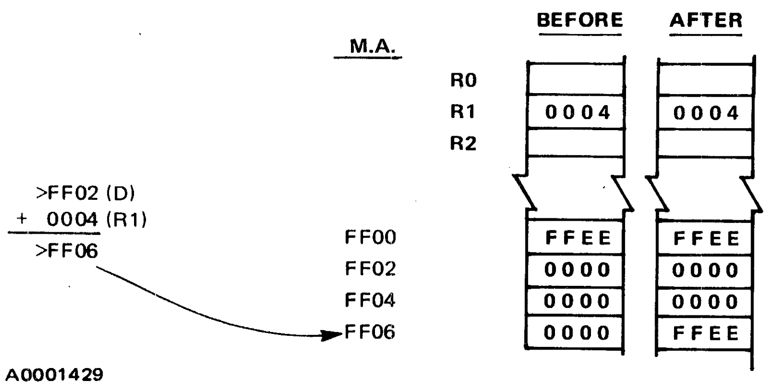
FIGURE 4-8. DIRECT MEMORY ADDRESSING EXAMPLE

ASSEMBLY LANGUAGE:

```
MOV   @>FF00,@>FF02(R1)       MOVE THE CONTENTS OF >FF00 TO >FF02 + RI CONTENTS
```

MACHINE LANGUAGE:

OP CODE			B	TD		DR				TS		SR				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	0	1	0	0	0	0	1	1	0	0	0	0	0	>C860
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	>FF00 (SOURCE)
1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	0	>FF02 (DESTINATION)



A0001429

FIGURE 4-9. DIRECT MEMORY ADDRESSING, INDEXED EXAMPLE

4.5.6 IMMEDIATE ADDRESSING

This mode allows an absolute value to be specified as an operand; this value is used in connection with a register contents or is loaded into the WP or the Status Register interrupt mask. Examples are shown below:

```
LI        R2,100       LOAD 100 INTO REGISTER 2
CI        R8,>100      COMPARE R8 CONTENTS TO >100, RESULTS IN ST
LWPI      >FC00        SET WP TO MA >FC00
```

4.5.7 PROGRAM COUNTER RELATIVE ADDRESSING

This mode allows a change in Program Counter contents, either an unconditional change or a change conditional on Status Register contents. Examples are shown below:

```
JMP       $+6          JUMP TO LOCATION, 6 BYTES FORWARD
JMP       THERE        JUMP TO LOCATION LABELLED THERE
JEQ       $+4          IF ST EQ BIT = 1, JUMP 4 BYTES (MA + 4)
JMP       >FE26        JUMP TO M.A. >FE26 (LINE-BY-LINE ASSEMBLER ONLY)
```

The dollar symbol ($) means "from this address"; thus, $+6 means "this address plus 6 bytes."

4.6 INSTRUCTIONS

Table 4-2 lists terms used in describing the instructions of the TM 990/100M. Table 4-3 is an alphabetical list of instructions. Table 4-4 is a numerical list of instructions by op code. Examples are shown in both assembly language (A.L.) and machine language (M.L.). The greater-than sign (>) indicates hexadecimal.

TABLE 4-2. INSTRUCTION DESCRIPTION TERMS

TERM	DEFINITION
B	Byte indicator (1 = byte, 0 = word)
C	Bit count
DR	Destination address register
DA	Destination address
IOP	Immediate operand
LSB(n)	Least significant (right most) bit of (n)
M.A.	Memory Address
MSB(n)	Most significant (left most) bit of (n)
N	Don't care
PC	Program counter
Result	Result of operation performed by instruction
SR	Source address register
SA	Source address
ST	Status register
STn	Bit n of status register
T_D	Destination address modifier
T_S	Source address modifier
WR or R	Workspace register
WRn or Rn	Workspace register n
(n)	Contents of n
a → b	a is transferred to b
(a) → b	Contents of a is transferred to be
[n]	Absolute value of n
+	Arithmetic addition
—	Arithmetic subtraction
AND	Logical AND
OR	Logical OR
$\oplus$	Logical exclusive OR
n	Logical complement of n
>	Hexadecimal value

TABLE 4-3. INSTRUCTION SET, ALPHABETICAL INDEX

ASSEMBLY LANGUAGE MNEMONIC	MACHINE LANGUAGE OP CODE	FORMAT	STATUS REG. BITS AFFECTED	RESULT COMPARED TO ZERO	INSTRUCTION	PARAGRAPH
A	A000	1	0-4	X	Add (word)	4.6.1
AB	B000	1	0-5	X	Add (byte)	4.6.1
ABS	0740	6	0-2	X	Absolute Value	4.6.6
AI	0220	8	0-4	X	Add Immediate	4.6.8
ANDI	0240	8	0-2	X	AND Immediate	4.6.8
B	0440	6	–		Branch	4.6.6
BL	0680	6	–		Branch and Link (R11)	4.6.6
BLWP	0400	6	–		Branch; New Workspace Pointer	4.6.6
C	8000	1	0-2		Compare (word)	4.6.1
CB	9000	1	0-2,5		Compare (byte)	4.6.1
CI	0280	8	0-2		Compare Immediate	4.6.8
CKOF	03C0	7	–		User Defined	4.6.7
CKON	03A0	7	–		User Defined	4.6.7
CLR	04C0	6	–		Clear Operand	4.6.6
COC	2000	3	2		Compare Ones Corresponding	4.6.3
CZC	2400	3	2		Compare Zeroes Corresponding	4.6.3
DEC	0600	6	0-4	X	Decrement (by one)	4.6.6
DECT	0640	6	0-4	X	Decrement (by two)	4.6.6
DIV	3C00	9	4		Divide	4.6.3
IDLE	0340	7	–		Computer Idle	4.6.7
INC	0580	6	0-4	X	Increment (by one)	4.6.6
INCT	05C0	6	0-4	X	Increment (by two)	4.6.6
INV	0540	6	0-2	X	Invert (One's Complement)	4.6.6
JEQ	1300	2	–		Jump Equal (ST2=1)	4.6.2
JGT	1500	2	–		Jump Greater Than (ST1=1), Arithmetic	4.6.2
JH	1B00	2	–		Jump High (ST0=1 and ST2=0), Logical	4.6.2
JHE	1400	2	–		Jump High or Equal (ST0 or ST2=1), Logical	4.6.2
JL	1A00	2	–		Jump Low (ST0 and ST2=0), Logical	4.6.2
JLE	1200	2	–		Jump Low or Equal (ST0=0 or ST2=1), Logical	4.6.2
JLT	1100	2	–		Jump Less Than (ST1 and ST2=0), Arithmetic	4.6.2
JMP	1000	2	–		Jump Unconditional	4.6.2
JNC	1700	2	–		Jump No Carry (ST3=0)	4.6.2
JNE	1600	2	–		Jump Not Equal (ST2=0)	4.6.2
JNO	1900	2	–		Jump No Overflow (ST4=0)	4.6.2
JOC	1800	2	–		Jump On Carry (ST3=1)	4.6.2

TABLE 4-3. INSTRUCTION SET, ALPHABETICAL INDEX (Concluded)

ASSEMBLY LANGUAGE MNEMONIC	MACHINE LANGUAGE OP CODE	FORMAT	STATUS REG. BITS AFFECTED	RESULT COMPARED TO ZERO	INSTRUCTION	PARAGRAPH
JOP	1C00	2	–		Jump Odd Parity (ST5=1)	4.6.2
LDCR	3000	4	0-2,5	X	Load CRU	4.6.4
LI	0200	8	–	X	Load Immediate	4.6.8
LIMI	0300	8	12-15		Load Interrupt Mask Immediate	4.6.8
LREX	03E0	7	12-15		Load and Execute	4.6.7
LWPI	02E0	8	–		Load Immediate to Workspace Pointer	4.6.8
MOV	C000	1	0-2	X	Move (word)	4.6.1
MOVB	D000	1	0-2,5	X	Move (byte)	4.6.1
MPY	3800	9	–		Multiply	4.6.3
NEG	0500	6	0-2	X	Negate (Two's Complement)	4.6.6
ORI	0260	8	0-2	X	OR Immediate	4.6.8
RSET	0360	7	12-15		Reset AU	4.6.7
RTWP	0380	7	0-15		Return from Context Switch	4.6.7
S	6000	1	0-4	X	Subtract (word)	4.6.1
SB	7000	1	0-5	X	Subtract (byte)	4.6.1
SBO	1D00	2	–		Set CRU Bit to One	4.6.2
SBZ	1E00	2	–		Set CRU Bit to Zero	4.6.2
SETO	0700	6	–		Set Ones	4.6.6
SLA	0A00	5	0-4	X	Shift Left Arithmetic	4.6.5
SOC	E000	1	0-2	X	Set Ones Corresponding (word)	4.6.1
SOCB	F000	1	0-2,5	X	Set Ones Corresponding (byte)	4.6.1
SRA	0800	5	0-3	X	Shift Right (sign extended)	4.6.5
SRC	0B00	5	0-3	X	Shift Right Circular	4.6.5
SRL	0900	5	0-3	X	Shift Right Logical	4.6.5
STCR	3400	4	0-2,5	X	Store From CRU	4.6.4
STST	02C0	8	–		Store Status Register	4.6.8
STWP	02A0	8	–		Store Workspace Pointer	4.6.8
SWPB	06C0	6	–		Swap Bytes	4.6.6
SZC	4000	1	0-2	X	Set Zeroes Corresponding (word)	4.6.1
SZCB	5000	1	0-2,5	X	Set Zeroes Corresponding (byte)	4.6.1
TB	1F00	2	2		Test CRU Bit	4.6.2
X	0480	6	–		Execute	4.6.6
XOP	2C00	9	6		Extended Operation	4.6.9
XOR	2800	3	0-2	X	Exclusive OR	4.6.3

TABLE 4-4. INSTRUCTION SET, NUMERICAL INDEX

MACHINE LANGUAGE OP CODE (HEXADECIMAL)	ASSEMBLY LANGUAGE MNEMONIC	INSTRUCTION	FORMAT	STATUS BITS AFFECTED
0200	LI	Load Immediate	8	0-2
0220	AI	Add Immediate	8	0-4
0240	ANDI	And Immediate	8	0-2
0260	ORI	Or Immediate	8	0-2
0280	CI	Compare Immediate	8	0-2
02A0	STWP	Store WP	8	–
02C0	STST	Store ST	8	–
02E0	LWPI	Load WP Immediate	8	–
0300	LIMI	Load Int. Mask	8	12-15
0340	IDLE	Idle	7	–
0360	RSET	Reset AU	7	12-15
0380	RTWP	Return from Context Sw.	7	0-15
03A0	CKON	User Defined	7	–
03C0	CKOF	User Defined	7	–
03E0	LREX	Load & Execute	7	–
0400	BLWP	Branch; New WP	6	–
0440	B	Branch	6	–
0480	X	Execute	6	–
04C0	CLR	Clear to Zeroes	6	–
0500	NEG	Negate to Ones	6	0-2
0540	INV	Invert	6	0-2
0580	INC	Increment by 1	6	0-4
05C0	INCT	Increment by 2	6	0-4
0600	DEC	Decrement by 1	6	0-4
0640	DECT	Decrement by 2	6	0-4
0680	BL	Branch and Link	6	–
06C0	SWPB	Swap Bytes	6	–
0700	SETO	Set to Ones	6	–
0740	ABS	Absolute Value	6	0-2
0800	SRA	Shift Right Arithmetic	5	0-3
0900	SRL	Shift Right Logical	5	0-3
0A00	SLA	Shift Left Arithmetic	5	0-4
0B00	SRC	Shift Right Circular	5	0-3
1000	JMP	Unconditional Jump	2	–
1100	JLT	Jump on Less Than	2	–
1200	JLE	Jump on Less Than or Equal	2	–
1300	JEQ	Jump on Equal	2	–
1400	JHE	Jump on High or Equal	2	–
1500	JGT	Jump on Greater Than	2	–
1600	JNE	Jump on Not Equal	2	–
1700	JNC	Jump on No Carry	2	–
1800	JOC	Jump on Carry	2	–
1900	JNO	Jump on No Overflow	2	–
1A00	JL	Jump on Low	2	–
1B00	JH	Jump on High	2	–
1C00	JOP	Jump on Odd Parity	2	–
1D00	SBO	Set CRU Bits to Ones	2	–
1E00	SBZ	Set CRU Bits to Zeroes	2	–
1F00	TB	Test CRU Bit	2	2
2000	COC	Compare Ones Corresponding	3	2

TABLE 4-4. INSTRUCTION SET, NUMERICAL INDEX (Concluded)

MACHINE LANGUAGE OP CODE (HEXADECIMAL	ASSEMBLY LANGUAGE MNEMONIC	INSTRUCTION	FORMAT	STATUS BITS AFFECTED
2400	CZC	Compare Zeroes Corresponding	3	2
2800	XOR	Exclusive Or	3	0-2
2C00	XOP	Extended Operation	9	6
3000	LDCR	Load CRU	4	0-2,5
3400	STCR	Store CRU	4	0-2,5
3800	MPY	Multiply	9	—
3C00	DIV	Divide	9	4
4000	SZC	Set Zeroes Corresponding (Word)	1	0-2
5000	SZCB	Set Zeroes Corresponding (Byte)	1	0-2,5
6000	S	Subtract Word	1	0-4
7000	SB	Subtract Byte	1	0-5
8000	C	Compare Word	1	0-2
9000	CB	Compare Byte	1	0-2,5
A000	A	Add Word	1	0-4
B000	AB	Add Byte	1	0-5
C000	MOV	Move Word	1	0-2
D000	MOVB	Move Byte	1	0-2,5
E000	SOC	Set Ones Corresponding (Word)	1	0-2
F000	SOCB	Set Ones Corresponding (Byte)	1	0-2,5

4.6.1 FORMAT 1 INSTRUCTION.

These are dual operand instructions with multiple addressing modes for source and destination operands.

GENERAL FORMAT:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OP CODE			B	T_D		DR				T_S		SR			

If B = 1, the operands are bytes and the operand addresses are byte addresses. If B = 0, the operands are words and the operand addresses are word addresses.

MNEMONIC	OP CODE 0 1 2	B 3	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
A	1 0 1	0	Add	Yes	0-4	(SA)+(DA) → (DA)
AB	1 0 1	1	Add bytes	Yes	0-5	(SA)+(DA) → (DA)
C	1 0 0	0	Compare	No	0-2	Compare (SA) to (DA) and set appropriate status bits
CB	1 0 0	1	Compare bytes	No	0-2,5	Compare (SA) to (DA) and set appropriate status bits
MOV	1 1 0	0	Move	Yes	0-2	(SA) → (DA)
MOVB	1 1 0	1	Move bytes	Yes	0-2,5	(SA) → (DA)
S	0 1 1	0	Subtract	Yes	0-4	(DA) – (SA) → (DA)
SB	0 1 1	1	Subtract bytes	Yes	0-5	(DA) – (SA) → (DA)
SOC	1 1 1	0	Set ones corresponding	Yes	0-2	(DA) OR (SA) → (DA)
SOCB	1 1 1	1	Set ones corresponding bytes	Yes	0-2,5	(DA) OR (SA) → (DA)
SZC	0 1 0	0	Set zeroes corresponding	Yes	0-2	(DA) AND ($\overline{\text{SA}}$) → (DA)
SZCB	0 1 0	1	Set zeroes corresponding bytes	Yes	0-2,5	(DA) AND ($\overline{\text{SA}}$) → (DA)

EXAMPLES

(1) ASSEMBLY LANGUAGE:

```
A    @>100,R2        ADD CONTENTS OF MA >100 & R2, SUM IN R2
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	>A0A0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	>0100

(2) ASSEMBLY LANGUAGE:

```
CB   R1,R2       COMPARE BYTE R1 TO R2, SET ST
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	>9081

NOTE

In byte instruction designating a register, the left byte is used. In the above example, the left byte (8 MSB's) of R1 is compared to the left byte of R2, and the ST set to the results.

4.6.2 FORMAT 2 INSTRUCTIONS

4.6.2.1 Jump Instructions

Jump instructions cause the PC to be loaded with the value [PC+2(signed displacement)] if bits of the Status Register are at specified values. Otherwise, no operation occurs and the next instruction is executed since the PC was incremented by two and now points to the next instruction. The signed displacement field is a word (not byte) count to be added to PC. Thus, the jump instruction has a range of –128 to 127 words (–256 to 254 bytes) from the memory address following the jump instruction. No ST bits are affected by a jump instruction.

GENERAL FORMAT:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OP CODE								SIGNED DISPLACEMENT (WORDS)							

MNEMONIC	OP CODE								MEANING	ST CONDITION TO CHANGE PC
	0	1	2	3	4	5	6	7		
JEQ	0	0	0	1	0	0	1	1	Jump equal	ST2 = 1
JGT	0	0	0	1	0	1	0	1	Jump greater than	ST1 = 1
JH	0	0	0	1	1	0	1	1	Jump high	ST0 = 1 and ST2 = 0
JHE	0	0	0	1	0	1	0	0	Jump high or equal	ST0 = 1 or ST2 = 1
JL	0	0	0	1	1	0	1	0	Jump low	ST0 = 0 and ST2 = 0
JLE	0	0	0	1	0	0	1	0	Jump low or equal	ST0 = 0 or ST2 = 1
JLT	0	0	0	1	0	0	0	1	Jump less than	ST1 = 0 and ST2 = 0
JMP	0	0	0	1	0	0	0	0	Jump unconditional	unconditional
JNC	0	0	0	1	0	1	1	1	Jump no carry	ST3 = 0
JNE	0	0	0	1	0	1	1	0	Jump not equal	ST2 = 0
JNO	0	0	0	1	1	0	0	1	Jump no overflow	ST4 = 0
JOC	0	0	0	1	1	0	0	0	Jump on carry	ST3 = 1
JOP	0	0	0	1	1	1	0	0	Jump odd parity	ST5 = 1

In assembly language, $ in the operand indicates "at this instruction". Essentially JMP $ causes an unconditional loop to the same instruction location, and JMP $+2 is essentially a no-op ($+2 means "here plus two bytes"). Note that the number following the $ is a *byte* count while displacement in machine language is in *words.*

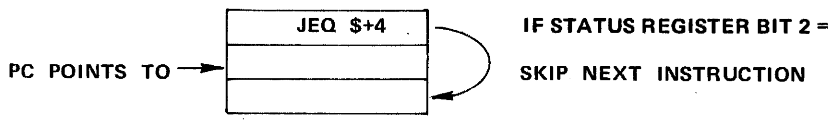
EXAMPLES

(1) ASSEMBLY LANGUAGE:

```
JEQ    $+4        IF EQ BIT SET, SKIP 1 INSTRUCTION
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	1	>1301



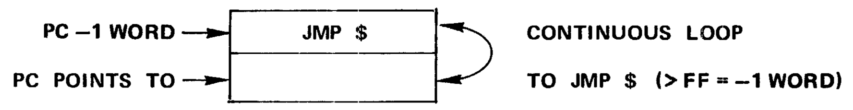
The above instruction continues execution 4 bytes (2 words) from the instruction location or, in other words, two bytes (one word) from the Program Counter value (incremented by 2 and now pointing to next instruction while JEQ executes). Thus, the signed displacement of 1 word (2 bytes) is the value to be added to the PC.

(2) *ASSEMBLY LANGUAGE:*

```
JMP   $      REMAIN AT THIS LOCATION
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	1	0	0	0	0	1	1	1	1	1	1	1	1	>10FF



This causes an unconditional loop back to one word less than the Program Counter value (PC + >FF = PC-1 word). The Status Register is not checked. A JMP $+2 means "go to the next instruction" and has a displacement of zero (a no-op). No-ops can substitute for deleted code or can be used for timing purposes.

4.6.2.2 **CRU Single-Bit Instructions.**

These instructions test or set values at the Communications Register Unit (CRU). The CRU bit is selected by the CRU address in bits 3 to 14 of register 12 plus the signed displacement value. The selected bit is set to a one or zero, or it is tested and the bit value placed in equal bit (2) of the Status Register. The signed displacement has a value of –128 to 127.

NOTE

CRU addressing is discussed in detail in paragraph 4.7.

	0–7	8–15
General Format:	**OP CODE**	**SIGNED DISPLACEMENT**

MNEMONIC	OP CODE 0 1 2 3 4 5 6 7	MEANING	STATUS BITS AFFECTED	DESCRIPTION
SBO	0 0 0 1 1 1 0 1	Set bit to one	–	Set the selected CRU output bit to 1.
SBZ	0 0 0 1 1 1 1 0	Set bit to zero	–	Set the selected CRU output bit to 0.
TB	0 0 0 1 1 1 1 1	Test bit	2	If the selected CRU input bit = 1, set ST2.

EXAMPLE

R12, BITS 3 TO 14 = >100

ASSEMBLY LANGUAGE:

```
SBO   4      SET CRU ADDRESS >104 TO ONE
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	1	1	1	0	1	0	0	0	0	0	1	0	0	>1D04

4.6.3 FORMAT 3/9 INSTRUCTIONS

These are dual operand instructions with multiple addressing modes for the source operand, and workspace register addressing for the destination. The MPY and DIV instructions are termed format 9 but both use the same format as format 3. The XOP instruction is covered in paragraph 4.6.9.

General Format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OP CODE						DR (REGISTER ONLY)				T_S		SR			

MNEMONIC	OP CODE 0 1 2 3 4 5	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
COC	0 0 1 0 0 0	Compare ones corresponding	No	2	Test (DR) to determine if 0's are in each bit position where 1's are in (SA). If so, set ST2.
CZC	0 0 1 0 0 1	Compare zeros corresponding	No	2	Test (DR) to determine if 0's are in each bit position where 1's are in (SA). If so, set ST2.
XOR	0 0 1 0 1 0	Exclusive OR	Yes	0-2	(DR) $\oplus$ (SA) $\rightarrow$ (DR)
MPY	0 0 1 1 1 0	Multiply	No		Multiply unsigned (DR) by unsigned (SA) and place unsigned 32-bit product in DR (most significant) and DR + 1 (least significant). If WR15 is DR, the next word in memory after WR15 will be used for the least significant half of the product.
DIV	0 0 1 1 1 1	Divide	No	4	If unsigned (SA) is less than or equal to unsigned (DR), perform no operation and set ST4. Otherwise divide unsigned (DR) and (DR) by unsigned (SA). Quotient $\rightarrow$ (DR), remainder $\rightarrow$ (DR+1). If DR=15, the next word in memory after WR15 will be used for the remainder.

Exclusive OR Logic = $1 \oplus 0 = 1$
$0 \oplus 0 = 0$
$1 \oplus 1 = 0$

EXAMPLES

(1) ASSEMBLY LANGUAGE:

```
MPY    R2,R3        MULTIPLY CONTENTS OF R2 AND R3, RESULT IN R3 AND R4
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	1	1	0	0	0	1	1	0	0	0	0	1	0	>38C2

	BEFORE	AFTER	
R2	0002	0002	
R3	0003	0000	32-BIT
R4	N	0006	RESULT

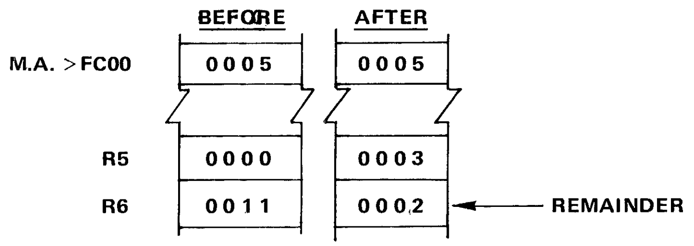
The destination operand is always a register, and the values multiplied are 16-bits, unsigned. The 32-bit result is placed in the destination register and destination register +1, zero filled on the left.

(2) *ASSEMBLY LANGUAGE:*

```
DIV   @>FC00,R5       DIVIDE CONTENTS OF R5 AND R6 BY VALUE AT M.A. >FC00
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	1	1	1	0	1	0	1	1	0	0	0	0	0	>3D60
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	>FC00



The unsigned 32-bit value in the destination register and destination register +1 is divided by the source operand value. The result is placed in the destination register. The remainder is placed in the destination register +1.

(3) *ASSEMBLY LANGUAGE:*

```
COC   R10,R11      ONES IN R10 ALSO IN R11?
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	0	0	0	1	0	1	1	0	0	1	0	1	0	>22CA

Locate all binary ones in the source operand. If the destination operand also has ones in these positions, set the equal flag in the Status Register; otherwise, reset this flag. The following sets the equal flag:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
R10	1	0	1	0	1	0	1	0	0	0	0	0	1	1	0	0	>AA0C
R11	1	1	1	0	1	1	1	1	1	1	0	0	1	1	0	1	>EFCD

Set EQ bit in Status Register to 1.

4.6.4 FORMAT 4 (CRU MULTIBIT) INSTRUCTIONS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
General Format:	OP CODE						C				T_S		SR			

The C field specifies the number of bits to be transferred. If C = 0, 16 bits will be transferred. The CRU base register (WR 12, bits 3 through 14) defines the starting CRU bit address. The bits are transferred serially and the CRU address is incremented with each bit transfer, although the contents of WR12 are not affected. T_S and SA provide multiple mode addressing capability for the source operand. If 8 or fewer bits are transferred (C = 1 through 8), the source address is a byte address. If 9 or more bits are transferred (C = 0, 9 through 15), the source address is a word (even number) address. If the source is addressed in the workspace register indirect autoincrement mode, the workspace register is incremented by 1 if C = 1 through 8, and is incremented by 2 otherwise.

MNEMONIC	OP CODE	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
	0 1 2 3 4 5				
LDCR	0 0 1 1 0 0	Load communcation register	Yes	0-2,5†	Beginning with LSB of (SA), transfer the specified number of bits from (SA) to the CRU.
STCR	0 0 1 1 0 1	Store communcation register	Yes	0-2,5†	Beginning with LSB of (SA), transfer the specified number of bits from the CRU to (SA). Load unfilled bit positions with 0.

†ST5 is affected only if $1 \leqslant C \leqslant 8$.

EXAMPLE

ASSEMBLY LANGUAGE:

```
LDCR    @>FE00,8        LOAD 8 BITS ON CRU FROM M.A. >FE00
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	1	0	0	1	0	0	0	1	0	0	0	0	0	>3220
1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	>FE00

NOTE

CRU addressing is discussed in detail in paragraph 4.7.

4.6.5 FORMAT 5 (SHIFT) INSTRUCTIONS

These instructions shift (left, right, or circular) the bit patterns in a workspace register. The last bit value shifted out is placed in the carry bit (3) of the Status Register. If the SLA instruction causes a one to be shifted into the sign bit, the ST overflow bit (4) is set. The C field contains the number of bits to shift.

	0–7	8–11	12–15
General Format:	OP CODE	C	R

If C = 0, bits 12 through 15 of R0 contain the shift count. If C = 0 and bits 12 through 15 of WR0 = 0, the shift count is 16.

MNEMONIC	OP CODE 0 1 2 3 4 5 6 7	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
SLA ✓	0 0 0 0 1 0 1 0	Shift left arithmetic	Yes	0-4	Shift (R) left. Fill vacated bit positions with 0.
SRA	0 0 0 0 1 0 0 0	Shift right arithmetic	Yes	0-3	Shift (R) right. Fill vacated bit positions with original MSB of (R).
SRC	0 0 0 0 1 0 1 1	Shift right circular	Yes	0-3	Shift (R) right. Shift previous LSB into MSB.
SRL ✓	0 0 0 0 1 0 0 1	Shift right logical	Yes	0-3	Shift (R) right. Fill vacated bit positions with 0's.

EXAMPLES

(1) *ASSEMBLY LANGUAGE:*

```
SRA    R1,2        SHIFT R1 RIGHT 2 POSITIONS, CARRY SIGN
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0821 ~~>0841~~

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
R1 BEFORE	1	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	>8F0F
R1 AFTER	1	1	1	0	0	0	1	1	1	1	0	0	0	0	1	1	>E3C3

SIGN BIT CARRIED IN

(2) *ASSEMBLY LANGUAGE:*

```
SRC    R5,4        CIRCULAR SHIFT R5 4 POSITIONS
```

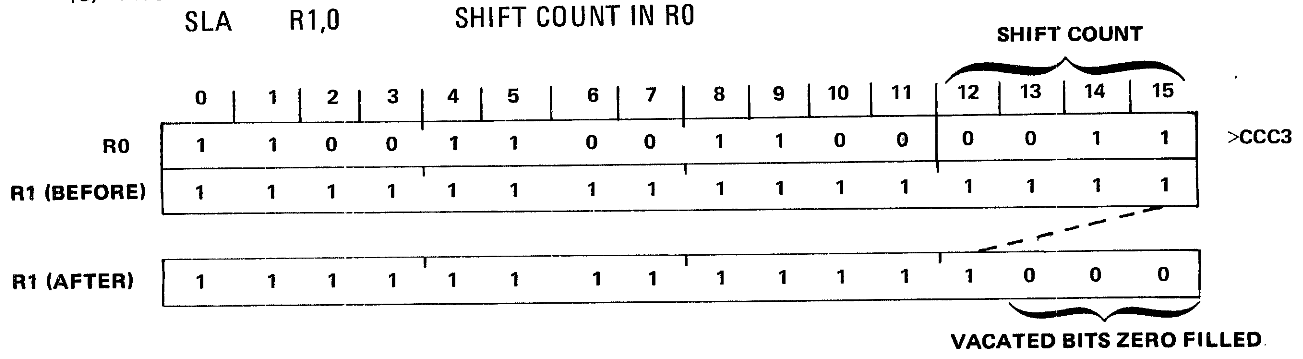
MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	1	0B45 ~~>0845~~

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
R5 BEFORE	0	0	0	0	1	0	0	1	0	0	0	0	1	1	1	1	>090F
R5 AFTER	1	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	>F090

(3) ASSEMBLY LANGUAGE:

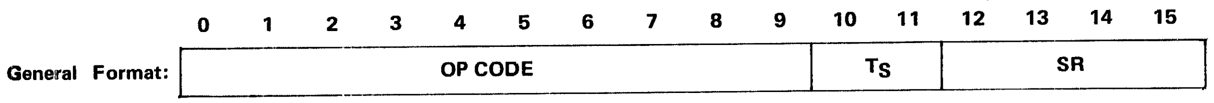
SLA R1,0 SHIFT COUNT IN R0



SHIFT COUNT
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
R0 1 1 0 0 1 1 0 0 1 1 0 0 0 0 1 1
>CCC3
R1 (BEFORE) 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
R1 (AFTER) 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
VACATED BITS ZERO FILLED

4.6.6 FORMAT 6 INSTRUCTIONS

These are single operand instructions.



The T_S and S fields provide multiple mode addressing capability for the source operand.

MNEMONIC	OP CODE 0 1 2 3 4 5 6 7 8 9	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
B	0 0 0 0 0 1 0 0 0 1	Branch	No	–	SA → (PC)
BL	0 0 0 0 0 1 1 0 1 0	Branch and link	No	–	(PC) → (R11); SA → (PC)
BLWP	0 0 0 0 0 1 0 0 0 0	Branch and load workspace pointer	No	–	(SA) → (WP); (SA+2) → (PC); (old WP) → (new WR13); (old PC) → (new WR14); (old ST) → (new WR15); the interrupt input ($\overline{\text{INTREQ}}$) is not tested upon completion of the BLWP instruction.
CLR	0 0 0 0 0 1 0 0 1 1	Clear operand	No	–	0000 → (SA)
SETO	0 0 0 0 0 1 1 1 0 0	Set to ones	No	–	$FFFF_{16}$ → (SA)
INV	0 0 0 0 0 1 0 1 0 1	Invert	Yes	0-2	$\overline{(SA)}$ → (SA) (ONE'S complement)
NEG	0 0 0 0 0 1 0 1 0 0	Negate	Yes	0-4	– (SA) → (SA)(TWO'S complement)
ABS	0 0 0 0 0 1 1 1 0 1	**Absolute value***	No	0-4	|(SA)| → (SA)
SWPB	0 0 0 0 0 1 1 0 1 1	Swap bytes	No	–	(SA), bits 0 thru 7 → (SA), bits 8 thru 15; (SA), bits 8 thru 15 → (SA), bits 0 thru 7.
INC	0 0 0 0 0 1 0 1 1 0	Increment	Yes	0-4	(SA) + 1 → (SA)
INCT	0 0 0 0 0 1 0 1 1 1	Increment by two	Yes	0-4	(SA) + 2 → (SA)
DEC	0 0 0 0 0 1 1 0 0 0	Decrement	Yes	0-4	(SA) – 1 → (SA)
DECT	0 0 0 0 0 1 1 0 0 1	Decrement by two	Yes	0-4	(SA) – 2 → (SA)
X†	0 0 0 0 0 1 0 0 1 0	Execute	No	–	Execute the instruction at SA.

***Operand is compared to zero for setting the status bit (i.e., before execution).**

†If additional memory words for the execute instruction are required to define the operands of the instruction located at SA, these words will be accessed from PC and the PC will be updated accordingly. The instruction acquisition signal (IAQ) will not be true when the TMS 9900 accesses the instruction at SA. Status bits are affected in the normal manner for the instruction executed.

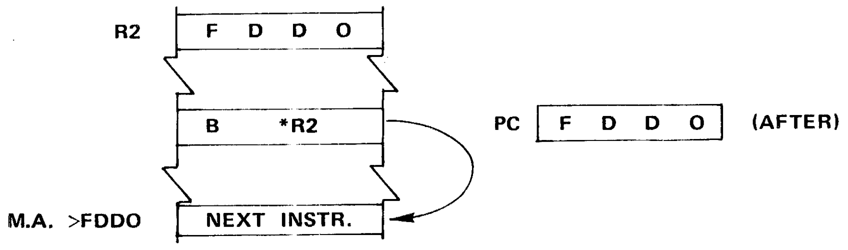
EXAMPLES

(1) *ASSEMBLY LANGUAGE:*

```
B    *R2        BRANCH TO M.A. IN R2
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	1	0	0	0	1	0	1	0	0	1	0	>0442

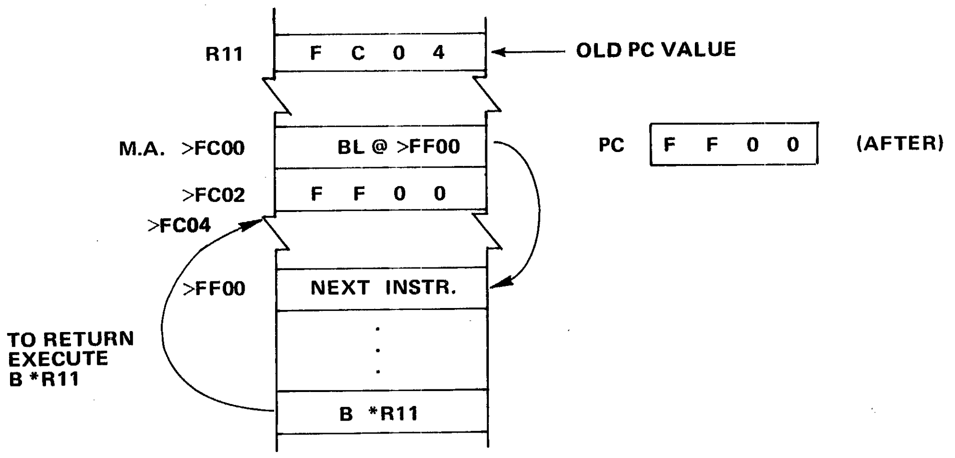


(2) *ASSEMBLY LANGUAGE:*

```
BL     @>FF00       BRANCH TO M.A. >FF00, SAVE OLD PC VALUE (AFTER EXECUTION) IN R11
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	>04A0
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	>FF00



(3) *ASSEMBLY LANGUAGE:*

```
BLWP    @>FD00       BRANCH, GET NEW WORKSPACE AREA
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	>0420
1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	>FD00

This context switch provides a new workspace register file and stores return values in the new workspace. See Figure 4-10. The operand (>FD00 above) is the M.A. of a two-word transfer vector, the first word the new WP value, the second word the new PC value.

4.6.7 FORMAT 7 (RTWP, CONTROL) INSTRUCTIONS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
General Format:	OP CODE											N				

External instructions cause the three most-significant address lines (A0 through A2) to be set to the levels described in the table below and cause the CRUCLK line to be pulsed, allowing external control functions to be initiated. The RSET instruction resets the I/O lines on the TMS 9901 to input lines; the TMS 9902 is not affected. RSET also clears the interrupt mask in the Status Register. The LREX instruction causes a delayed load interrupt, delayed by two IAQ cycles after LREX execution. The load operation gives control to the monitor.

$\overline{\text{CKOF}}$ and $\overline{\text{CKON}}$ can be used by monitoring pins 9 and 10 respectively of U20. See sheet 2 of the schematics in Appendix F.

MNEMONIC	OP CODE	MEANING	STATUS BITS AFFECTED	DESCRIPTION	ADDRESS BUS*
	0 1 2 3 4 5 6 7 8 9 10				A0 A1 A2
IDLE	0 0 0 0 0 0 1 1 0 1 0	Idle	—	Suspend TMS 9900 instruction execution until an interrupt, $\overline{\text{LOAD}}$, or $\overline{\text{RESET}}$ occurs	L H L
RSET	0 0 0 0 0 0 1 1 0 1 1	Reset I/O & SR	12–15	0 → ST12 thru ST15	L H H
CKOF	0 0 0 0 0 0 1 1 1 1 0	User defined		---	H H L
CKON	0 0 0 0 0 0 1 1 1 0 1	User defined		---	H L H
LREX	0 0 0 0 0 0 1 1 1 1 1	Load interrupt		Control to *TIBUG*	H H H
RTWP	0 0 0 0 0 0 1 1 1 0 0	Return from Subroutine	0–15	(R13) → (WP) (R14) → (PC) (R15) → (ST)	

*These outputs from the TMS 9900 go to a SN74LS138 as shown in Figure 5-6

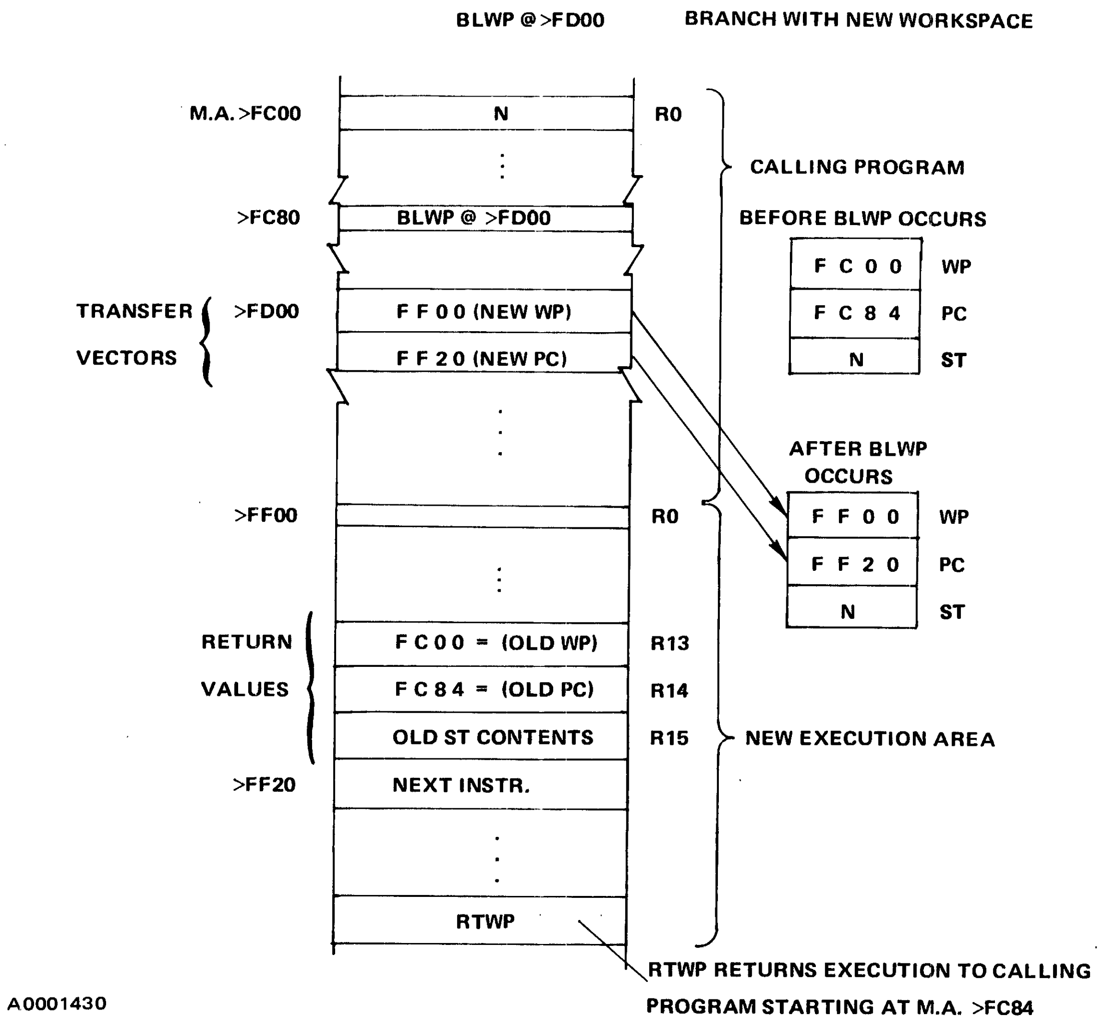


FIGURE 4-10. BLWP EXAMPLE

Essentially, the RTWP instruction is a return to the next instruction that follows the BLWP instruction (i.e., RTWP is a return from a BLWP context switch, similar to the B *R11 return from a BL instruction). BLWP provides the necessary values in registers 13, 14, and 15 (see Figure 4-10).

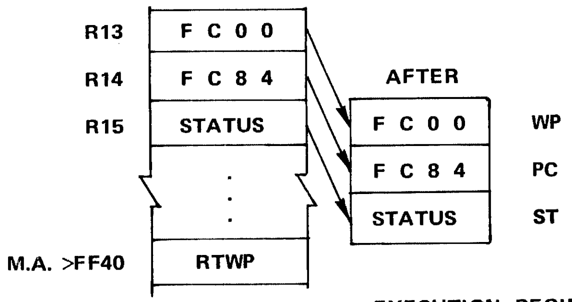
EXAMPLE

ASSEMBLY LANGUAGE:

```
RTWP      RETURN FROM CONTEXT SWITCH
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	>0380



EXECUTION BEGINS AT M.A. >FC84
WITH R0 AT M.A. >FC00.

4.6.8 FORMAT 8 (IMMEDIATE, INTERNAL REGISTER LOAD/STORE) INSTRUCTIONS

4.6.8.1 Immediate Register Instructions

General format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OP CODE											N	R			
IOP															

MNEMONIC	OP CODE 0 1 2 3 4 5 6 7 8 9 10	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
AI	0 0 0 0 0 0 1 0 0 0 1	Add immediate	Yes	0-4	(R) + IOP → (R)
ANDI	0 0 0 0 0 0 1 0 0 1 0	AND immediate	Yes	0-2	(R) AND IOP → (R)
CI	0 0 0 0 0 0 1 0 1 0 0	Compare immediate	Yes	0-2	Compare (R) to IOP and set appropriate status bits
LI	0 0 0 0 0 0 1 0 0 0 0	Load immediate	Yes	0-2	IOP → (R)
ORI	0 0 0 0 0 0 1 0 0 1 1	OR immediate	Yes	0-2	(R) OR IOP → (R)

AND Logic:	0·1, 1·0 = 0	OR Logic:	0 + 1, 1 + 0 = 1
	0·0 = 0		0 + 0 = 0
	1·1 = 1		1 + 1 = 0

4.6.8.2 Internal Register Load Immediate Instructions

General format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OP CODE											N				
IOP															

MNEMONIC	OP CODE 0 1 2 3 4 5 6 7 8 9 10	MEANING	DESCRIPTION
LWPI	0 0 0 0 0 0 1 0 1 1 1	Load workspace pointer immediate	IOP → (WP), no ST bits affected
LIMI	0 0 0 0 0 0 1 1 0 0 0	Load interrupt mask	IOP, bits 12 thru 15 → ST12 thru ST15

4.6.8.3 Internal Register Store Instructions

General format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OP CODE											N	R			

No ST bits are affected.

MNEMONIC	OP CODE	MEANING	DESCRIPTION
	0 1 2 3 4 5 6 7 8 9 10		
STST	0 0 0 0 0 0 1 0 1 1 0 0	Store status register	(ST) → (R)
STWP	0 0 0 0 0 0 1 0 1 0 1	Store workspace pointer	(WP) → (R)

EXAMPLES

(1) ASSEMBLY LANGUAGE:

```
AI    R2,>FF        ADD >FF TO CONTENTS OF R2
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	>0222
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	>00FF

	BEFORE	AFTER
R2	0 0 0 F	0 1 0 E

(2) ASSEMBLY LANGUAGE:

```
CI    R2,>10E       COMPARE R2 TO >10E
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	>0282
0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	0	>010E

R2 contains "after" results (> IOE) of instruction in Example (1) above; thus the ST equal bit becomes set.

(3) ASSEMBLY LANGUAGE:

```
LWPI   >FC00       WP SET AT >FC00 (M.A. OF R0)
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	>02E0
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	>FC00

This is used to define the workspace area in a task, usually placed at the beginning of a task.

(4) *ASSEMBLY LANGUAGE:*

```
STWP    R2      STORE WP CONTENTS IN R2
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	0	1	0	1	0	1	0	0	0	1	0	>02A2

This places the M.A. of R0 in a workspace register.

4.6.9 FORMAT 9 (XOP) INSTRUCTION

Other format 9 instructions (MPY, DIV) are explained in paragraph 4.6.3 (format 3).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
General Format:	0	0	1	0	1	1	D (XOP NUMBER)				T_S		SR			

The T_S and SR fields provide multiple mode addressing capability for the source operand. When the XOP is executed, ST6 is set and the following transfers occur:

$(40_{16} + 4D) \rightarrow (WP)$	First vector at 40_{16}
$(42_{16} + 4D) \rightarrow (PC)$	Each vector uses 4 bytes (2 words)
$SA \rightarrow$ (new R11)	
(old WP) $\rightarrow$ (new WR13)	
(old PC) $\rightarrow$ (new WR14)	
(old ST) $\rightarrow$ (new WR15)	

The TMS 9900 does not test interrupt request ($\overline{\text{INTREQ}}$) upon completion of the XOP instruction.

An XOP is a means of calling one of 16 subtasks available for use by any executing task. The EPROM memory area between M.A. 40_{16} and $7E_{16}$ is reserved for the transfer vectors of XOP's 0 to 15 (see Figure 4-1). Each XOP vector consists of two words, the first a WP value, the second a PC value, defining the workspace pointer and entry point for a new subtask. These values are placed in their respective hardware registers when the XOP is executed.

The old WP, PC, and ST values (of the XOP calling task) are stored (like the BLWP instruction) in the new workspace, registers 13, 14, and 15. Return to the calling routine is through the RTWP instruction. Also stored, in the new R11, is the M.A. of the source operand. This allows passing a parameter to the new subtask, such as the memory address of a string of values to be processed by the XOP-called routine. Figure 4-11 depicts calling an XOP to process a table of data; the data begins at M.A. $FF00_{16}$.

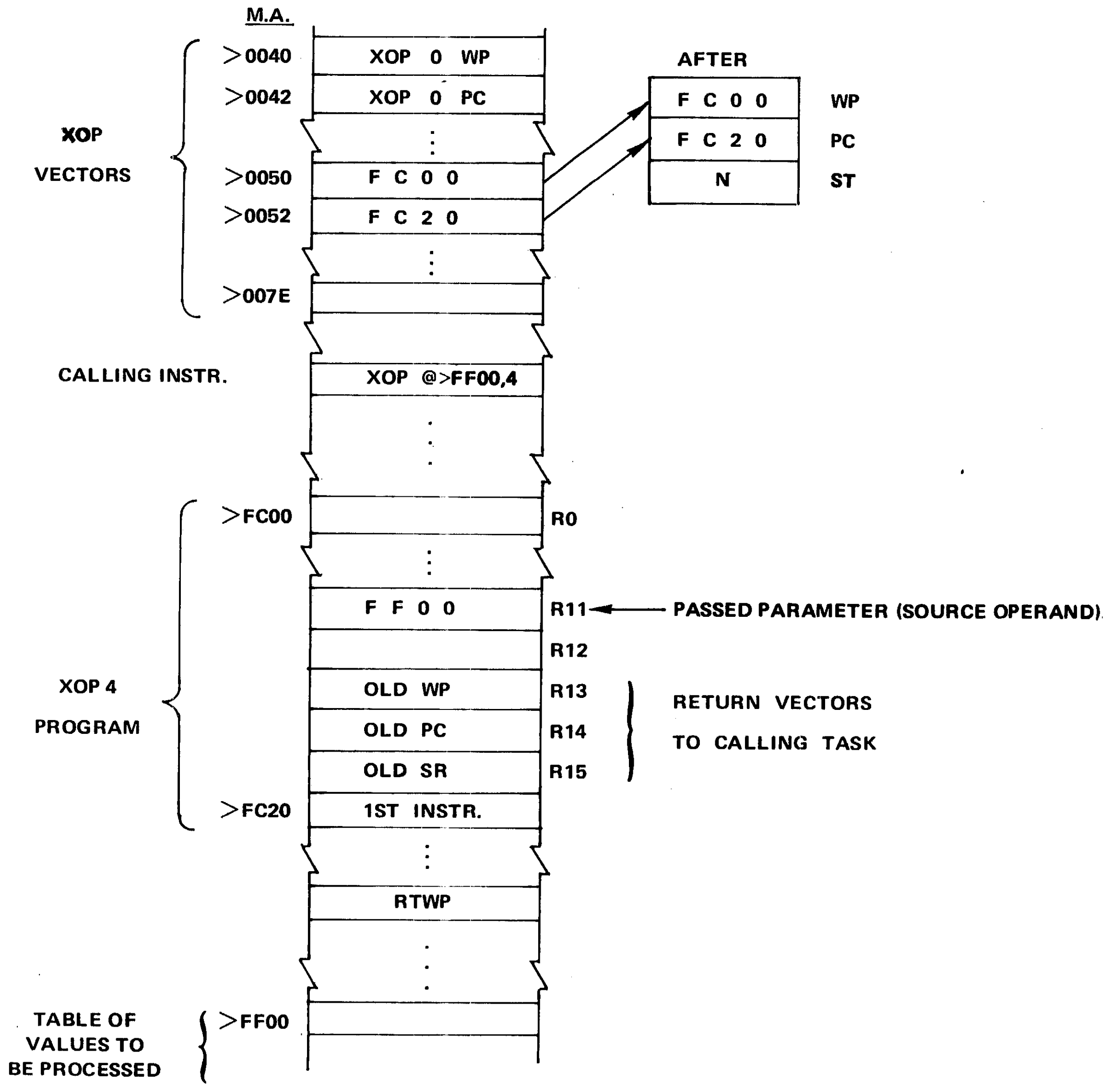
XOP's 0, 1 and 8 to 15 are used by the *TIBUG* monitor, calling software routines (supervisor calls) as requested by tasks. This user-accessible software performs tasks such as write to terminal, convert binary to hex ASCII, etc. These monitor XOP's are discussed in Section 3.3.

ASSEMBLY LANGUAGE:

```
    XOP      @>FF00,4
```

MACHINE LANGUAGE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	1	0	1	1	0	1	0	0	1	0	0	0	0	0	>2D20
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	>FF00



A0001431

FIGURE 4-11. XOP EXAMPLE

4.7 CRU ADDRESSING

The Communications Register Unit (CRU) is the I/O data interface for the TM 990/100M microcomputer. When CRU instructions are executed, data is written or read through the CRUOUT or CRUIN pins respectively of the TMS 9900 to or from designated devices addressed via the address bus of the microprocessor.

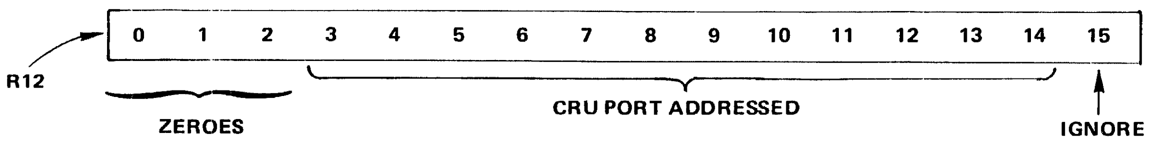
The CRU address is maintained in register 12 of the workspace register area. Only bits 3 through 14 of the register are interpreted by the CPU for the desired CRU address, and this 12-bit value is called the CRU base address.

TM 990/100M devices driven off of the CRU interface include the TMS 9901 parallel interface and the TMS 9902 serial interface which are accessed through the CRU addresses noted in Table 4-5. This table also lists the functions of the other CRU addresses which can be used for on-card or off-card I/O use. Addressing the TMS 9901 and TMS 9902 for use as interval timers is explained, along with programming examples, in section 4.10. Further detailed information on these two devices can be obtained from their respective data manuals.

The five instructions that program the CRU interface are:

- LDCR Load from memory a pattern of 1 to 16 bits and serially transmit this pattern through the CRUOUT pin of the TMS 9900 (paragraph 4.6.4).
- STCR Store into memory a pattern of 1 to 16 bits obtained serially at the CRUIN pin of the TMS 9900 (paragraph 4.6.4).
- SBO Set a CRU bit to a logical one; essentially, this sends a logical one through the CRUOUT pin of the TMS 9900 (paragraph 4.6.2.2).
- SBZ Set a CRU bit to a logical zero; essentially, this sends a logical zero through the CRUOUT pin of the TMS 9900 (paragraph 4.6.2.2).
- TB Test a CRU bit; essentially, this tests the value at the CRUIN pin of the TMS 9900, and the test results are reflected in the equal bit of the Status Register (paragraph 4.6.2.2).

To execute any of these five instructions, a CRU address must be present in register 12; this value can be loaded by software; e.g., use a load immediate instruction (LI). Although the register is a full 16 bits, only bits 3 through 14 are used to contain the CRU base address. Bits 0, 1, 2, 3, are zeroes and bit 15 is ignored as shown below:



The LDCR and STCR instructions use a byte or word of memory depending respectively if 1 to 8 bits or more than 8 bits are to be loaded or stored. In STCR instructions, the right bits of the memory area are used for storage, and unused left-side bits are zero filled. Figure 4-12 depicts an LDCR instruction using a byte of memory. Figure 4-13 depicts an STCR instruction using a word of memory.

The TB, SBO, and SBZ instructions use a displacement of +127 bits and −128 bits from the CRU bit designated in bits 3 to 14 of R12. Thus, if bit 300_{16} is designated in R12, bits 3 to 14, the following assembly language instructions and comments would apply:

```
TB        >10      TEST CRU BIT >310
SBO       -1       SET CRU BIT >2FF TO  ONE
SBZ       16       SET  CRU BIT >310 TO  ZERO
```

The LDCR and STCR instructions address the CRU using the value in R12; these instructions do not have the advantage of specifying a displacement from the R12 value such as used by the CRU bit instructions. If it is necessary to change the CRU address, it is important to understand that only bits 3 to 14 need be modified. For example, if it is desired to load (LDCR) successive groups of 16 CRU ports, a value of 32 (not 16) must be added to the contents of R12 for each group in order to accurately change the contents of R12 bits 3 to 14 (AI R12,32). An alternate method would be to load a new value into R12 (LI R12, > 200; LI R12, > 210; etc.).

TABLE 4-5. CRU ADDRESS MAP

CONTENTS OF R12 (BITS 0 TO 15)	CRU BASE ADDRESS (R12, BITS 3 TO 14)	FUNCTION
0000_{16} to $007E_{16}$	00_{16} to $3F_{16}$	Reserved, on-card expansion
0080_{16} to $00BE_{16}$	40_{16} to $5F_{16}$	TMS 9902, on-card serial I/O Interface, timer
$00C0_{16}$ to $00FE_{16}$	60_{16} to $7F_{16}$	Reserved, on-card expansion
0100_{16} to $013E_{16}$	80_{16} to $9F_{16}$	TMS 9901, on-card 16 I/O parallel interface, interrupt status register, interrupt mask register, interval timer
0140_{16} to $01FE_{16}$	$A0_{16}$ to FF_{16}	Reserved, on-card expansion
200_{16} to $1FFE_{16}$	100_{16} to FFF_{16}	Off-card CRU lines

```
LI    R12,>200      LOAD CRU BASE ADDRESS >100 IN BITS 3 TO 14 OF R12
LDCR    R5,6       6 BITS TO CRU
```

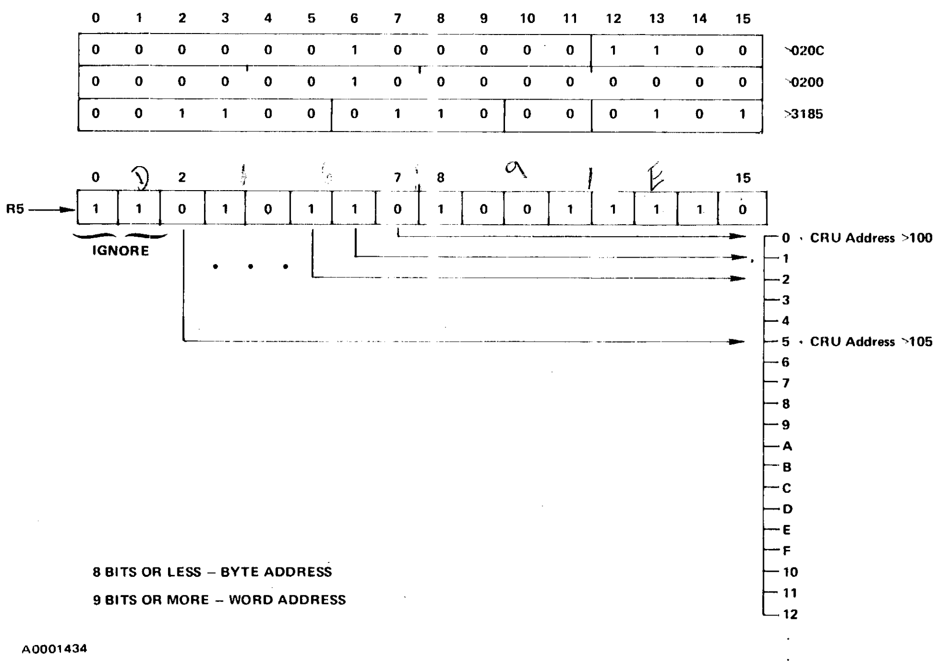
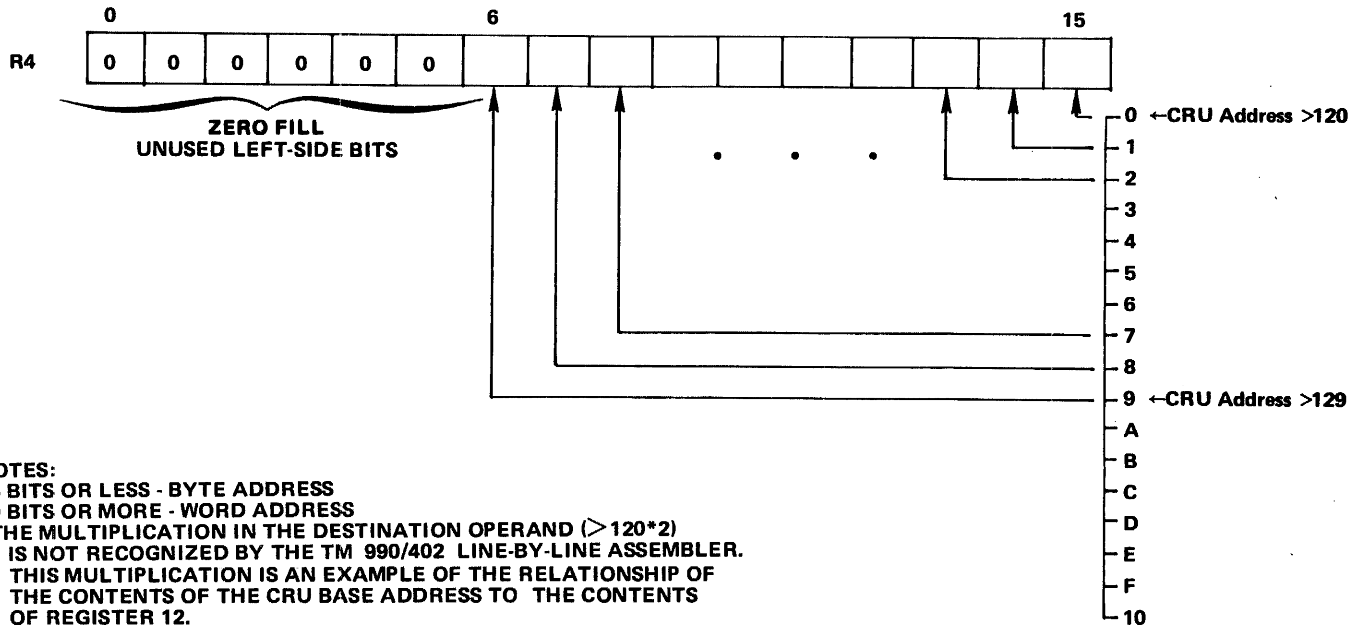


FIGURE 4-12. LDCR BYTE INSTRUCTION

```
LI      R12,>120*2   LOAD CRU BASE ADDRESS >120 IN BITS 3 TO 14 OF R12
STCR    R4,10          10 BITS FROM CRU TO R4
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	>020C
0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	>0240
0	0	1	1	0	1	1	0	1	0	0	0	0	1	0	0	>3684



NOTES:
8 BITS OR LESS - BYTE ADDRESS
9 BITS OR MORE - WORD ADDRESS
THE MULTIPLICATION IN THE DESTINATION OPERAND (>120*2)
IS NOT RECOGNIZED BY THE TM 990/402 LINE-BY-LINE ASSEMBLER.
THIS MULTIPLICATION IS AN EXAMPLE OF THE RELATIONSHIP OF
THE CONTENTS OF THE CRU BASE ADDRESS TO THE CONTENTS
OF REGISTER 12.

A0001436

FIGURE 4-13. STCR WORD INSTRUCTION

4.8 COMPARISON OF JUMPS, BRANCHES, XOP'S

This comparison is shown in Table 4-6.

4.9 INTERRUPTS

4.9.1 INTERRUPT OPERATION

The TM 990/100M employs 16 interrupt levels with level 0 the highest priority and level 15 the lowest priority. Level 0 is reserved for the reset function. Reset, which can be initiated by the RESET pushbutton (Figure 1-2) or by remote activation of the $\overline{\text{PRES}}$ signal, places the board under monitor control.

TABLE 4-6. COMPARISON OF JUMPS, BRANCHES, XOP'S

MNEMONIC	PARAGRAPH	DEFINITION SUMMARY
JMP	4.6.2	One-word instruction, destination restricted to +127, −128 words from Program Counter value.
B	4.6.6	Two-word instruction, branch to any memory location.
BL	4.6.6	Same as B with PC return address in R11.
BLWP	4.6.7	Same as B with new workspace; old WP, PC and ST contents (return vectors) are in new R13, R14, R15.
XOP	4.6.9	Same as BLWP with address of parameter (source operand) in new R11. Sixteen XOP vectors outside program in M.A. 40_{16} to $7E_{16}$; can be called by any program.

Interrupts are controlled by the TMS 9901 interface which polls interrupt signals from 15 input lines ($\overline{\text{INT1}}$ to $\overline{\text{INT15}}$), determines the priority of the incoming signal, and sends a four-bit code of the highest priority interrupt to the TMS 9900 along with an interrupt request ($\overline{\text{INTREQ}}$). The four-bit code is sent on lines IC0 to IC3.

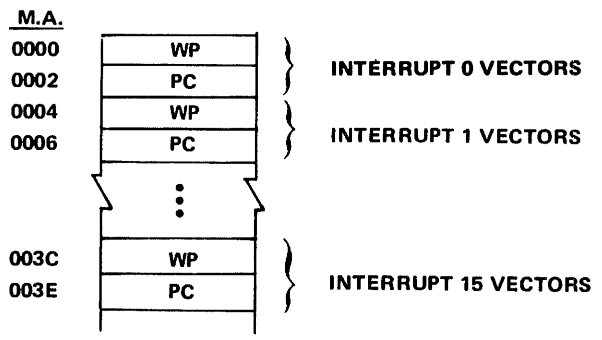
The TMS 9900 compares the level of incoming interrupt request to the interrupt mask in the least significant four bits (12 to 15) of the Status Register. If the level of the incoming interrupt is equal to or less than the value in the Status Register mask, a context switch takes place similar to a BLWP instruction (paragraph 4.6.6). A pair of vector addresses (the new WP and PC values) are obtained from one of the 16 interrupt traps in EPROM (M.A. 0000_{16} to $003E_{16}$), as shown in Figure 4-14. Then the following takes place:

- The current WP, PC, and ST contents are saved.
- The new values from the interrupt vectors are placed in the WP and PC hardware registers.
- The old WP, PC, and ST values are placed respectively in R13, R14, and R15 of the new workspace.
- A value of one less than the new interrupt value is placed in the ST interrupt mask (bits 12 to 15).
- Execution begins and continues until another interrupt of higher priority occurs or until a return instruction is executed (RTWP).

If a higher priority interrupt occurs, a second interrupt context switch takes place after at least one instruction is executed of the first interrupt. This allows execution of a LIMI instruction to inhibit other interrupts. Completion of the second interrupt passes control back to the first interrupt.

4.9.2 PROGRAMMABLE INTERRUPTS

Interrupt traps 0, 3, and 4 contain vector values burned into EPROM. Interrupts 3 and 4 can be programmed by the user.



A0001432

FIGURE 4-14. INTERRUPT TRAP LOCATIONS

- Interrupt trap 0 is used for the reset function. This is not a user programmable interrupt.

- Interrupt trap 3 is the real time clock utilized by programming the TMS 9901 using CRU instructions. This programming is shown in the *TMS 9901 Programmable Systems Interface Data Manual.* Vectors in interrupt trap 3 are $FF68_{16}$ for the WP vector and $FF88_{16}$ for the first of a two-word instruction to be inserted in RAM by the user. See Figure 4-15. This two-word instruction area could contain a B or BL instruction as discussed in paragraph 4.6.6. The branch would be to the start of a subroutine set up to handle the interrupt. The subroutine would return to the interrupted program with the RTWP instruction, using the return values in R13, R14, and R15 of the interrupt workspace.

- Interrupt trap 4 originates from the $\overline{\text{INT}}$ output of the TMS 9902 as shown in the *TMS 9902 Asynchronous Communication Controller Manual.* A moveable plug (J11) allows this signal to be routed either to connector P1 or to the TMS 9901 as input for interrupt 4 as shown in the Schematics (Appendix F). Vectors in trap 4 are to $FF8C_{16}$ for the workspace and to $FFAC_{16}$ for the first of a two-word instruction. The user can fill these RAM locations as desired. See Figure 4-15.

Four conditions causing $\overline{\text{INT}}$ to be active (low causing interrupts to occur) are as follows:

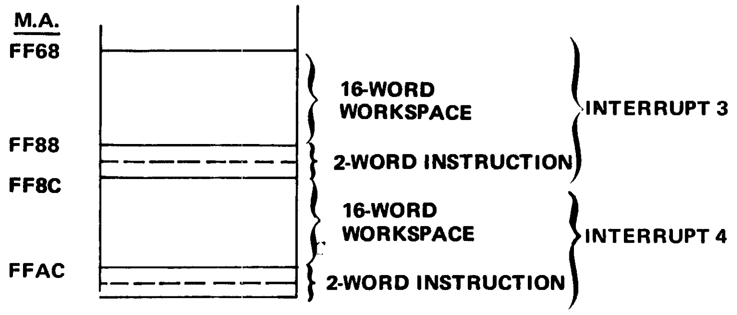
- TMS 9902 CRU bit 21 a one and a data set status change (DSCH) occurs.

- TMS 9902 CRU bit 20 a one and timer elapses (TIMELP)

- TMS 9902 CRU bit 19 a one and the transmit buffer is empty (XBIENB).

- TMS 9902 CRU bit 18 a one and the receive buffer is loaded (RIENB).

If the user desires to fill interrupt trap locations (M.A. $000C_{16}$ to 0012_{16}) with his own vector values, he must reburn the EPROM with the desired values.



A0001433

FIGURE 4-15. DEDICATED INSTRUCTION AND WORKSPACE AREAS FOR INTERRUPTS 3 AND 4

4.10 PROGRAMMING THE INTERVAL TIMERS

Two interval timers are available to the TM 990/100M; one from the TMS 9901 and one from the TMS 9902. Detailed information on these two devices can be found in the respective data manuals for the TMS 9901 and TMS 9902.

Both interval timers can be programmed to cause interrupts at the TMS 9900:

- To trap 3 for the TMS 9901
- To trap 4 for the TMS 9902

4.10.1 TMS 9901 INTERVAL TIMER

A detailed discussion of the TMS 9901 interval timer can be found in the TMS 9901 data manual. There are several possible sequences of coding that can program and enable the interrupt 3 interval timer, and since the timer has a maximum period of 349 milliseconds before issuing an interrupt, the programmer must decide whether to set the interval period in the calling program or in the code handling the interrupt. If the interrupt period desired is longer than 349 milliseconds, then it may be advantageous to reset the timer in the interrupt subroutine which also triggers the interrupt and returns control back to the interrupted program. In any case, the timer must be initially set and triggered following the general sequence below:

(1) Set the CRU address of the TMS 9901 in bits 3 to 14 of R12.

(2) Enable the clock interrupt at the TMS 9901 (interrupt 3).

(3) Set the Status Register interrupt mask to a value of 3 or greater.

(4) Set a register to the value of the interval desired (bits 1 to 14) with bit 15 set to one to enable the clock as shown in Figure 4-16. This figure shows the code and a representation of the CRU for setting a time of 250 milliseconds and for setting the TMS 9901 to the clock mode. The first bit serially brought in on the CRU will be a value of one in bit 15 of the register which sets the TMS 9901 to the clock mode; successive bits (1 to 14) then set the clock interval value. The final bit brought in triggers the timer.

(5) When the interrupt occurs, the interrupt handler must reset the interrupt at the TMS 9901 before returning to the interrupted program.

The clock decrements the value set in step (4) at the rate of $\phi/64$ (approximately 46,875 Hz with a 3 MHz clock). The maximum interval register value of all ones in 14 bits (16,383) takes approximately 349 milliseconds to decrement to zero.

The timer can also be started and stopped, then the timer register bits read with an STCR instruction to determine the elapsed time (elapsed bit count divided by 46,875 equals elapsed time in seconds).

The code in Figure 4-17 is an example of a code to set up and call the TMS 9901 interval timer and also the code of the interrupt handling subroutine. Note that the calling program first clears the counting register (R0) of the interrupt workspace. Then it sets up the interrupt masks at the TMS 9901 and TMS 9900 after setting the TMS 9901 CRU address in R12. Then the calling program sets an initial value in the timer register (CLK1 to CLK14 as shown in the TMS 9901 data manual). Because the desired output on the terminal is a message every 15 seconds, a minimum interval is set in the calling program while the interrupt handler is responsible for setting the time and clearing the interrupt after it occurs. The handler keeps a count of the intervals to determine the 15 seconds. Since interrupt 3 causes a context switch to the WP and PC areas shown in Figure 4-15, a branch to the handler is first placed in the RAM instruction area shown for interrupt 3. The interrupt will continually interrupt the executing program with return values to that program stored in R13 to R15 of the interrupt workspace. Assembled code is shown for the TM 990/402 line-by-line assembler as well as the PXRASM assembler.

4.10.2 TMS 9902 INTERVAL TIMER

The TMS 9902 interval timer is programmable through the CRU, but it requires a different sequence of events than for the TMS 9901 timer. A detailed discussion of the TMS 9902 interval timer can be found in the TMS 9902 data manual. The interval register of the TMS 9902 can contain a maximum value of FF_{16}, providing a maximum interval of 16.32 milliseconds at an internal clock frequency of 1 MHz. The interrupt is routed to the TMS 9900 through $\overline{INT4}$ of the TMS 9901; thus the interrupt masks of both these devices must be programmed. J1 must be in the "9902" position to route interrupts from the TMS 9902 to the microprocessor via the TMS 9901; code to run the TMS 9902 interval timer generally follows the following sequence:

(1) Set the TMS 9901 CRU address in R12 and enable interrupt 4 at that device.

(2) Set the Status Register interrupt mask to a value of 4 or greater.

(3) Set the TMS 9902 CRU address in R12.

(4) Reset the TMS 9902; this sets LDIR and LDCTRL.

(5) Load the interval timer contents (ITC) on the CRU (bits 0 to 7); ITC/15,625 = interval time in seconds at an internal clock of 1 MHz.

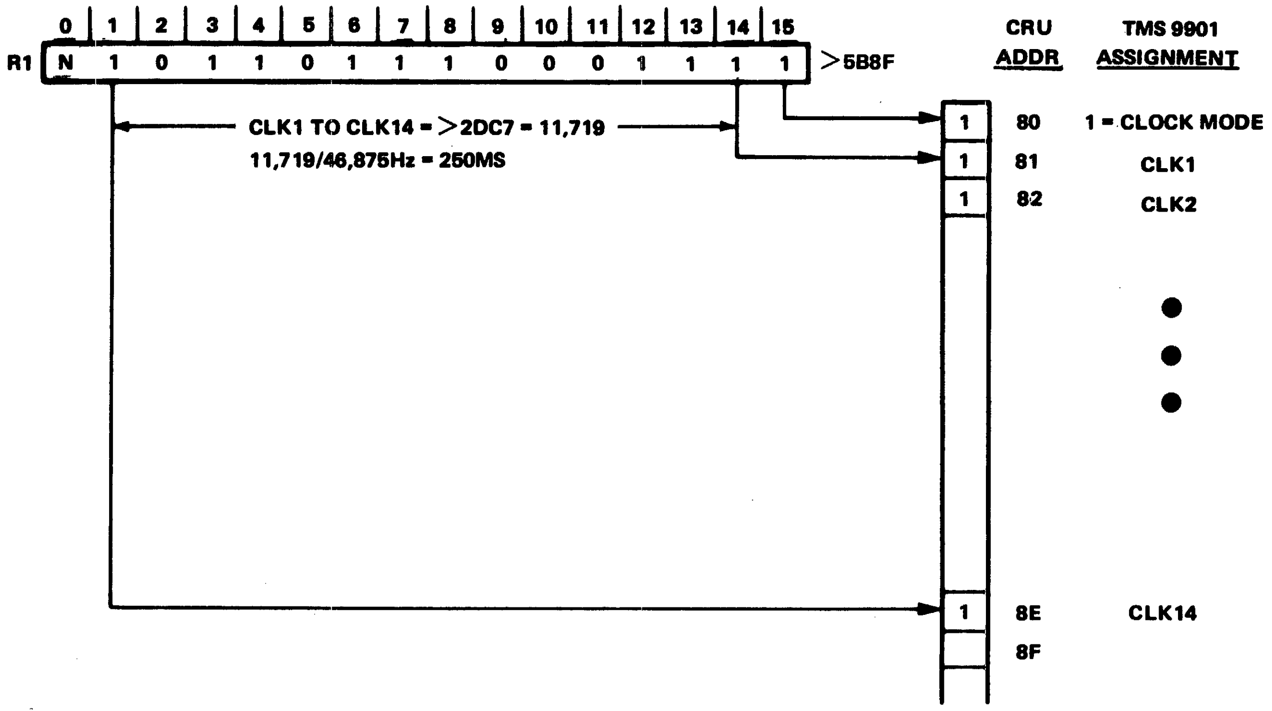
(6) Set TIMENB (CRU bit 20) to ready interrupt and reset TIMELP and TIMERR.

(7) Reset LDCTRL to zero (CRU bit 14).

(8) Set LDIR to a one (CRU bit 13) to begin loading the interval register. When loaded, LDIR is reset.

(9) Set LDIR to begin timer countdown.

```
LI      R12, 2*>80    CRU ADDRESS OF TMS 9901 (2 X >80 = >100)
LI      R1, >5B8F     CLOCK, >2DC7 COUNTS,
LDCR    R1, 15        SET CLOCK VALUE AT CLOCK REGISTER
```



NOTE:
THE FIRST SERIAL INPUT FROM CRU (A ONE IN BIT 15 OF R1) SETS CLOCK MODE.
LAST INPUT TO CLOCK REGISTER (CLK1 TO CLK14) STARTS THE CLOCK.

A0001436

FIGURE 4-16. ENABLING AND TRIGGERING TIMS 9901 INTERVAL TIMER

ASSEMBLED USING PXRASM ON 990/4 COMPUTER

```
                   *   *    *    *    *    *    *    *    *    *
                   *   THIS PROGRAM CAUSES AN INTERRUPT THROUGH INT3    *
                   *   EVERY 15 SECONDS USING THE INTERVAL TIMER IN THE *
                   *   TMS 9901. THE AORG OPCODE IS A DIRECTIVE TO THE  *
                   *   PXRASM ASSEMBLER TO GENERATE A TAG CHARACTER AND *
                   *   CODE TO LOAD AT AN ABSOLUTE ADDRESS. J.WALSH 5-77 *
                   *     *    *    *    *    *    *    *    *    *
                         IDT  'INT9901'
                   *
                   * PROGRAM CALLING THE INTERRUPT
                   *
FD00                     AORG >FD00              SET ABSOLUTE ADDRESS
FD00  02E0               LWPI >FD20              DEFINE WORKSPACE
FD02  FD20
FD04  04E0               CLR  @>FF68             CLEAR INTERRUPT REGISTER 0
FD06  FF68
FD08  020C               LI   R12,2*>80          SET 9901 CRU ADDRESS
FD0A  0100
FD0C  1E00               SBZ  0                  9901 TO INTERRUPT MODE
FD0E  1D03               SBO  3                  ENABLE INTERRUPT 3
FD10  0300               LIMI 3                  SET STATUS REGISTER INTERRUPT

FD12  0003
FD14  0201               LI   R1,3               CLOCK COUNT 1, CLOCK MODE
FD16  0003
FD18  33C1               LDCR R1,15              SET CLOCK COUNT,, ENABLE COUNT
FD1A  10FF               JMP  $                  LOOP AT THIS LOCATION
                   *
                   * INTERRUPT PROGRAM
                   *
FE00                     AORG >FE00              SET ABSOLUTE VALUE
FE00  0280               CI   R0,60              IS COUNT = 60 = 15 SECONDS?
FE02  003C
FE04  130B               JEQ  $+24               YES, PRINT MESSAGE
FE06  0580               INC  R0                 NO, INCREMENT COUNTER
FE08  020C               LI   R12,2*>80          SET 9901 CRU ADDRESS
FE0A  0100
FE0C  0201               LI   R1,>5B8F           CLOCK COUNT OF 11,719
FE0E  5B8F
FE10  33C1               LDCR R1,15              COUNT TOO 9901, ENABLE TIMER
FE12  1E00               SBZ  0                  9901 INTERRUPT MODE
FE14  1D03               SBO  3                  CLEAR INTERRUPT 3
FE16  0300               LIMI 3                  RESET INTERRUPT MASK AT 9900
FE18  0003
FE1A  0380               RTWP RETURN TO PROGRAM
FE1C  2FA0               XOP  @>FE26,14          WRITE MESSAGE
FE1E  FE26
FE20  04C0               CLR  R0                 RESET TIMER REGISTER
FE22  0460               B    @>FE00             REINVOKE INTERRUPT
FE24  FE00
FE26    31               TEXT '15 SECONDS HAVE ELAPSED.'
FE3E  0707               DATA >0707,>0707        BELLS
FE40  0707
FE42    00               BYTE 0                  END OF MESSAGE
                   *
                   *   INSTRUCTION IN INTERRUPT RAM AREA
                   *
FF88                     AORG >FF88
FF88  0460               B    @>FE00             GO TO INTERRUPT ROUTINE
FF8A  FE00
                         END                     DIRECTIVE TO PXRASM ASSEMBLER
```

ASSEMBLED USING TM 990/402 LINE-BY-LINE ASSEMBLER ON TM 990/100M

```
?R
W=80C4
P=0062  9E8
?E
FD00 02E0 LWPI >FD20
FD02 FD20
FD04 04E0 CLR @>FF68
FD06 FF68
FD08 020C LI R12,>100
FD0A 0100
FD0C 1E00 SBZ 0
FD0E 1D03 SBO 3
FD10 0300 LIMI 3
FD12 0003
FD14 0201 LI R1,3
FD16 0003
FD18 33C1 LDCR R1,15
FD1A 10FF JMP >FD1A
FD1C      /FE00
FE00 0280 CI R0,60
FE02 003C
FE04 130B JEQ $+24
FE06 0580 INC R0
FE08 020C LI R12,>100
FE0A 0100
FE0C 0201 LI R1,>5B8F
FE0E 5B8F
FE10 33C1 LDCR R1,15
FE12 1E00 SBZ 0
FE14 1D03 SBO 3
FE16 0300 LIMI 3
FE18 0003
FE1A 0380 RTWP
FE1C 2FA0 XOP @>FE26,14
FE1E FE26
FE20 04C0 CLR R0
FE22 0460 B @>FE00
FE24 FE00
FE26 3135 $15 SECONDS HAVE ELAPSED.
FE28 2053
FE2A 4543
FE2C 4F4E
FE2E 4453
FE30 2048
FE32 4156
FE34 4520
FE36 454C
FE38 4150
FE3A 5345
FE3C 442E
FE3E 0707 +>0707
FE40 0707 +>0707
FE42 0000 +0
FE44      /FF88
FF88 0460 B @>FE00
FF8A FE00
FF8C
```

MEMORY ADDRESS

MACHINE CODE

NOTE

THESE PROGRAMS WERE ASSEMBLED FOR EXECUTION ON A BOARD WITH EXPANSION RAM. FOR EXECUTION ON A NON-EXPANDED SYSTEM, ABSOLUTE MEMORY ADDRESSES MUST BE >FE00 OR GREATER AND REFERENCES TO THE RESULTING ABSOLUTE ADDRESSES MUST BE UPDATED FROM THE ABOVE EXAMPLES.

A0001437

FIGURE 4-17. EXAMPLE OF CODE TO RUN TMS 9901 INTERVAL TIMER

(10) When the interval timer has counted down to zero, the interrupt $(\overline{\text{INT}})$ is sent via jumper J1 to interrupt 4 of the TMS 9901.

NOTE

This interrupt should not be routed to the TMS 9901 from the TMS 9902 while under the monitor as explained in paragraph 6.6. If J1 is in the P1-18 position, the interrupt signal will be routed from connector P1, pin 18.

4.11 CONTEXT SWITCH TO ANOTHER PROGRAM SUCH AS MONITOR

By manipulating registers 13, 14, 15 and executing the RTWP instruction, execution can branch from one program to another, such as a user program to the *TIBUG* monitor. The following is code to branch into the monitor.

```
LI      R13,>FFB0     WP VALUE OF MONITOR
LI      R14,>80       PC VALUE OF MONITOR
LI      R15,0
RTWP
```

NOTE

The above example shows how to branch into a program using the RTWP instruction; it also branches into the monitor. Other more convenient methods to branch to the monitor include the following:

```
BLWP @> FFFC   MONITOR VECTORS AT M.A.> FFFC
```

or

```
B    @> 80     BRANCH DIRECTLY TO MONITOR ENTRY POINT
```

SECTION 5

THEORY OF OPERATION

5.1 GENERAL

This section covers theory of operation of the TM 990/100M. Information in the following manuals can be used to supplement material in this section:

- *TMS 9900 Microprocessor Data Manual*
- *TMS 9901 Programmable Systems Interface Data Manual*
- *TMS 9902 Asynchronous Communication Controller*

Figure 5-1 shows data flow within the TMS 990/100M, highlighting the four major buses:

- Address Bus
- Control Bus
- Data Bus
- Communications Register Unit Bus

5.2 SYSTEM CLOCK (Figure 5-2)

System timing is regulated by a crystal-controlled TMS 9904 clock driver. The tank circuit, shown in Figure 5-2, is tuned to the third harmonic (48 MHz) of the crystal frequency (16 MHz).

5.3 CENTRAL PROCESSING UNIT (Figures 5-3 to 5-6)

The TMS 9900 microprocessor is the central processing unit (CPU for the TM 990/100M. The processor's responsibilities include:

- Memory and bus control
- Instruction acquisition and interpretation
- Timing
- System initialization
- CRU programming

Figure 5-3 groups TMS 9900 pins by function. The address bus addresses devices such as the TMS 9901 and TMS 9902 as well as memory locations. Data is transferred to and from memory as 16-bit words. Interrupt requests and the interrupt level code (IC0 to IC3) come from the TMS 9901 interface.

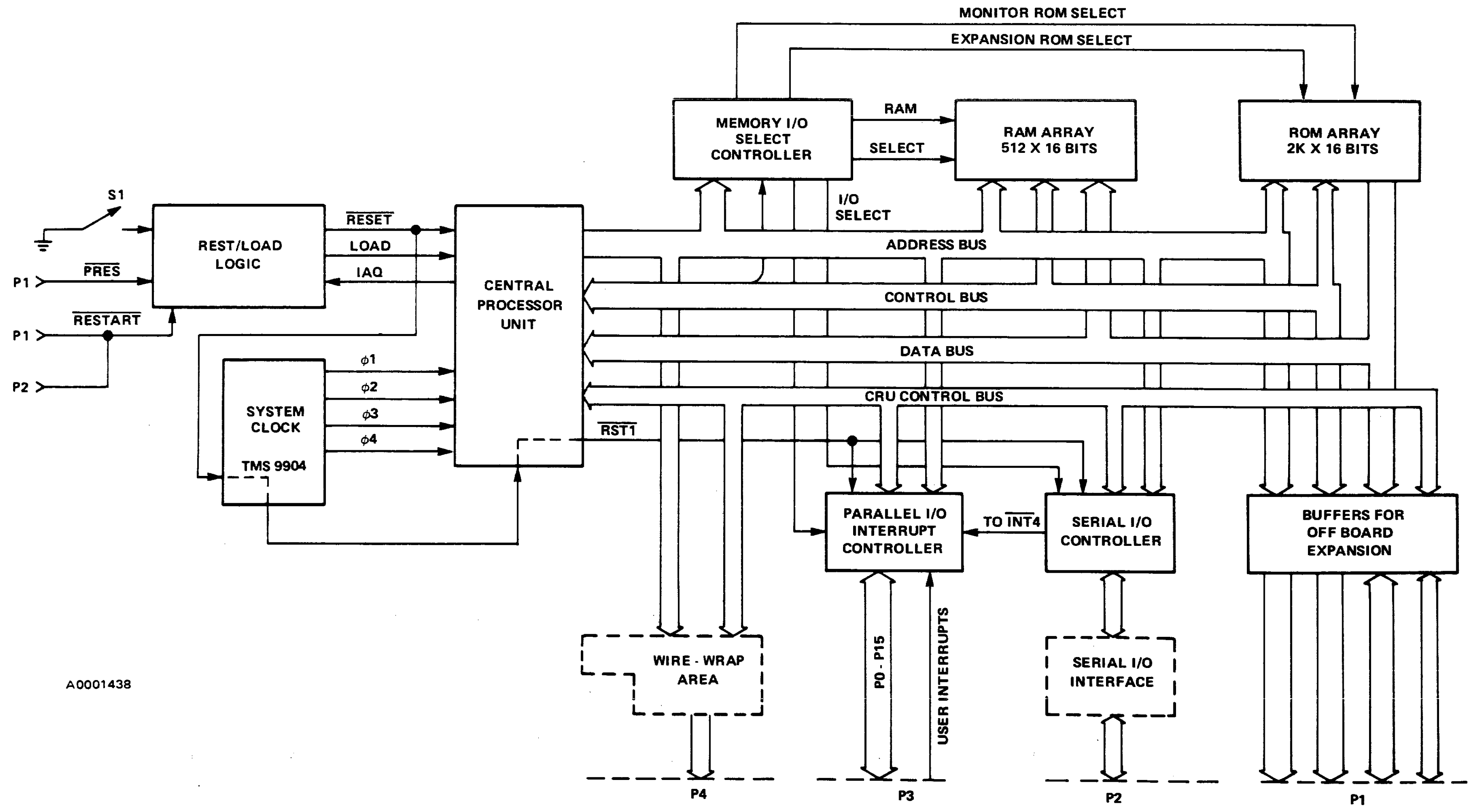


FIGURE 5-1. TM 990/100M BLOCK DIAGRAM

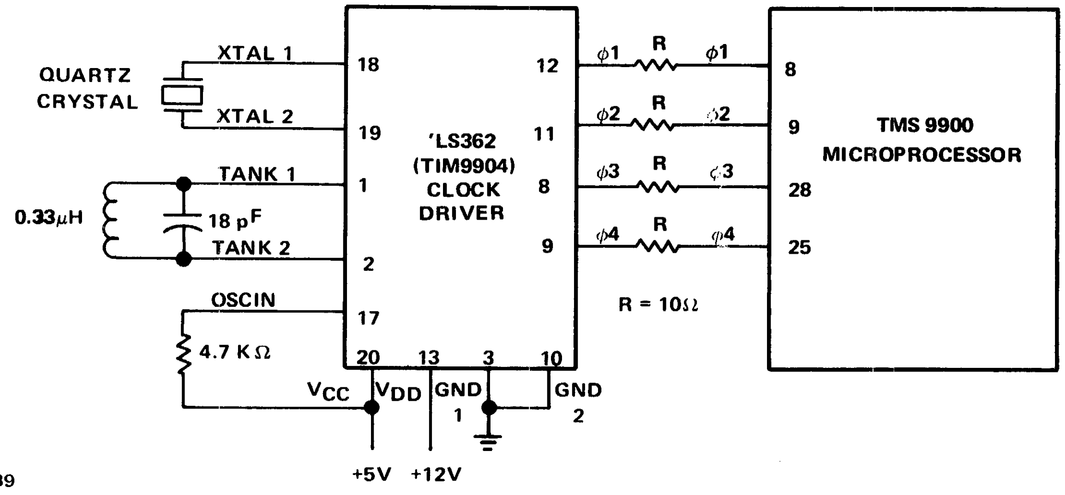


FIGURE 5-2. CRYSTAL-CONTROLLED OPERATION

CRU input instructions (STCR, TB) sample bits on CRUIN while CRU output instructions (LDCR, SBO, SBZ) place serial outputs on CRUOUT. CRU instructions also program the TMS 9901 and TMS 9902 as explained respectively in paragraphs 5.9 and 5.10 (examples are shown in paragraph 4.10).

Other signals are explained in detail in the *TMS 9900 Microprocessor Data Manual.*

Figures 5-4 and 5-5 show the data and address flow within the TMS 9900.

Figure 5-6 shows the logic of three instructions that can be user defined.

5.4 RESET AND LOAD (Figure 5-7)

The reset function resets the processor and TMS 9901, inhibiting memory write and the CRU clock. An interrupt occurs that resets the Status Register and begins execution under the monitor. Reset can occur in two ways:

- Actuating the RESET pushbutton on the card.
- Setting $\overline{\text{PRES}}$.B to a logic ZERO state through connector P1. This signal can generate a power-up reset by inserting a 39 μF tantalum electrolyte cpacitor as shown in the left side of Figure 5-7 and in the lower right of Figure 7-2.

The load function causes an interrupt to WP and PC vectors respectively at $FFFC_{16}$ and $FFFE_{16}$. It is implemented two ways:

- Executing the software instruction LREX.
- Setting $\overline{\text{RESTART}}$ to logic zero through connector P1. This can be used to generate a powerup load by inserting a 39 μF tantalum electrolyte capacitor as shown in the left side of Figure 5-7 and in the lower right of Figure 7-2.

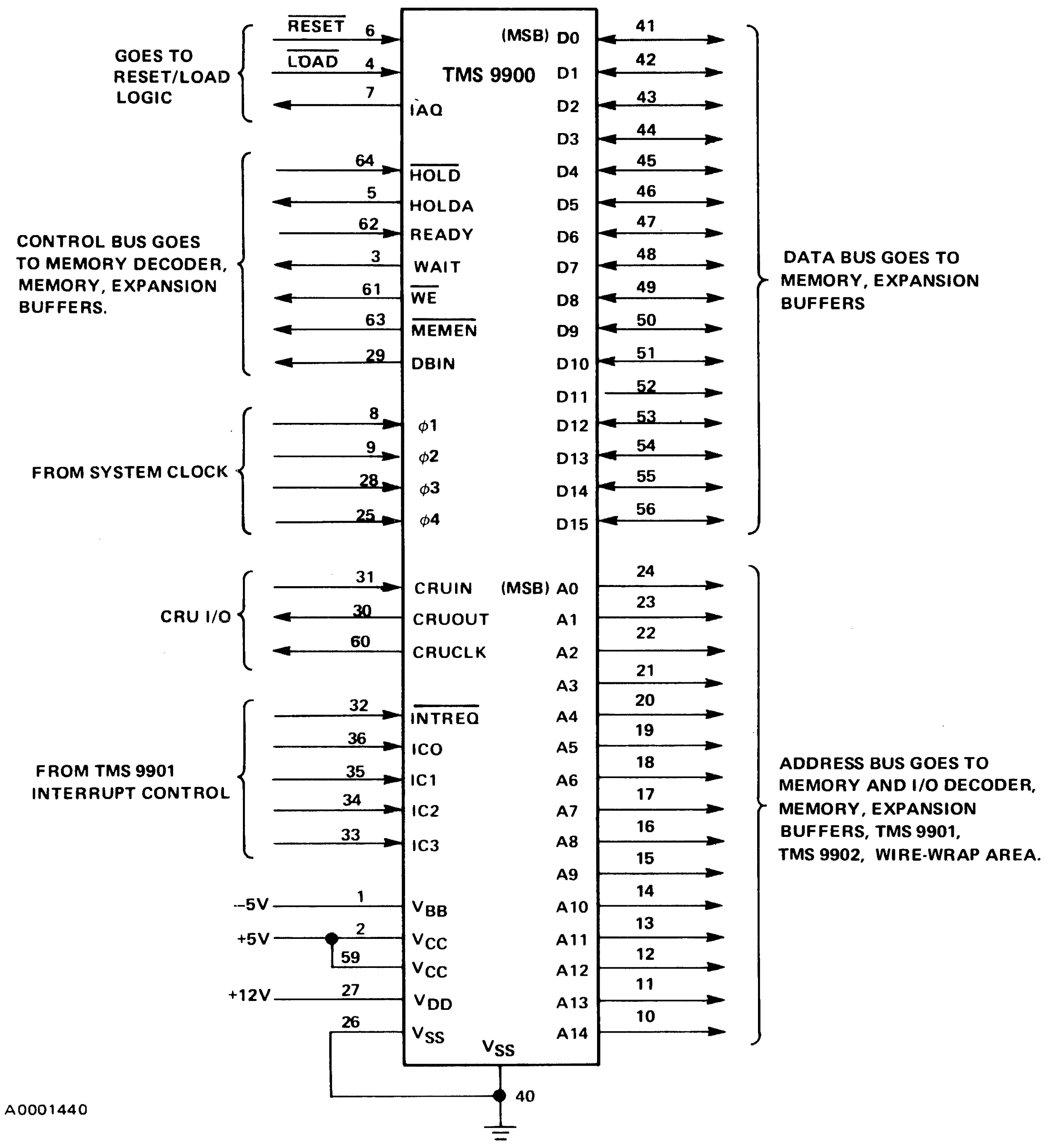
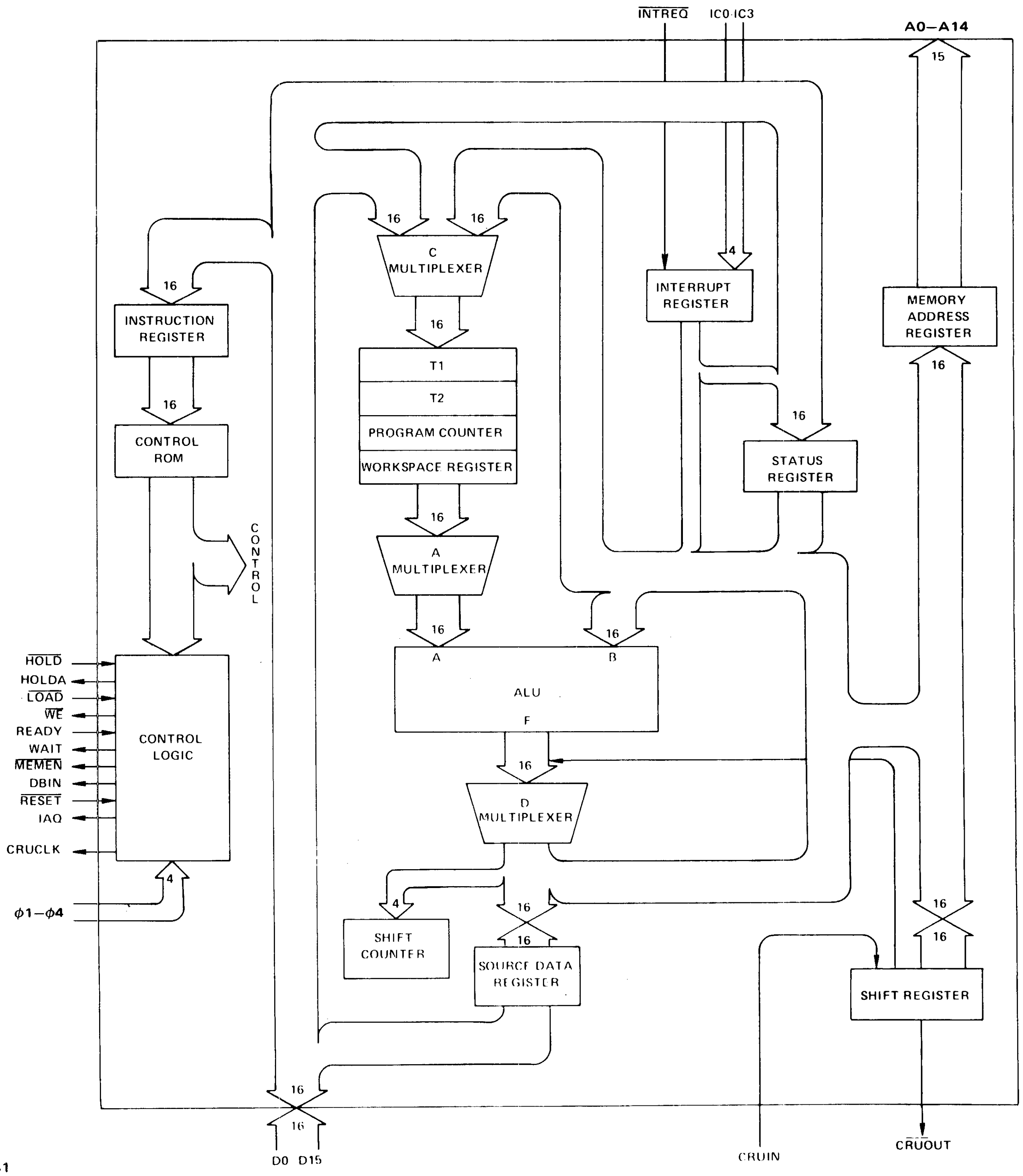
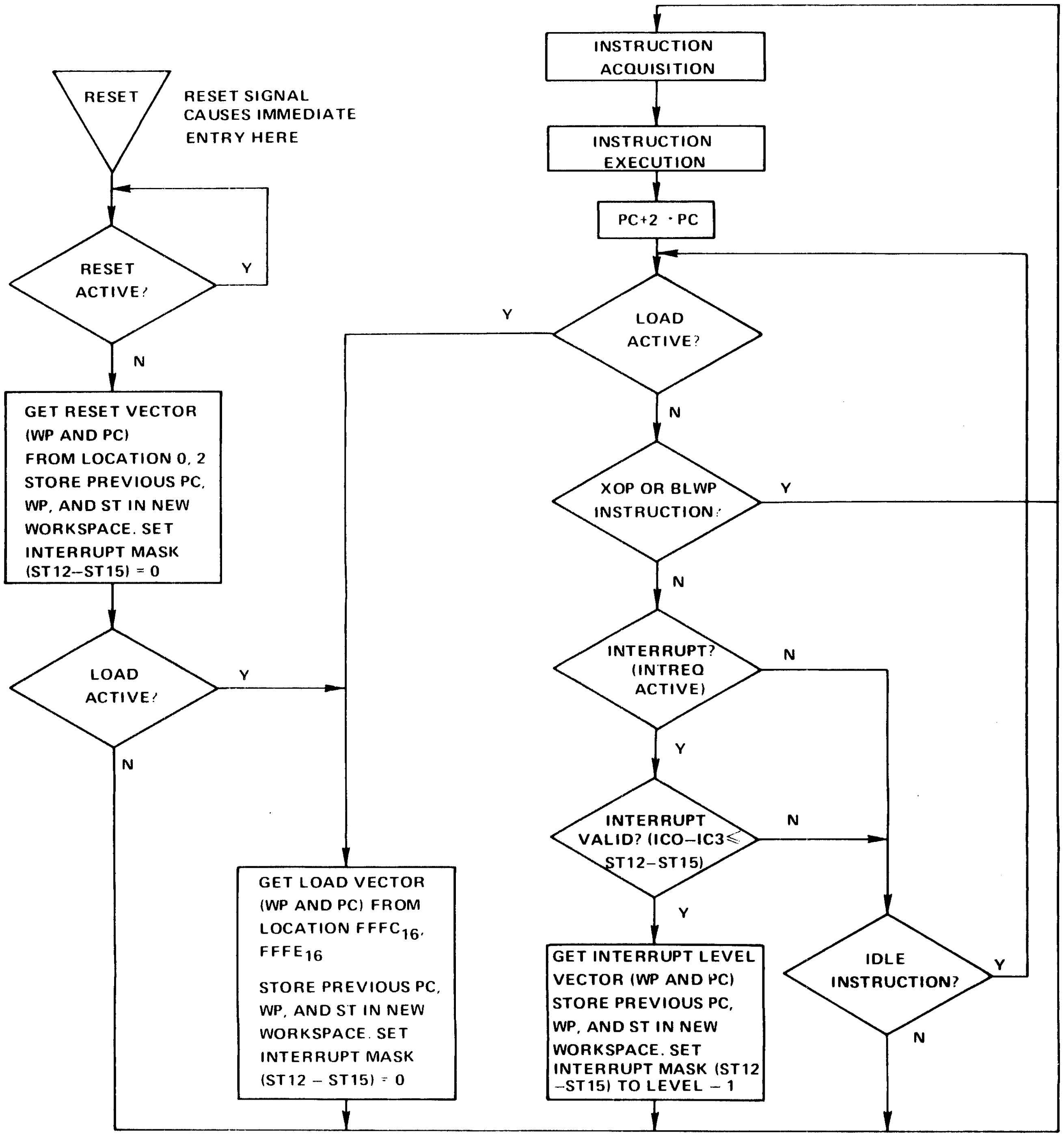


FIGURE 5-3. TMS 9900 SIGNALS



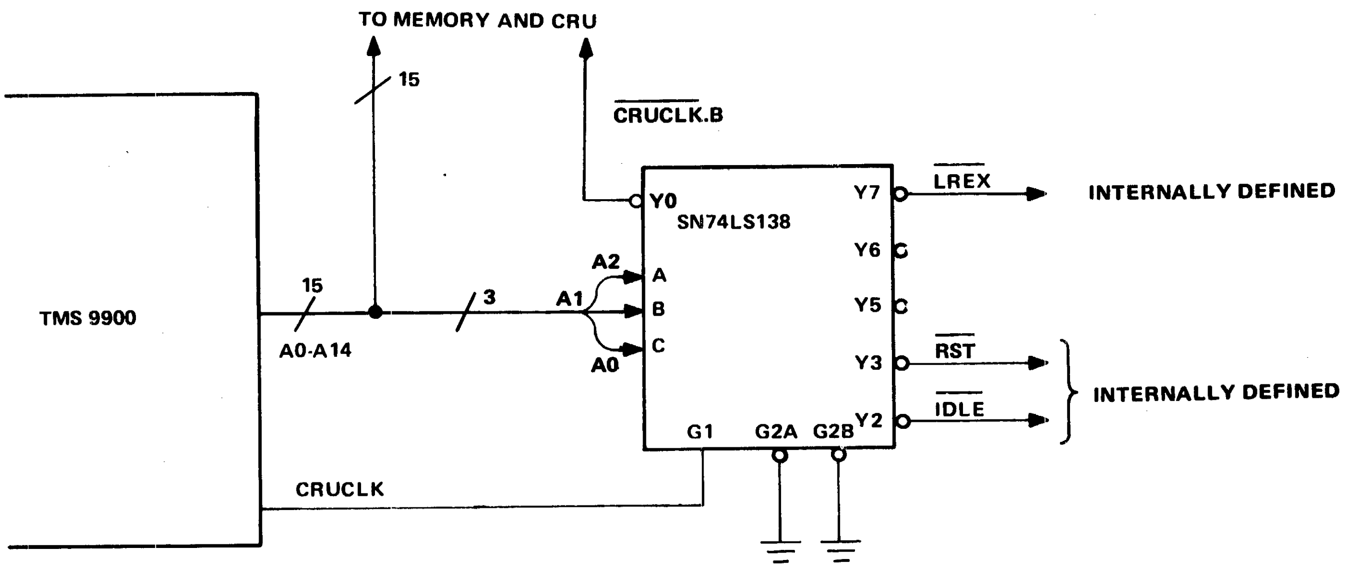
A0001441

FIGURE 5-4. TMS 9900 DATA AND ADDRESS FLOW


RESET
RESET SIGNAL CAUSES IMMEDIATE ENTRY HERE
RESET ACTIVE?
Y
N
GET RESET VECTOR (WP AND PC) FROM LOCATION 0, 2 STORE PREVIOUS PC, WP, AND ST IN NEW WORKSPACE. SET INTERRUPT MASK (ST12–ST15) = 0
LOAD ACTIVE?
Y
N
INSTRUCTION ACQUISITION
INSTRUCTION EXECUTION
PC+2 → PC
LOAD ACTIVE?
Y
N
XOP OR BLWP INSTRUCTION?
Y
N
INTERRUPT? (INTREQ ACTIVE)
N
Y
INTERRUPT VALID? (IC0–IC3 ≤ ST12–ST15)
N
Y
GET LOAD VECTOR (WP AND PC) FROM LOCATION FFFC16, FFFE16 STORE PREVIOUS PC, WP, AND ST IN NEW WORKSPACE. SET INTERRUPT MASK (ST12 – ST15) = 0
GET INTERRUPT LEVEL VECTOR (WP AND PC) STORE PREVIOUS PC, WP, AND ST IN NEW WORKSPACE. SET INTERRUPT MASK (ST12 –ST15) TO LEVEL – 1
IDLE INSTRUCTION?
Y
N

A0001443

FIGURE 5-5. TMS 9900 CPU FLOWCHART



A0001444

FIGURE 5-6. EXTERNAL INSTRUCTION DECODE LOGIC ON TMS 9900

5.5 MEMORY I/O DECODER (Figure 5-8)

This area is responsible for decoding the most significant (A_0 and A_8) bits of the address lines into chip select lines in order to address either RAM or ROM or an I/O device (TMS 9901 or TMS 9902). A 74S287 decodes address lines A_0 (MSB of a 15-bit address) through A_5 to determine memory address of a 16-bit word in RAM or ROM. A 74S288 decodes A_6 to A_8 to determine addressing of the TMS 9901, TMS 9902, outputs at the wire-wrap area, or external CRU. Signal $\overline{\text{MEMEN}}$ (memory enable) determines whether memory or an I/O device is being addressed.

Jumper J2 reflects whether the EPROM's in positions U42, U43, U44, and U45 are TMS 2708's or TMS 2716's, and changes the address map accordingly. See section 7.6.

$\overline{\text{SEL1}}$, $\overline{\text{SEL2}}$, $\overline{\text{SEL3}}$, $\overline{\text{SEL4}}$, and $\overline{\text{SEL5}}$ are five signals routed to the wire wrap area on the TM 990/100M. These signals are intended to be utilized as I/O device select lines. All lines are decoded for 32 consecutive CRU bits.

Table 5-1 lists the CRU bit address from which the lines are active.

5.6 RANDOM ACCESS MEMORY (Figure 5-9)

Four TMS 4042-2 chips, each consisting of 256 x 4 bits, comprise the random access memory. The standard TM 990/100M is populated with 256 words of RAM (four TMS 4042-2's). An optional four-chip block can be added to increase on-board RAM to 512 16-bit words. Figure 5-9 shows the RAM array.

5.7 READ ONLY MEMORY (Figure 5-10)

Blocks of TMS 2708 EPROM chips, each consisting of 1024 x 8 bits, comprise the eraseable read only memory (EPROM). A block of two TMS 2708 chips, containing 1024 words, comes with the TM 990/100M. An optional second block can be added to increase EPROM to 2048 16-bit words. Figure 5-10 shows the EPROM array. Jumper options at J3 and J4 select whether the EPROM's are TMS 2708's or TMS 2716's. See section 7.6.

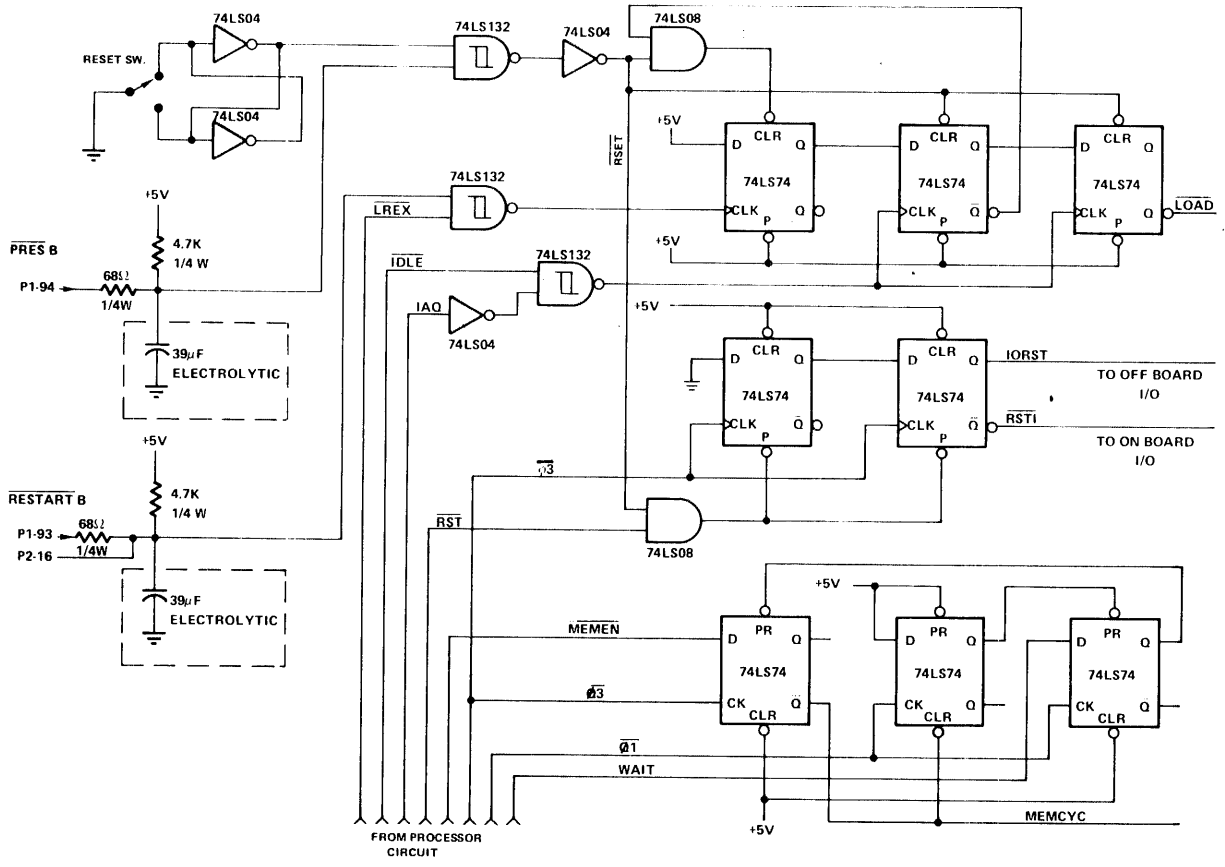


FIGURE 5-7. RESET AND LOAD LOGIC

NOTE

EPROM expansion to 4K is possible by using TMS 2716 EPROM's (2K x 8 bits) and making jumper changes. This is discussed in Section 7, Options.

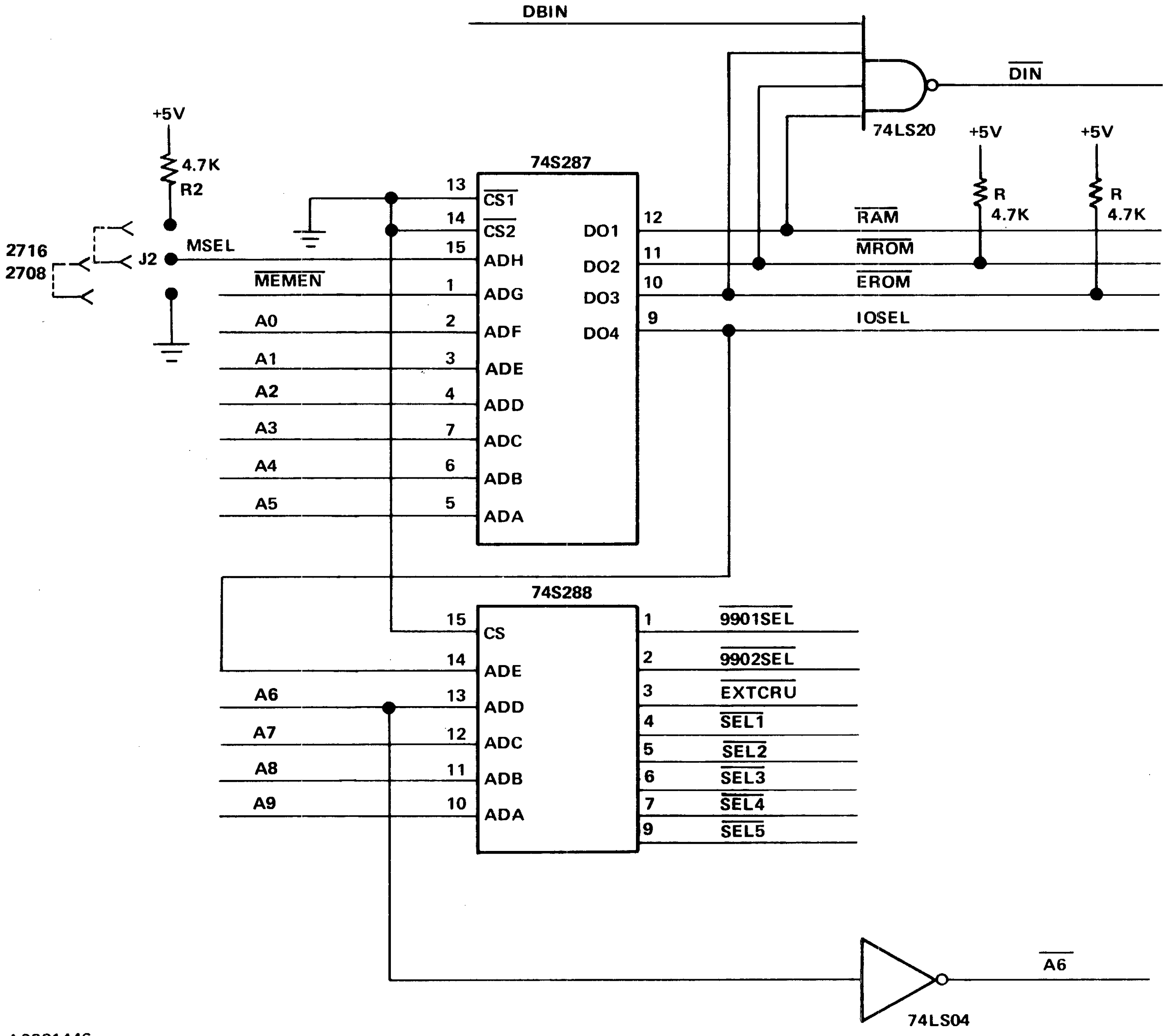
5.8 OFFBOARD EXPANSION BUFFERS (Figures 5-11 and 5-12)

Offboard expansion is possible by tapping signals at the P1 connector. The signals are buffered to drive board-to-board lines (Section 6, Applications, contains examples of memory and I/O expansion off board). Figures 5-11 and 5-12 show logic buffering the signals to connector P1. Table H1 in Appendix H lists connector P1 pins and signals at these pins.

5.9 TMS 9901, PARALLEL I/O, INTERRUPTS (Figure 5-13)

The TMS 9901 controls:

- 16-bit (maximum) parallel input and output
- Interrupt signals to the TMS 9900 CPU



A0001446

FIGURE 5-8. MEMORY I/O DECODER

TABLE 5-1. I/O DEVICE SELECT LINES

BASE ADDRESS REGISTER 12	CRU BIT NUMBER	SIGNAL (ACTIVE LOW)
0000_{16} ✓	0000_{16}	$\overline{SEL1}$
0040_{16} ✓	0020_{16}	$\overline{SEL2}$
$00C0_{16}$ ✓	0060_{16}	$\overline{SEL3}$
0140_{16} ✓	$00A0_{16}$	$\overline{SEL4}$
0180_{16} ✓	$00C0_{16}$	$\overline{SEL5}$

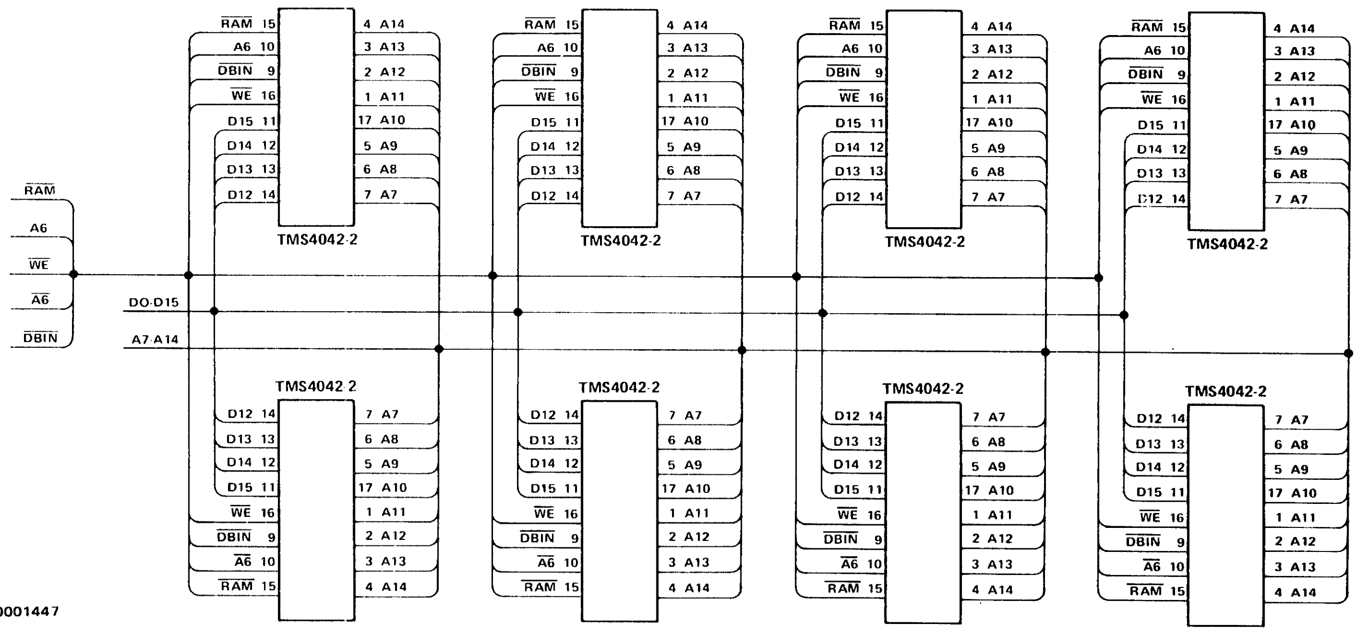
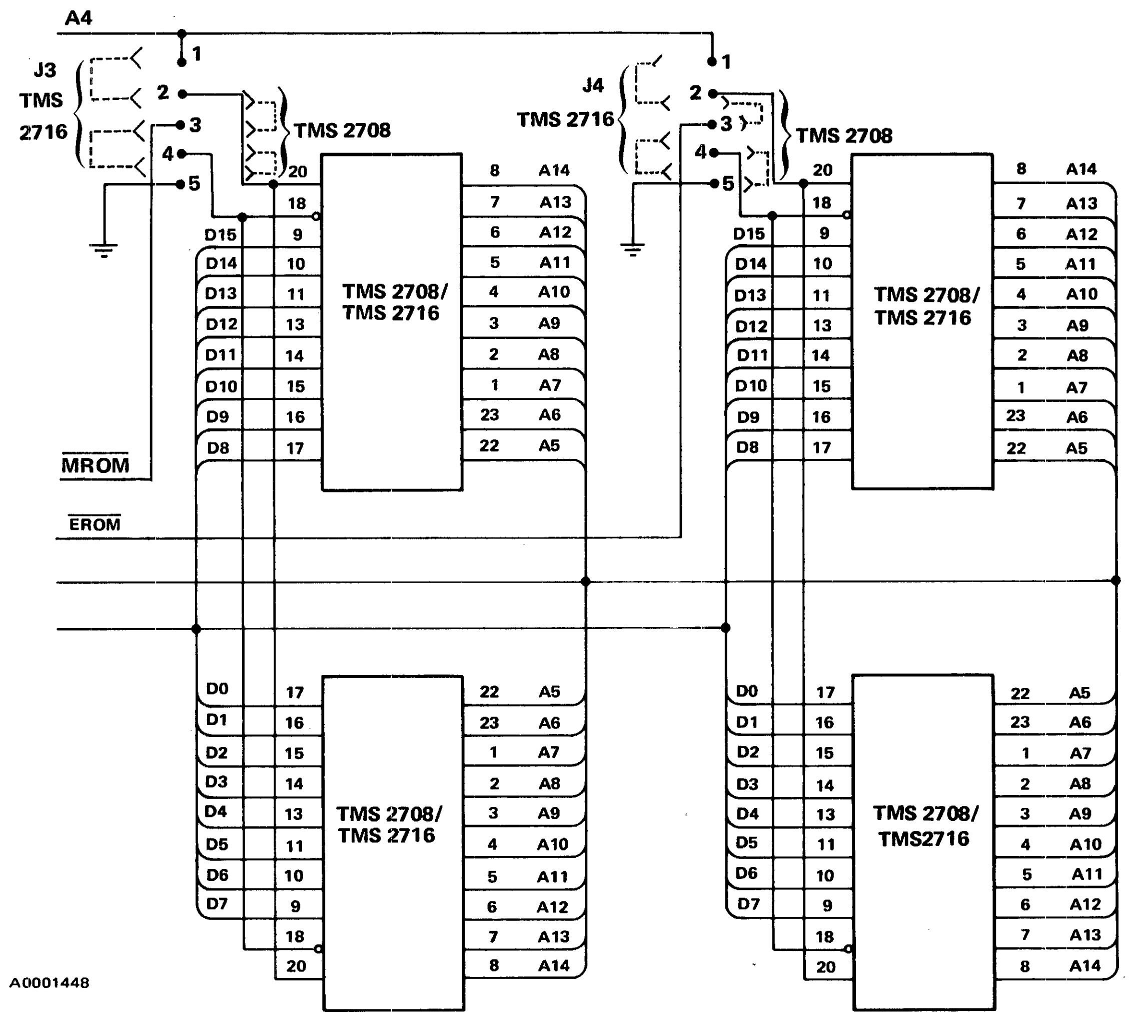


FIGURE 5-9. RANDOM ACCESS MEMORY



VOLTAGE	PIN
V_{CC}	24
V_{BB}	21
V_{DD}	19
V_{SS}	12

FIGURE 5-10. READ ONLY MEMORY

TMS 9901 transmission to and from memory is handled by CRU instructions. Data to be transmitted in parallel is received serially by the TMS 9901. Parallel received data is input to memory serially.

Interrupts received by the TMS 9901 are coded and sent via signals IC0 to IC3 to the CPU when signal INTREQ (interrupt request) goes low.

Figure 5-13 shows signal flow to and from the TMS 9901. Further information can be obtained from the TMS 9901 data manual.

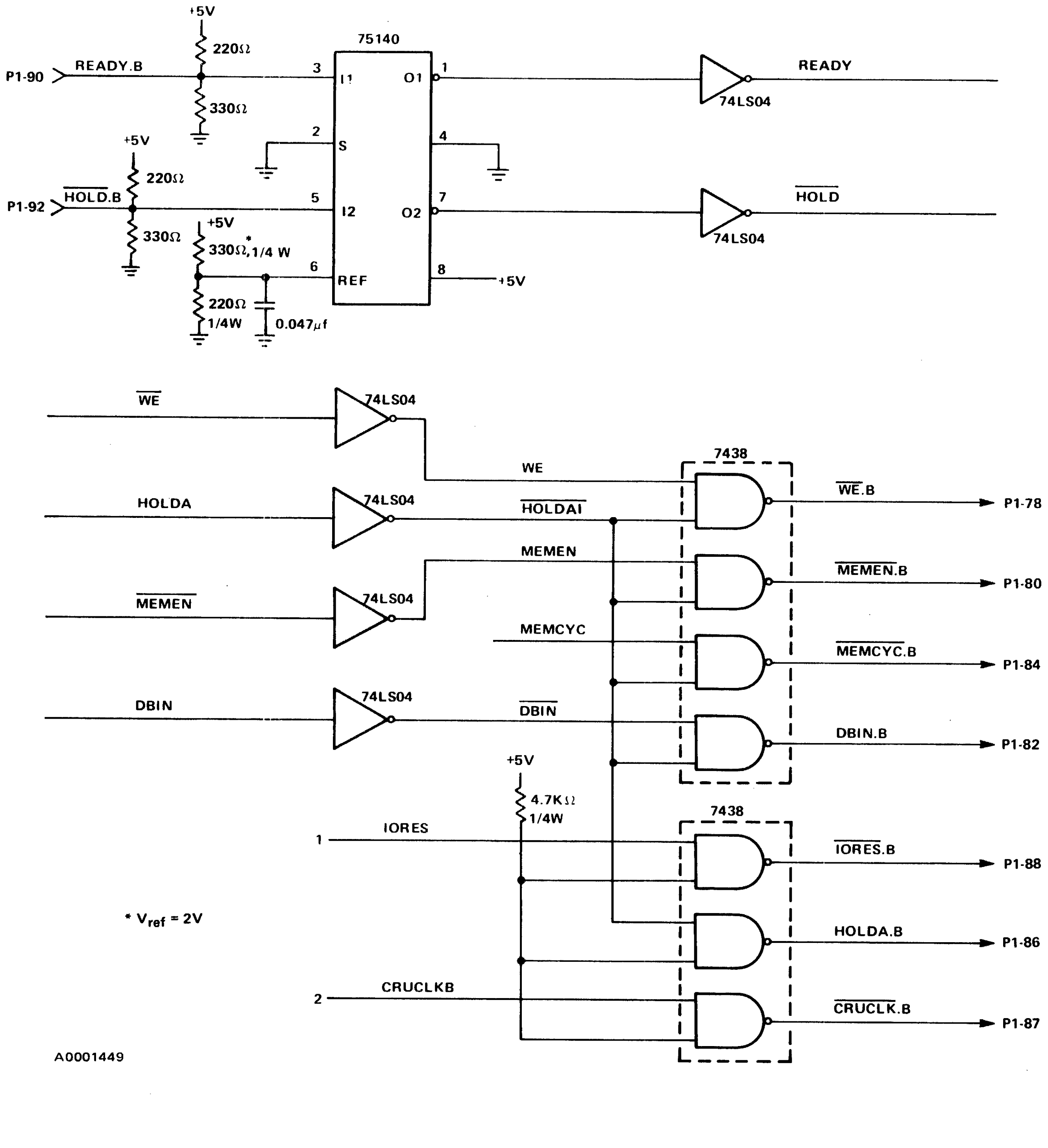


FIGURE 5-11. BUFFERING OF CONTROL SIGNALS TO CONNECTOR P1

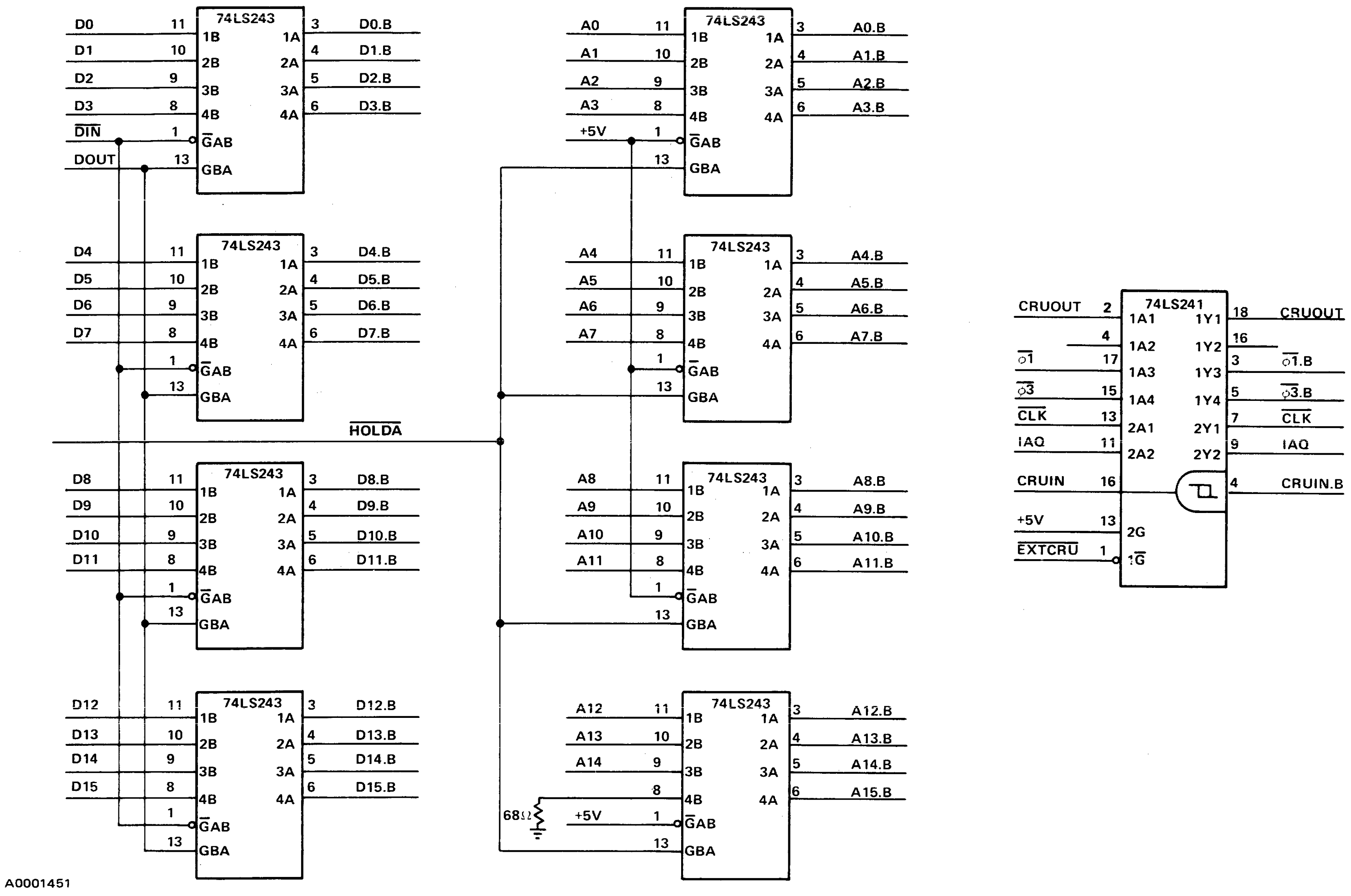

74LS243
74LS241
D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12 D13 D14 D15
D0.B D1.B D2.B D3.B D4.B D5.B D6.B D7.B D8.B D9.B D10.B D11.B D12.B D13.B D14.B D15.B
DIN
DOUT
HOLDA
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14
A0.B A1.B A2.B A3.B A4.B A5.B A6.B A7.B A8.B A9.B A10.B A11.B A12.B A13.B A14.B A15.B
+5V
68Ω
1B 2B 3B 4B 1A 2A 3A 4A GAB GBA
CRUOUT
ϕ1 ϕ3 CLK IAQ CRUIN EXTCRU
ϕ1.B ϕ3.B CLK IAQ CRUIN.B CRUOUT
1A1 1A2 1A3 1A4 2A1 2A2 1Y1 1Y2 1Y3 1Y4 2Y1 2Y2 2G 1G
A0001451

FIGURE 5-12. BUFFERING OF ADDRESS AND DATA SIGNALS TO CONNECTOR P1

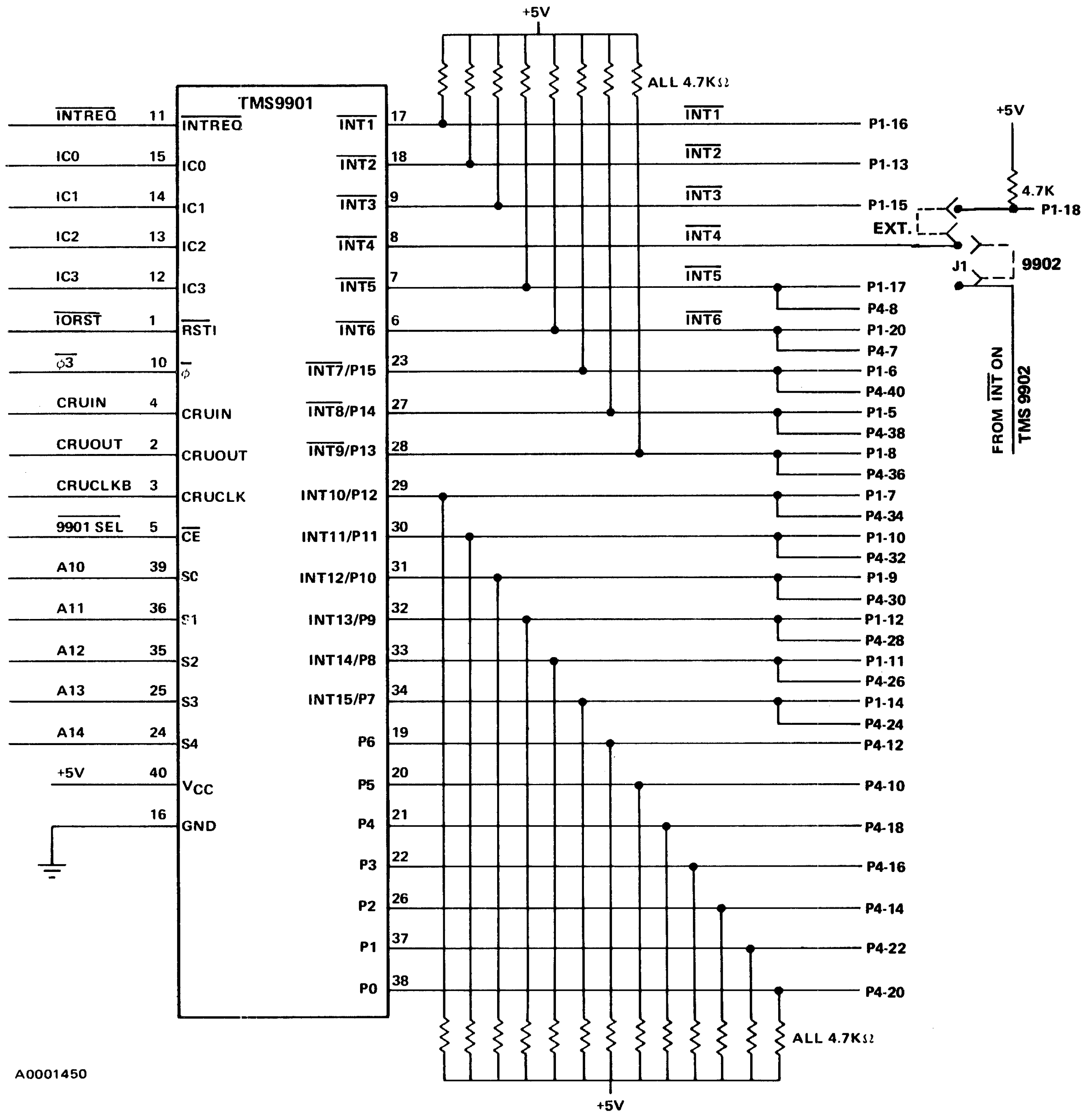


FIGURE 5-13. TMS 9901 EXTERNAL LOGIC

5.10 TMS 9902, SERIAL I/O INTERFACE (Figure 5-14)

The TMS 9902 controls serial I/O for the TM 990/100M. Through CRU instructions the user can set:

- Control criteria such as parity and character length
- Interval timer rate
- Receive data rate
- Transmit data rate

Data is transmitted and received through the CRUOUT and CRUIN lines. The TMS 9902 can interface with a terminal through the EIA connector, P2. An interfacing of level shifters is used to allow hookup to a Texas Instruments 743 KSR, teletypewriter, or other RS-232-C terminal. See Figure 5-14.

When operating under the monitor (supplied with assembly 999211-0001 only), the TMS 9902 is used to control communication by monitoring signals at the CRU. Signals used for communications purposes also cause an interrupt level 4 at the TMS 9901. Because of this, jumper J1 must be removed when using the *TIBUG* monitor to prevent the internal interrupt from incumbering monitoring operation. This interrupt is described in detail in paragraph 6.6. Further information is available from the TMS 9902 data manual.

5.11 SERIAL I/O INTERFACE (Figure 5-15)

This area provides an interface between the TMS 9902 and a 743 KSR, teletypewriter, or RS-232-C terminal. The board comes jumpered for 743 KSR operation (jumper J11 disconnected). Section 7 (Options) contains a description of accommodating optional terminals. J11 is installed if the terminal used is a teletypewriter. Jumper J7 must be in the EIA position to use an EIA terminal or a teletypewriter with the TM 990/100M. Jumper locations are shown in Figure 7-2.

5.12 WIRE-WRAP AREA (Figure 5-16)

A wire-wrap area has been provided for adding additional devices such as TMS 9901's or TMS 9902's. On the periphery of the wire-wrap area are pads containing voltages and signals as shown in Figure 5-16.

Spare pins from the 40-pin board edge connectors P3 and P4 are routed to an array of plated through holes near the bottom of each connector. This facilitates interconnection of these spare pins with circuitry added in the wire-wrap area.

The wire-wrap area consists of an array of .046 inch diameter holes spaced on 0.1 inch centers. It is suggested that networks placed in this area be mounted in sockets with wire-wrap tails. Interconnections are thus facilitated in wire-wrap. Two 16-pin DIP socket locations are dedicated for connection to power and miscellaneous CRU control signal. See Figure 5-16.

5.13 MULTIDROP I/O INTERFACE (Figure 5-17)

The Multidrop interface may be used for board-to-board communication over long distances. Generally, all that is required is a twisted pair line run between the boards. More than two boards may be linked together, each one is just "dropped" into place, hence the term "multidrop". If more than two boards are used, the boards not at the extreme ends of the twisted pair line (i.e., those "dropped in the middle") are considered non-terminating boards, and the termination resistor jumper plugs should be removed to prevent standing wave patterns which might occur, mostly at the higher baud rates. The two boards at the extremes of the line, regardless of whether additional boards exist in between, should have these resistor jumper plugs installed (J9–J12). Jumpers to be installed for the multidrop operation are listed below:

	INSTALL	REMOVE
Half Duplex, non-terminating	J5, J8, J7 (MD)	J6, J9–J12
Full Duplex, non-terminating	J7 (MD)	J5, J6, J8–J12
Half Duplex, terminating board	J7 (MD), J5, J6, J8–J10, J12	J11
Full Duplex, terminating board	J7 (MD), J6, J9, J10, J12	J11, J5, J8

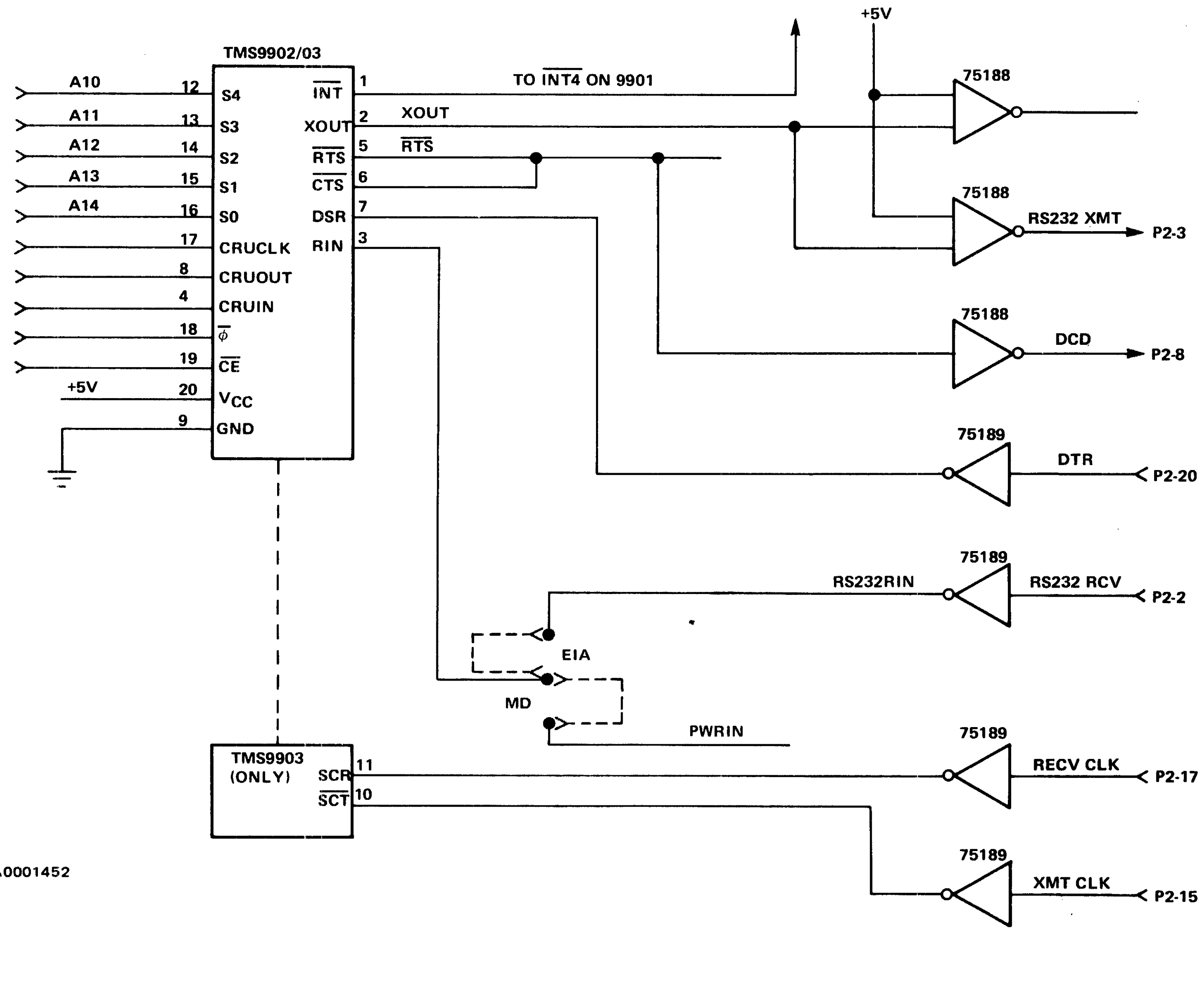
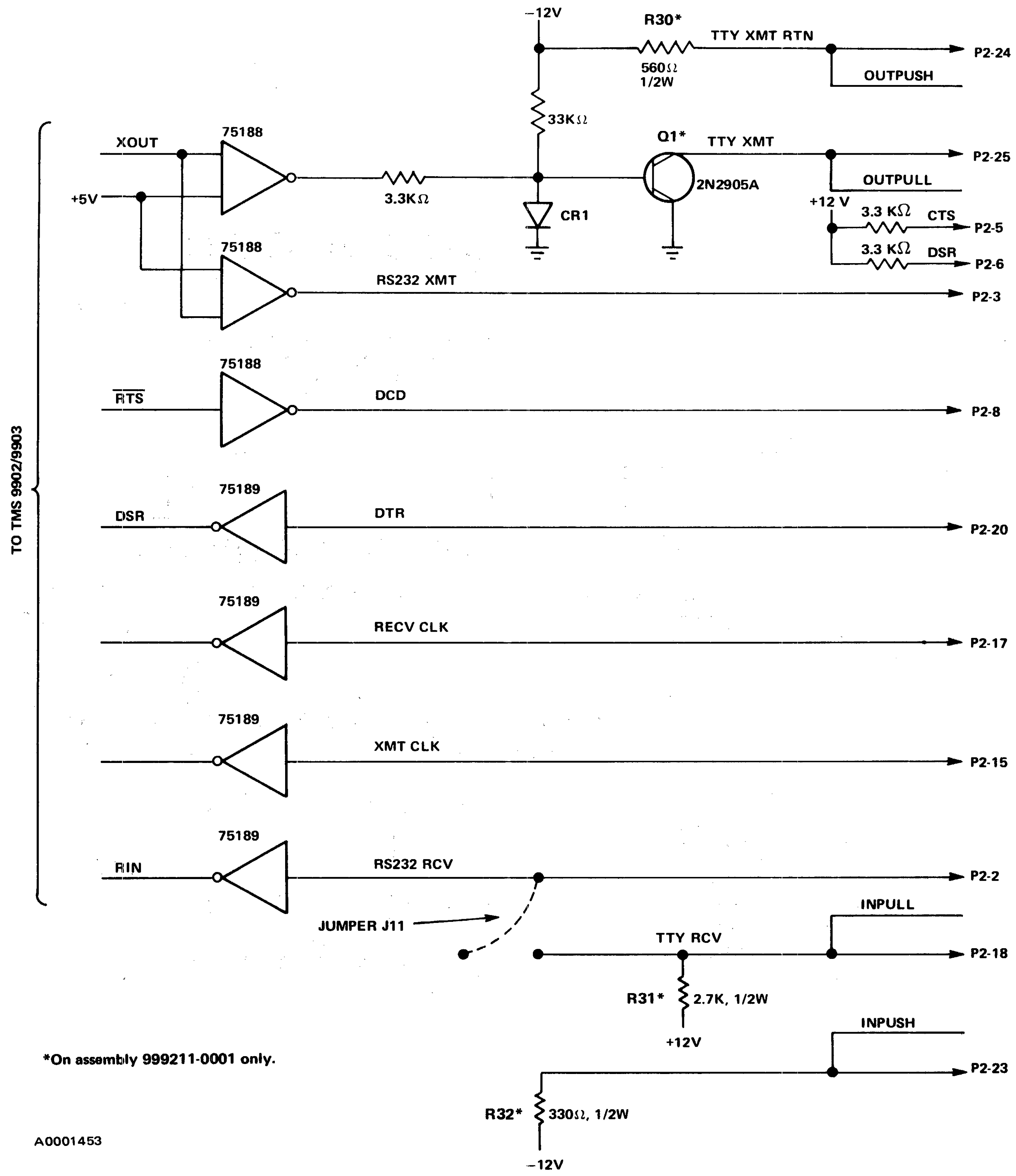


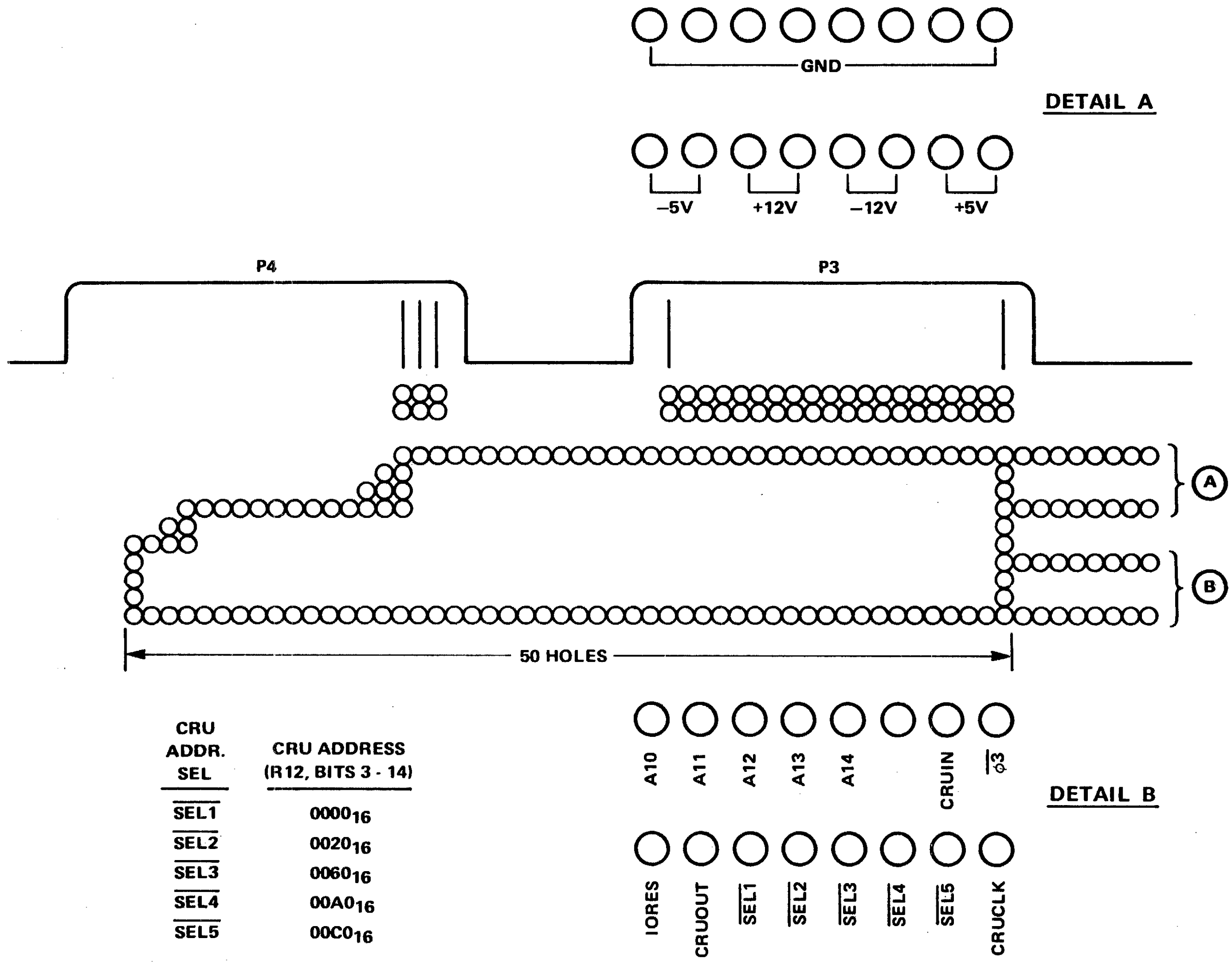
FIGURE 5-14. TMS 9902 EXTERNAL LOGIC



*On assembly 999211-0001 only.

A0001453

FIGURE 5-15. SERIAL I/O INTERFACE



CRU ADDR. SEL	CRU ADDRESS (R12, BITS 3 - 14)
$\overline{SEL1}$	0000_{16}
$\overline{SEL2}$	0020_{16}
$\overline{SEL3}$	0060_{16}
$\overline{SEL4}$	$00A0_{16}$
$\overline{SEL5}$	$00C0_{16}$

A0001454

FIGURE 5-16. SIGNALS AT WIRE-WRAP AREA

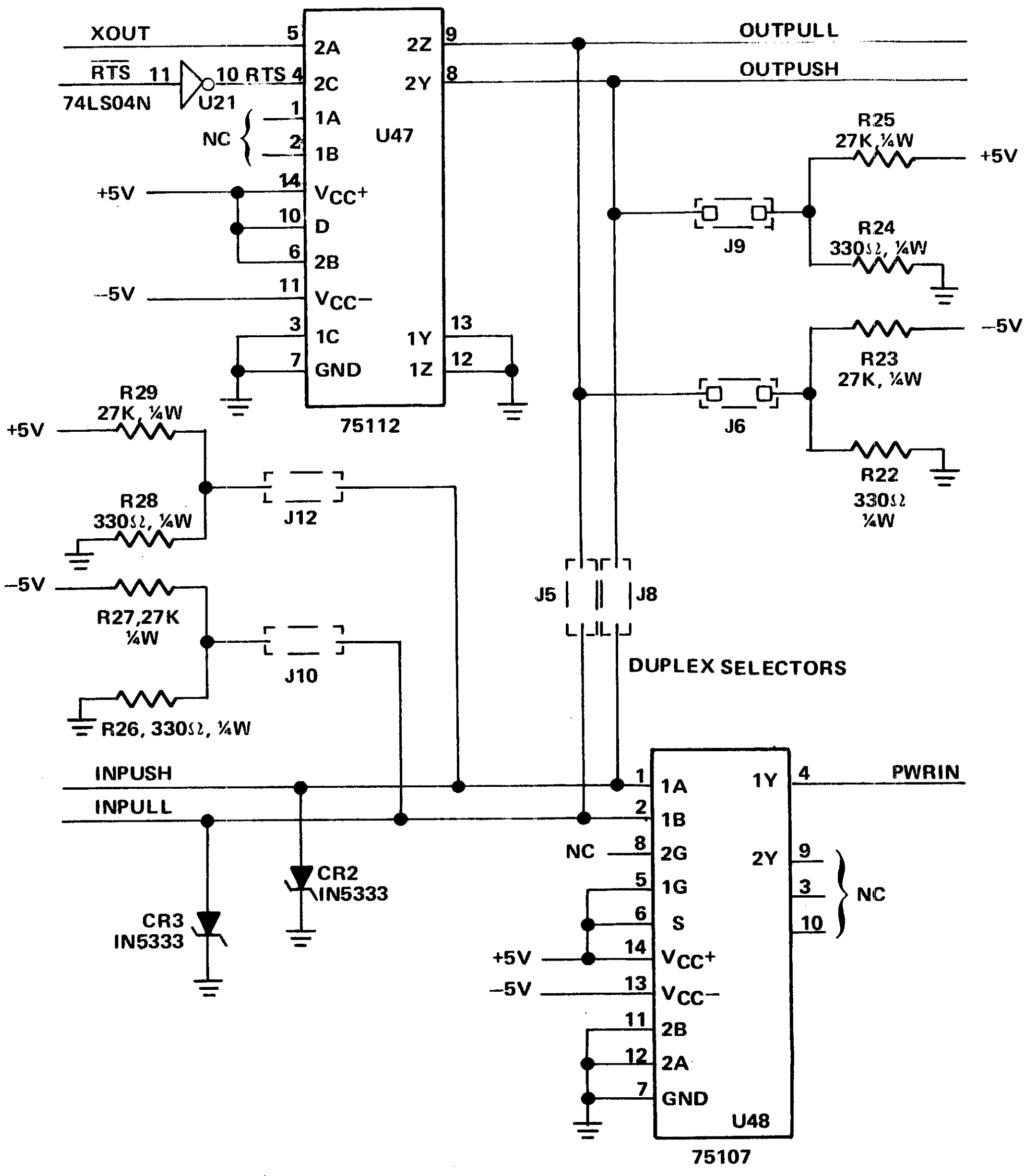


FIGURE 5-17. MULTI-DROP INTERFACE

SECTION 6

APPLICATIONS

6.1 GENERAL

This section covers various methods of communicating to applications external to the TM 990/100M. Figure 6-1 shows board locations applicable to this section.

A wirewrap area has been provided for wiring devices on board. This area, shown in detail in Figure 6-2, contains signal input and output pins located on its periphery. Table 6-1 lists the signatures of the pins. Note that a spare 40-pin connector (P3) is available adjacent to the wirewrap area.

6.2 WIRE-WRAP ADDITIONAL ON-CARD TMS 9901

An additional TMS 9901 may be added for an external application. Figure 6-3 shows wire-wrap wiring to add a TMS 9901 I/O controller and associated resistor packs. Sockets with wire-wrap tails are inserted into the board to accommodate the devices and wiring.

Signals and power available at the wire-wrap area are shown in Figure 6-2. The use of $\overline{\text{SEL1}}$ to the 74LS00 designates a CRU address of 0000_{16} (bits 3 to 14 of R12).

6.3 PARALLEL I/O PORT CIRCUITRY

Figure 6-4 shows a parallel I/O port that can be implemented in the wire-wrap area. Wire-wrap area signals are available as shown in Figure 6-2. This port consists of eight input and eight output lines. These 16 lines are interfaced to connector P3, pins 1 to 16.

6.4 OFF-CARD ADDITIONAL RANDOM ACCESS MEMORY

Figure 6-5 shows suggested wiring for adding up to 1K words of RAM off-board in 256-word increments. Table 6-2 is a list of materials for this addition.

6.5 ADD OFF-CARD TMS 9901

Figure 6-6 shows circuitry, connected through connector P1, for connecting an additional TMS 9901 off the card. The CRU address for the TMS 9901 in this configuration is $0FF0_{16}$.

6.6 ON-BOARD COMMUNICATIONS INTERRUPT

The TMS 9902 will issue a level 4 interrupt when programmed as in paragraph 4.9. Positioning jumper J1 (shown in Figure 6-1) to the "9902" position connectors the interrupt output of the TMS 9902 to interrupt level 4. This allows interrupt operation of the TMS 9902.

NOTE

As shown in Figure 6-7, the TMS 9902 timer as well as three other conditions cause an interrupt to be generated (INT) which can be routed to interrupt 4 of the TMS 9901. Because these signals are monitored through the CRU by the *TIBUG* monitor to facilitate I/O and other functions, the jumper at J1 must be in the "P1-18" position when operating under the monitor.

TIBUG operates the serial interface under polled control. If the TMS 9902 was to initiate interrupts to the TMS 9900 microprocessor, incorrect *TIBUG* operation would result.

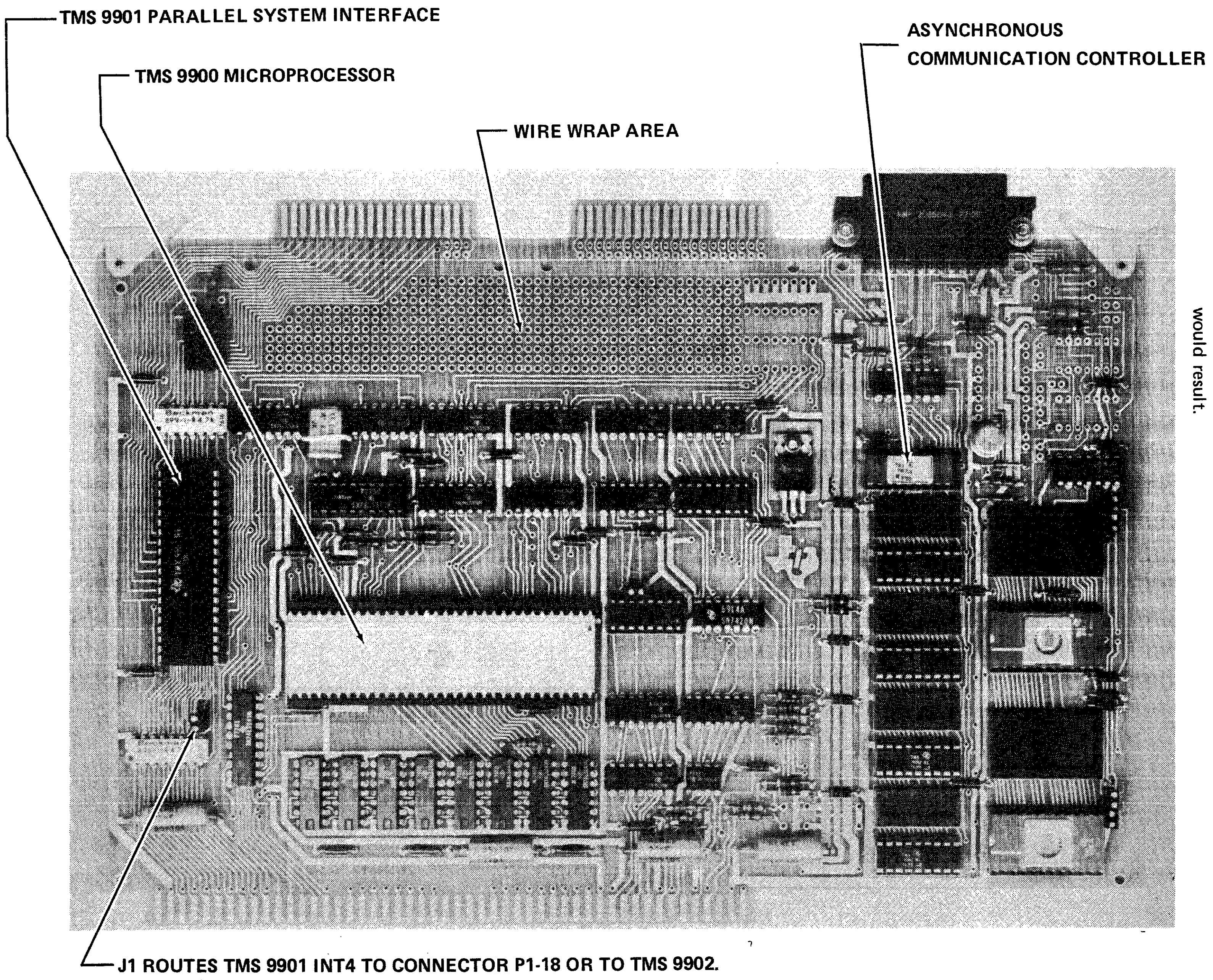
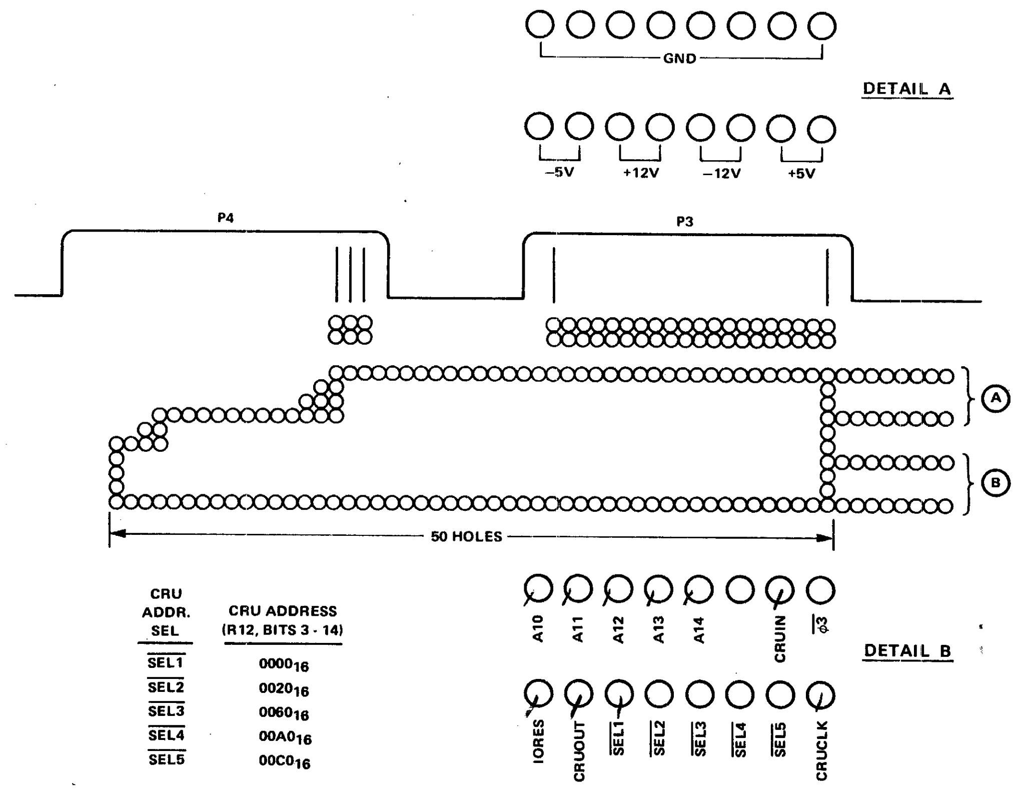

TMS 9901 PARALLEL SYSTEM INTERFACE
TMS 9900 MICROPROCESSOR
WIRE WRAP AREA
ASYNCHRONOUS COMMUNICATION CONTROLLER
J1 ROUTES TMS 9901 INT4 TO CONNECTOR P1-18 OR TO TMS 9902.

FIGURE 6-1. DEVICES USED IN VARIOUS APPLICATIONS

TABLE 6-1. I/O PINS AT WIREWRAP AREA

SIGNAL	DEFINITION
A10 to A14	Five LSB's of address bus
CRUCLKB	CRU clock input
CRUIN	Serial data to CRU
CRUOUT	Serial data from CRU
$\overline{\text{IORST}}$	I/O Reset
$\overline{\text{SEL1}}$	CRU address* is 0000_{16}
$\overline{\text{SEL2}}$	CRU address* is 0020_{16}
$\overline{\text{SEL3}}$	CRU address* is 0060_{16}
$\overline{\text{SEL4}}$	CRU address* is $00A0_{16}$
$\overline{\text{SEL5}}$	CRU address* is $00C0_{16}$
$\overline{\phi3}$	Clock 03
+5V	+5 volt supply
−12V	−12 volt supply
+12V	+12 volt supply
−5V	−5 volt supply

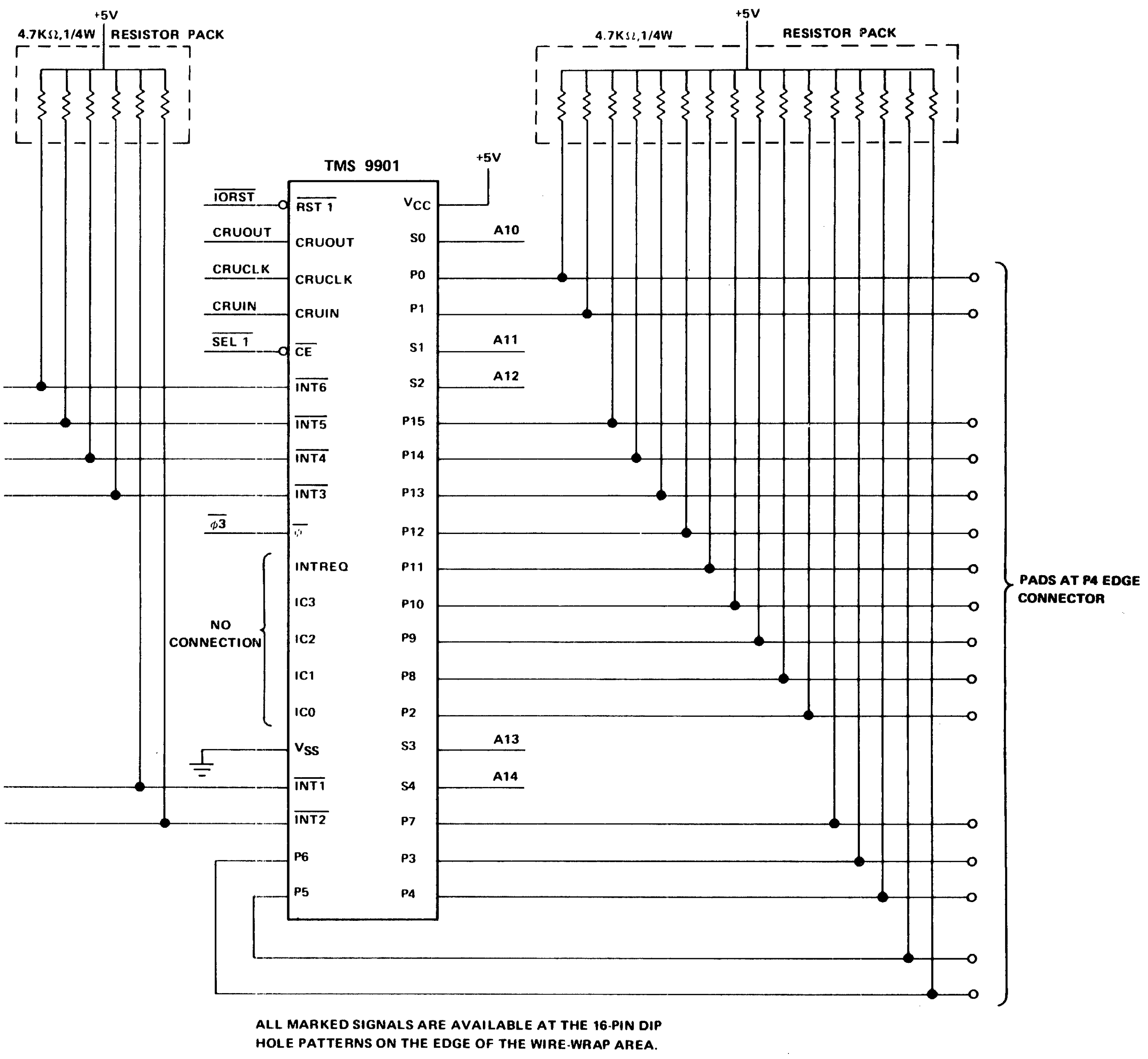
*CRU address is in bits 3 to 14 of R12.



CRU ADDR. SEL	CRU ADDRESS (R12, BITS 3 - 14)
$\overline{\text{SEL1}}$	0000_{16}
$\overline{\text{SEL2}}$	0020_{16}
$\overline{\text{SEL3}}$	0060_{16}
$\overline{\text{SEL4}}$	$00A0_{16}$
$\overline{\text{SEL5}}$	$00C0_{16}$

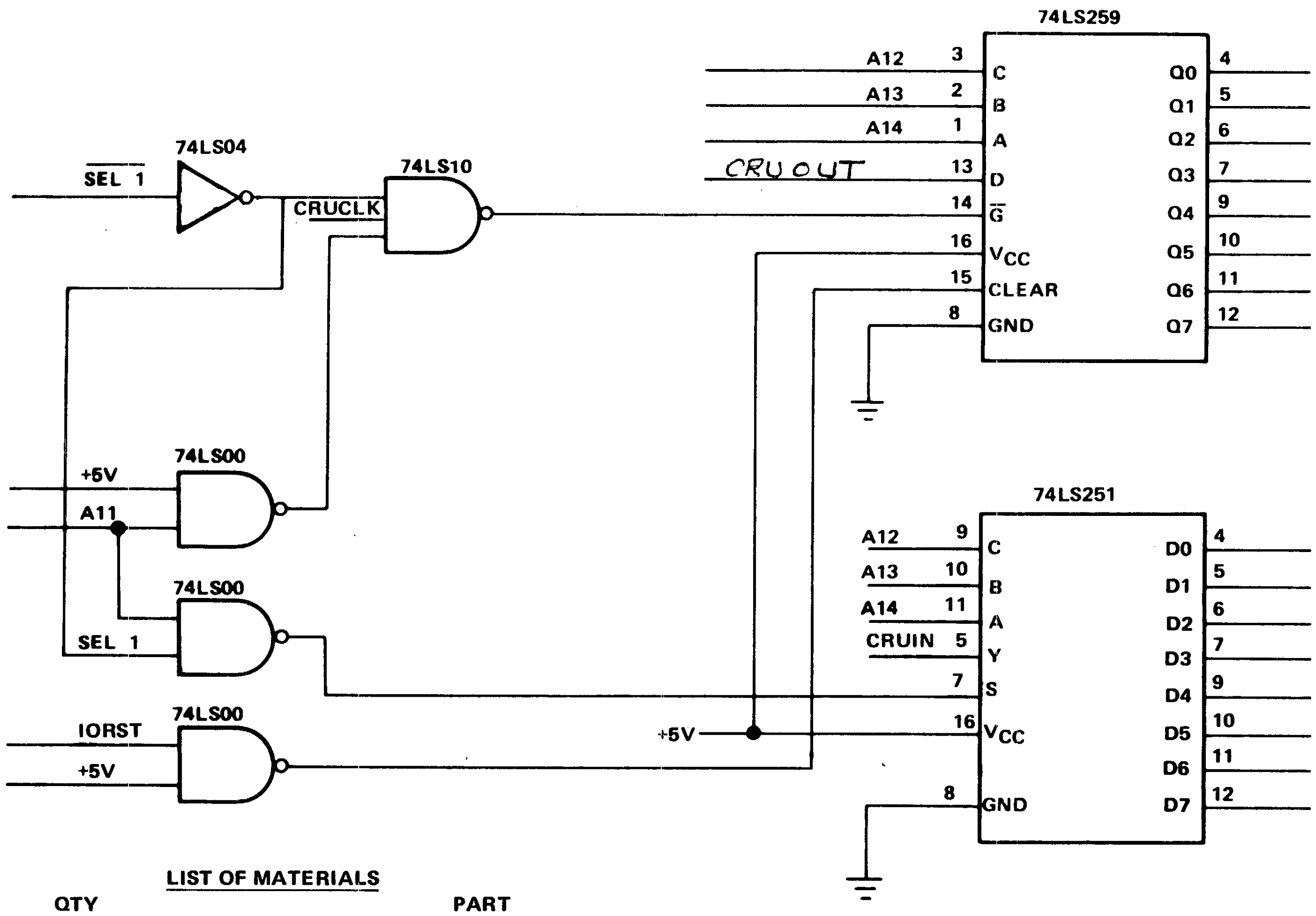
A0001454

FIGURE 6-2. SIGNALS AT WIRE-WRAP AREA



A0001455

FIGURE 6-3. ON-BOARD TMS 9901 WIRING

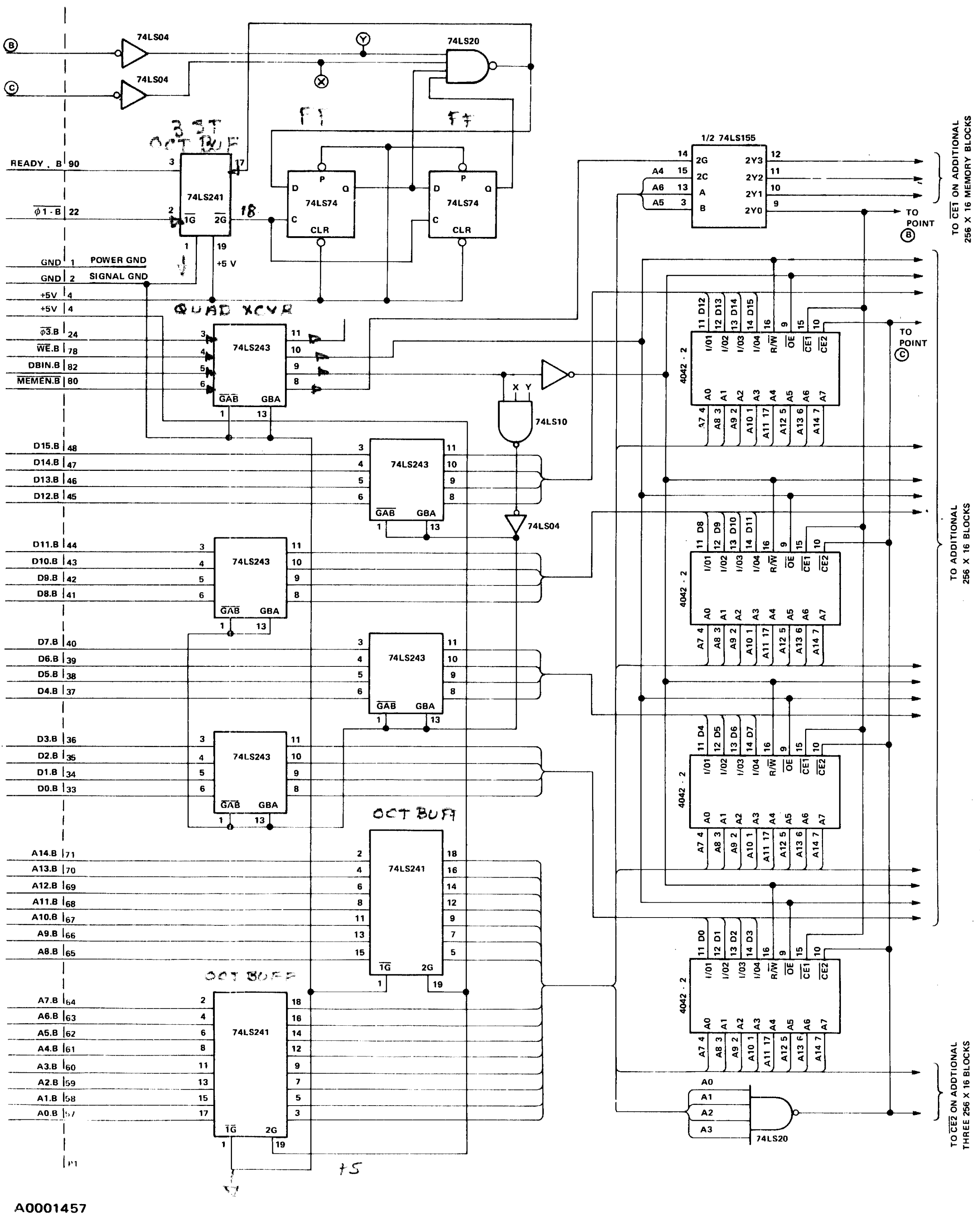


LIST OF MATERIALS

QTY	PART
2	16 - PIN DIP SOCKETS AND WIRE - WRAP PINS
2	14 - PIN DIP SOCKET AND WIRE - WRAP PINS
1	74LS00
1	74LS259
1	74LS04
1	74LS251
1	74LS10

A0001456

FIGURE 6-4. PARALLEL I/O PORT



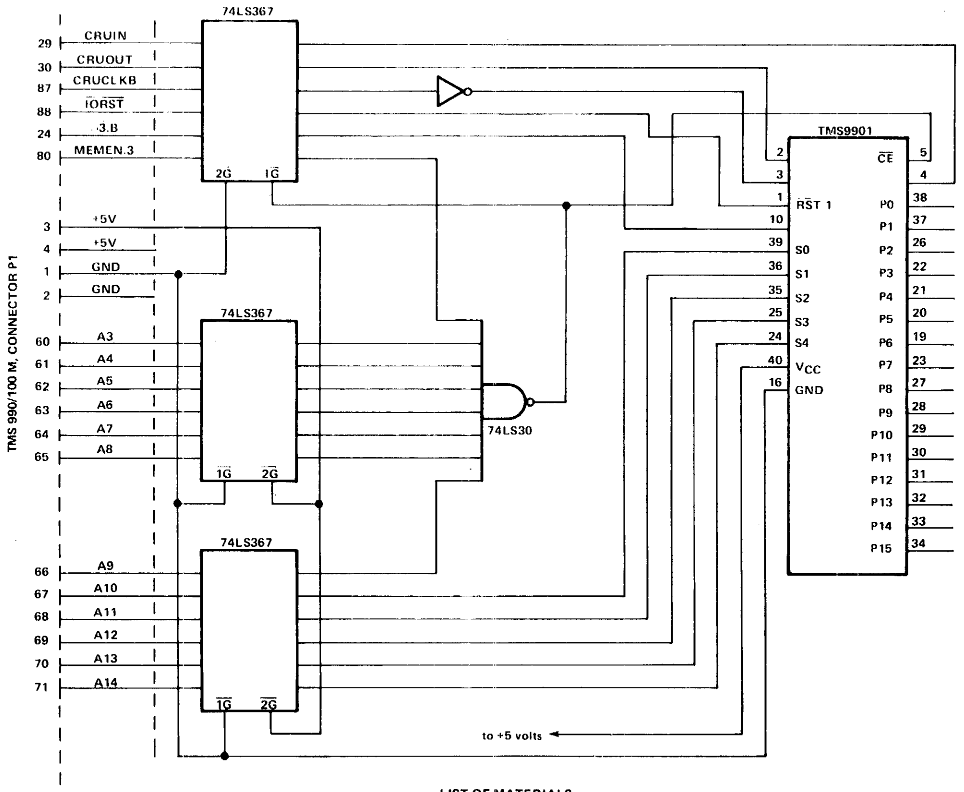
A0001457

FIGURE 6-5. OFF-BOARD EXPANSION OF RAM

TABLE 6-2. LIST OF MATERIALS FOR ADDING RAM

QUANTITY	PART
7	14-pin DIP Socket*
1	16-pin DIP Socket*
4 per 256 words	18-pin DIP Socket*
3	20-pin DIP Socket*
4 per 256 words	TMS 4042-2
1	74LS155
1	74LS20
1	74LS74
1	74LS04
4	74LS243
3	74LS241
1	74LS10

*And wire-wrap pins as required



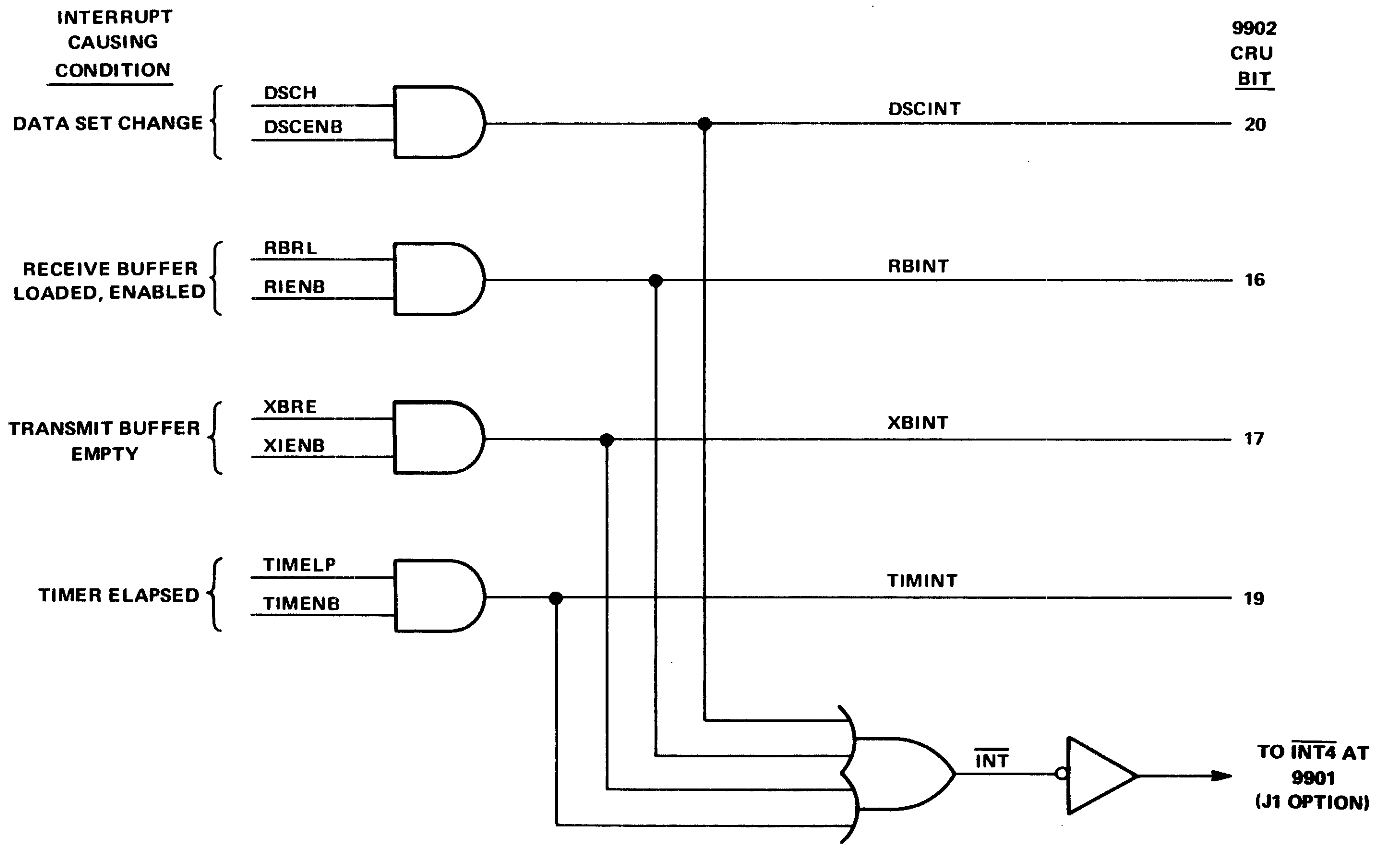
LIST OF MATERIALS

QTY	PART
1	14 - PIN DIP SOCKET*
4	16 - PIN DIP SOCKET*
1	40 - PIN DIP SOCKET
3	74LS367
1	74LS04
1	74LS30
1	TMS 9901

* AND WIRE - WRAP PINS AS REQUIRED

A0001458

FIGURE 6-6. CIRCUITRY TO ADD TMS 9901 OFF-BOARD



A0001459

FIGURE 6-7. FOUR INTERRUPT-CAUSING CONDITIONS AT TMS 9902

SECTION 7

OPTIONS

7.1 GENERAL

This section explains the various options available to the user of the TM 990/100M. These options include:

- Use of TMS 2716 EPROM's (2K x 8 bits each) instead of TMS 2708 EPROM's (1K x 8 bits each) (paragraph 7.2).
- On-card expansion of EPROM and RAM (paragraph 7.2)
- Asynchronous serial interrupt from TMS 9902 (paragraph 7.3).
- RS-232-C or teletypewriter interface (paragraph 7.4). Teletypewriter interface is with assembly 999211-0001 only.
- Microterminal use (paragraph 7.8).
- External switch actuation of a $\overline{\text{RESET}}$ or $\overline{\text{RESTART}}$ signal (paragraph 7.5).
- Memory chip and CRU device selected by bit masks in PROM's (paragraph 7.6).
- Assembler in EPROM (paragraph 7.7).

Figures 7-1 and7-2 show board locations application to this section. Table 7-1 is a summary of jumpers and capacitors used with these options.

7.2 ON-BOARD MEMORY EXPANSION (Figure 7-2)

7.2.1 EPROM EXPANSION

EPROM memory can be expanded on-board in two ways (all expansion memory is provided on assembly 999211-0003):

- Add two TMS 2708 EPROM chips (1K x 8 bits each) to provide an additional 1K words of memory.
- Use two or four TMS 2716 EPROM chips (2K x 8 bits each) to provide 2K or 4K words of memory.

Figure 7-3 shows placement of EPROM chips and corresponding memory addresses (in bytes). The board silkscreen designators identify the necessary jumper placement at J2, J3, and J4.

NOTE

Models 999211-1 and -2 come from the factory with 2 TMS 2708's which are installed in sockets at U42 and U44. Jumper J2 is installed in the "2708" position and Jumpers J3 and J4 in the "08" position. This configuration will allow up to four 2708's to be used in U42 to U45.

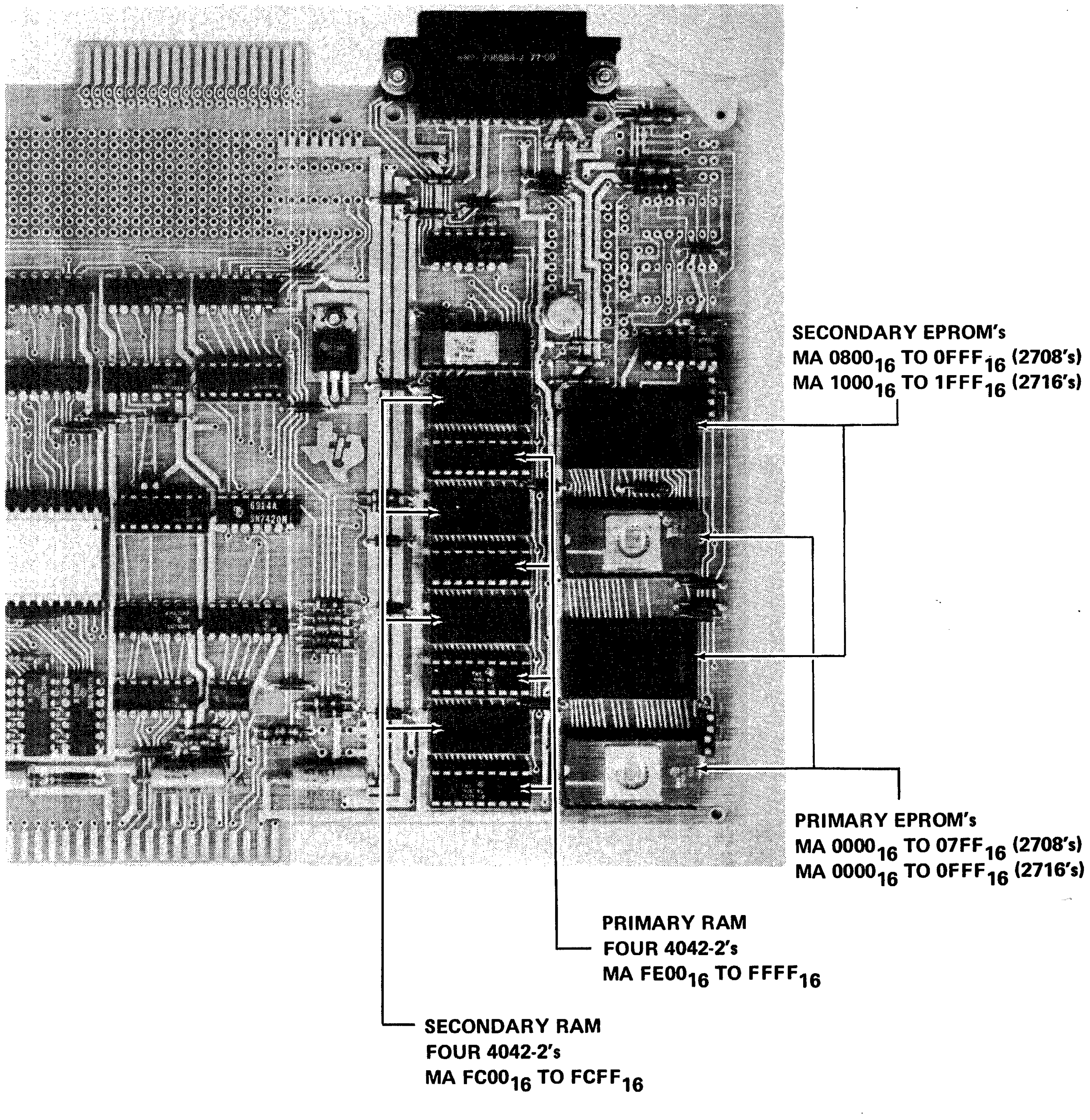

SECONDARY EPROM's
MA 0800_{16} TO $0FFF_{16}$ (2708's)
MA 1000_{16} TO $1FFF_{16}$ (2716's)
PRIMARY EPROM's
MA 0000_{16} TO $07FF_{16}$ (2708's)
MA 0000_{16} TO $0FFF_{16}$ (2716's)
PRIMARY RAM
FOUR 4042-2's
MA $FE00_{16}$ TO $FFFF_{16}$
SECONDARY RAM
FOUR 4042-2's
MA $FC00_{16}$ TO $FCFF_{16}$

FIGURE 7-1. MEMORY PLACEMENT ON BOARD

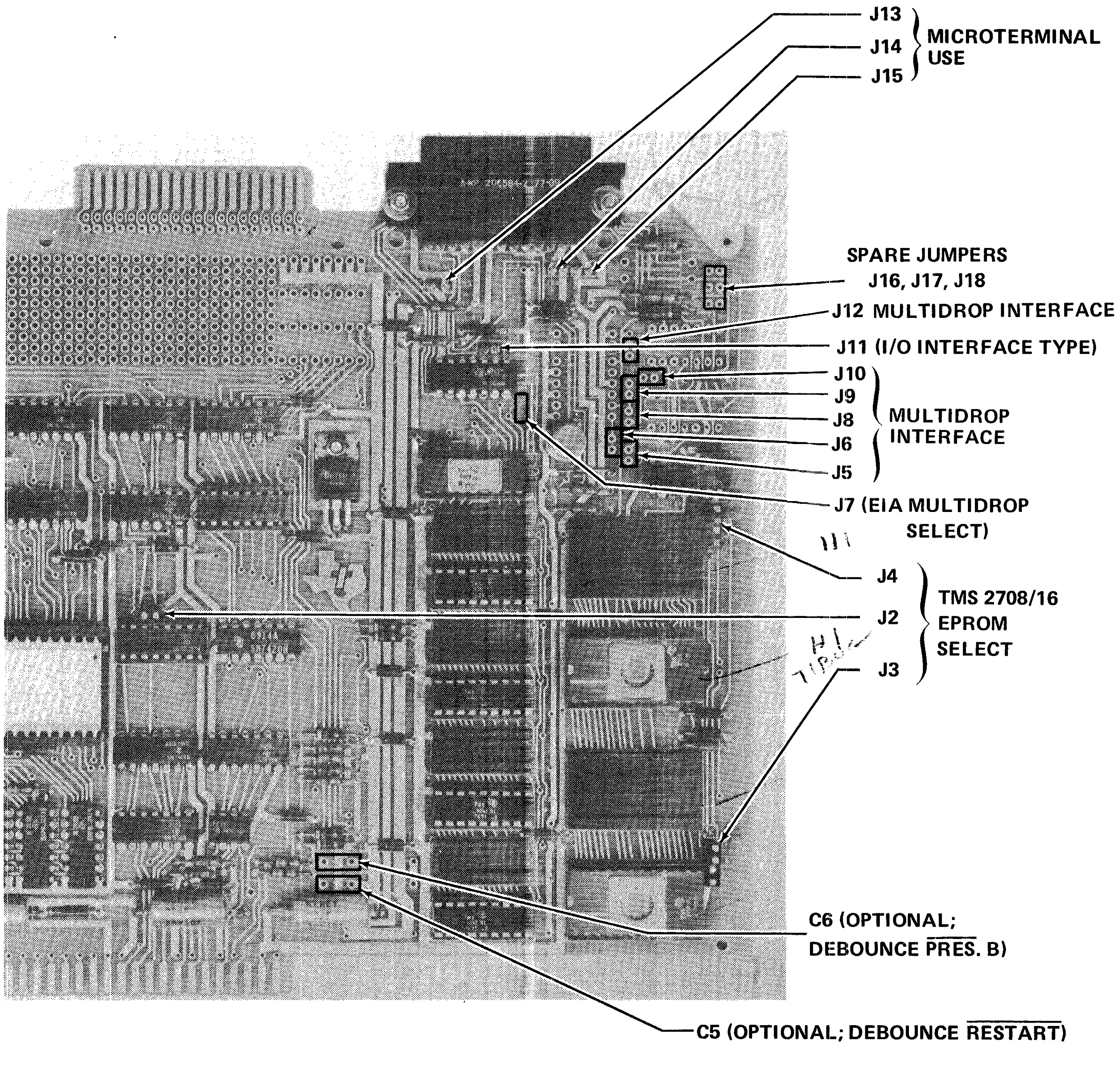

J13
J14
J15
MICROTERMINAL USE
SPARE JUMPERS J16, J17, J18
J12 MULTIDROP INTERFACE
J11 (I/O INTERFACE TYPE)
J10
J9
J8
J6
J5
MULTIDROP INTERFACE
J7 (EIA MULTIDROP SELECT)
J4
J2
J3
TMS 2708/16 EPROM SELECT
C6 (OPTIONAL; DEBOUNCE $\overline{\text{PRES.}}$ B)
C5 (OPTIONAL; DEBOUNCE $\overline{\text{RESTART}}$)

FIGURE 7-2. JUMPERS AND CAPACITORS USED FOR OPTION SELECTION

To utilize TMS 2716 EPROM's J2 must be positioned to "2716" and J3 and J4 to the "16" position.

EPROM types may not be mixed. That is, TMS 2716 may not be populated in U42 and U44 while TMS 2708's are populated in U43 and U45.

7.2.2 RAM EXPANSION

Four additional TMS 4042-2 RAM chips can be added as shown in Figure 7-3. This will provide an additional 512 bytes of RAM. All expansion memory is provided on assembly 999211-0003.

7.3 ASYNCHRONOUS SERIAL COMMUNICATION

An internal interrupt to interrupt trap 4 can be selected through programming considerations described in paragraph 4.9. This interrupt will signal changes in data set status and the current contents of the

TABLE 7-1. JUMPERS AND CAPACITORS USED WITH OPTIONS

OPTION	JUMPERS/CAPACITORS	PARAGRAPH
TMS 9902 INT to Interrupt 4	J1 (as shown on board)	7.10
P1-18 to interrupt 4	J1 (as shown on board)*	7.10
Use TMS 2708 EPROM's	J2, J3, J4 (as shown on board)*	7.2.1
Use TMS 2716 EPROM's	J2, J3, J4 (as shown on board)	7.2.1
20 mA Interface Use	J11 (installed)	7.4
RS-232-C Interface Use	J11 (disconnected)*	7.4
Microterminal Power	J13, J14, J15 (installed)	7.8
External RESTART signal	C5 (installed)	7.5
External PRES.B signal	C6 (installed)	7.5
Multidrop Interface	J5, J6, J8, J9, J10, J12	
EIA/Multidrop Select	J7	

*Configuration when shipped from factory

TMS 9902 transmit buffer or receive buffer. Further information is presented in the *TMS 9902 Asynchronous Communication Controller Data Manual.*

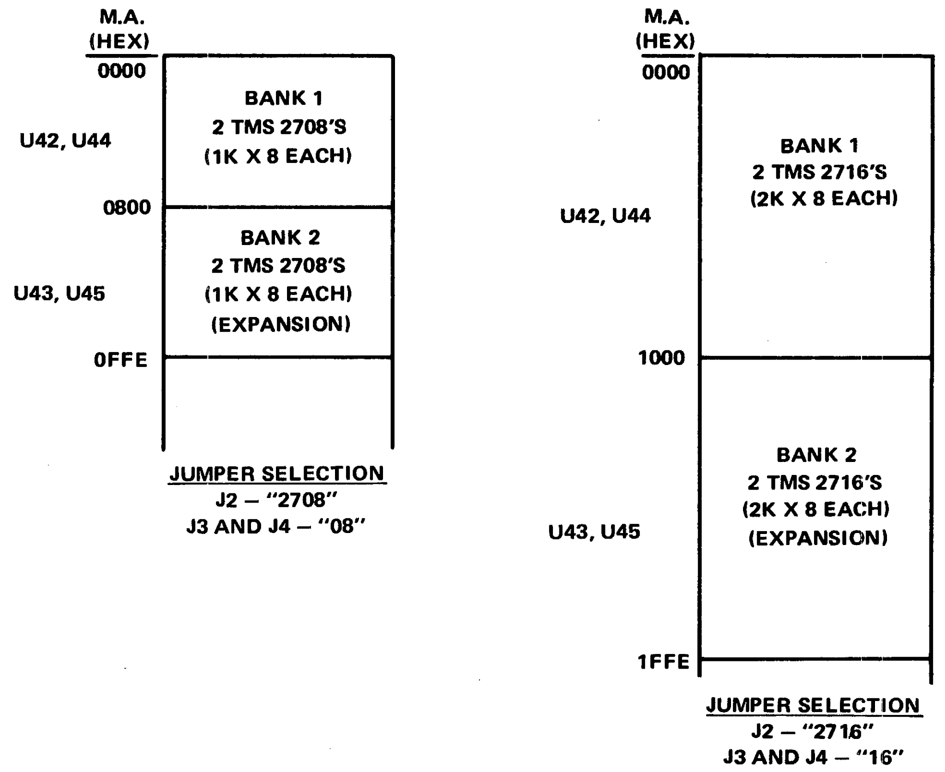
7.4 RS-232-C AND TELETYPEWRITER INTERFACES

Appendix A covers cabling for a Teletype Model 3320/5JE. To use this terminal (20 mA current loop), connect the jumper at J11.

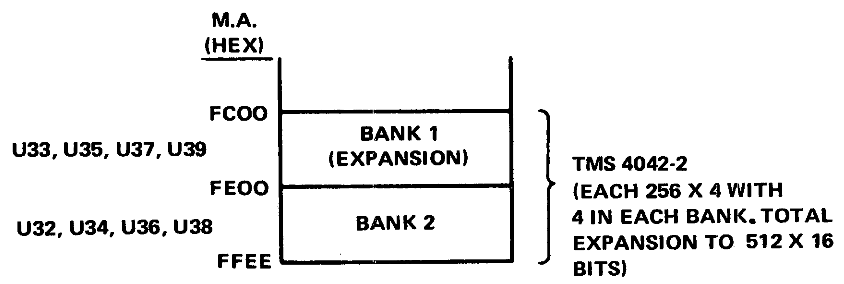
CAUTION

Verify correct voltage levels at connector P2 when attaching a teletypewriter type terminal.

Appendix B covers cabling for an RS-232-C-type terminal. To use this type of terminal, disconnect the jumper at J11.



(A) EPROM EXPANSION



A0001460

(B) RAM EXPANSION

FIGURE 7-3. MEMORY EXPANSION MAPS

7.5 EXTERNAL SYSTEM RESET

External switches can reset the system through connections at connector P1. They activate the following signals as shown in Appendix F (Schematics).

- $\overline{\text{RESTART}}$.B. This causes a load function. A 39 μF tantalum capacitor is required at C5 to debounce the switch. See Figure 7-2 for part placement.
- $\overline{\text{PRES}}$.B. This causes reset function. A 39 μF tantalum capacitor is required at C6 to debounce the switch. See Figure 7-2 for part placement.

7.6 MEMORY MAP CHANGE

On-board memory chip and CRU device addressing is through bit patterns in two PROMs, a 74S287 and a 74S288 as shown in Appendix F (Schematics). This memory map may be altered by the substitution of PROM's with the desired configuration.

7.7 TM 990/402 LINE-BY-LINE ASSEMBLER

A line-by-line assembler is available, programmed on two TMS 2708 EPROM's. It will assemble each instruction as it is input by the user. The resulting machine code will be printed on the terminal and placed in continuous memory locations. The *TIBUG* monitor must be present to use the assembler.

No relocatable labels can be used. Jump instructions use dollar-sign plus or minus byte displacements, and symbolic addresses are input as absolute locations. Error codes identify syntax errors (illegal op code), displacement errors (jump instructions), and range errors (e.g., R33). Figures 4-17 and 7-4 are examples of assembly outputs using the line-by-line assembler.

7.8 TM 990/301 MICROTERMINAL

An alternate to a hard-copy terminal is a TM 990/301 microterminal for user communication to and from the TM 990/100M. The size of a hand-held calculator, the TM 990/301 uses its light-emitting diode (LED) display to show hexadecimal or decimal values. Features of the TM 990/301 include:

- Hexadecimal to signed decimal and signed decimal to hexadecimal conversion of displayed value.
- Display and change contents of Workspace Pointer, Program Counter, Status Register, or CRU ports.
- Increment through memory displaying contents.
- Display and change contents of memory addresses.
- Halt or single step user program execution.
- Begin program execution.
- Keyboard values 0 through F_{16}.

This microterminal comes with its own cable which attaches to the 25-pin connector P2. To supply power to the microterminal, place jumpers at J13, J14, and J15. When the microterminal is not connected, make sure that these jumpers are disconnected. Jumper J7 must be in the EIA position for microterminal operation. See Figure 7-2.

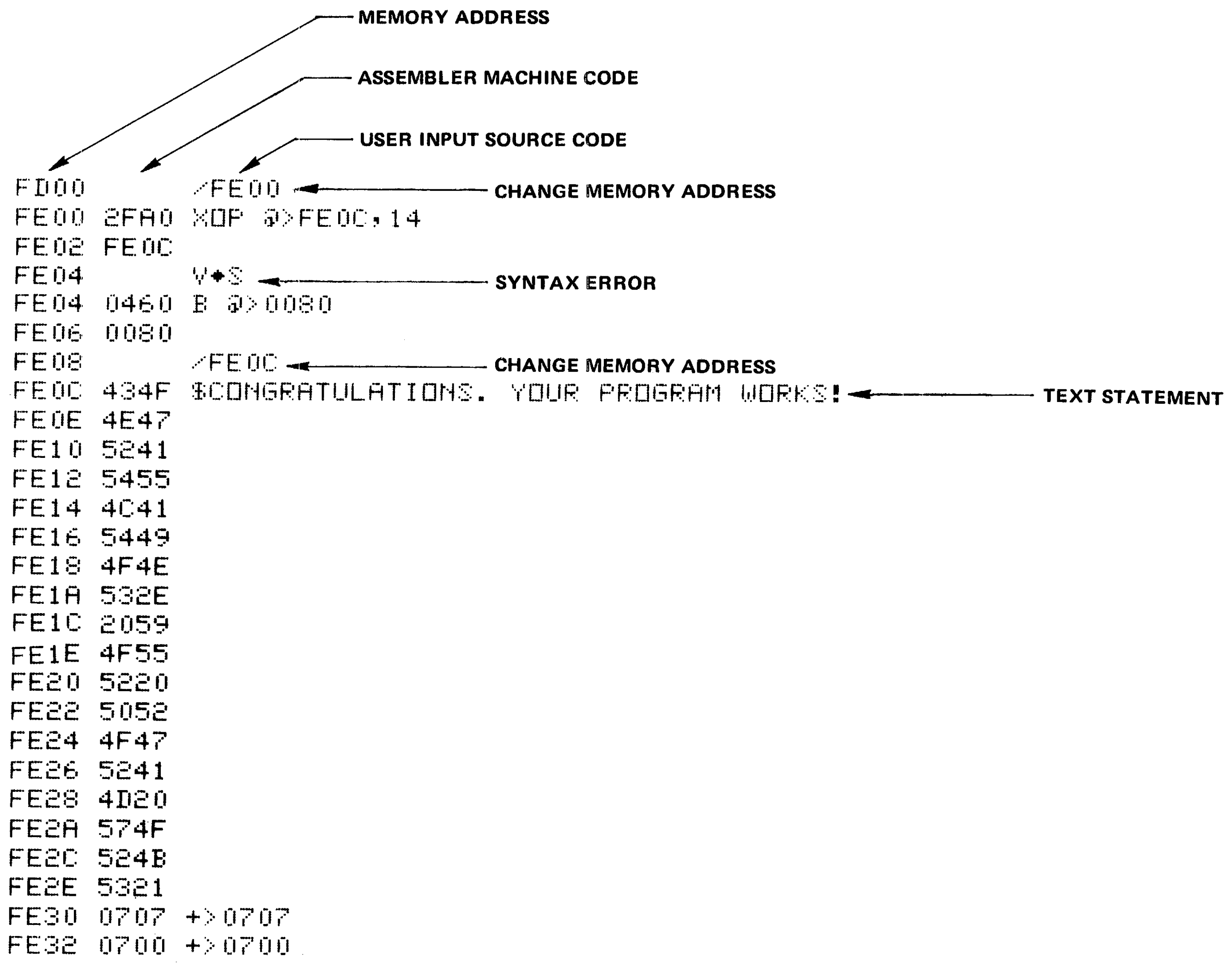
Figure 7-5 shows the microterminal and cabling to the TM 990/100M.

7.9 OEM CHASSIS

An original equipment manufacturer (OEM) chassis is available. It features slots for four boards, a motherboard backplane interfacing to P1 on the board, and a terminal strip for power, $\overline{\text{PRES}}$.B, $\overline{\text{INT1}}$.B, and $\overline{\text{RESTART}}$.B. A dimensional drawing of the OEM chassis is shown in Figure 7-6. A schematic of the backplane is shown in Figure 7-7. P1 pin assignments are listed in Table H-1 of Appendix H.

NOTE

Dimension between card slots is one inch.



```
FD00         /FE00
FE00 2FA0 XOP @>FE0C,14
FE02 FE0C
FE04         V*S
FE04 0460 B @>0080
FE06 0080
FE08         /FE0C
FE0C 434F $CONGRATULATIONS. YOUR PROGRAM WORKS!
FE0E 4E47
FE10 5241
FE12 5455
FE14 4C41
FE16 5449
FE18 4F4E
FE1A 532E
FE1C 2059
FE1E 4F55
FE20 5220
FE22 5052
FE24 4F47
FE26 5241
FE28 4D20
FE2A 574F
FE2C 524B
FE2E 5321
FE30 0707 +>0707
FE32 0700 +>0700
```

FIGURE 7-4. LINE-BY-LINE ASSEMBLER OUTPUT

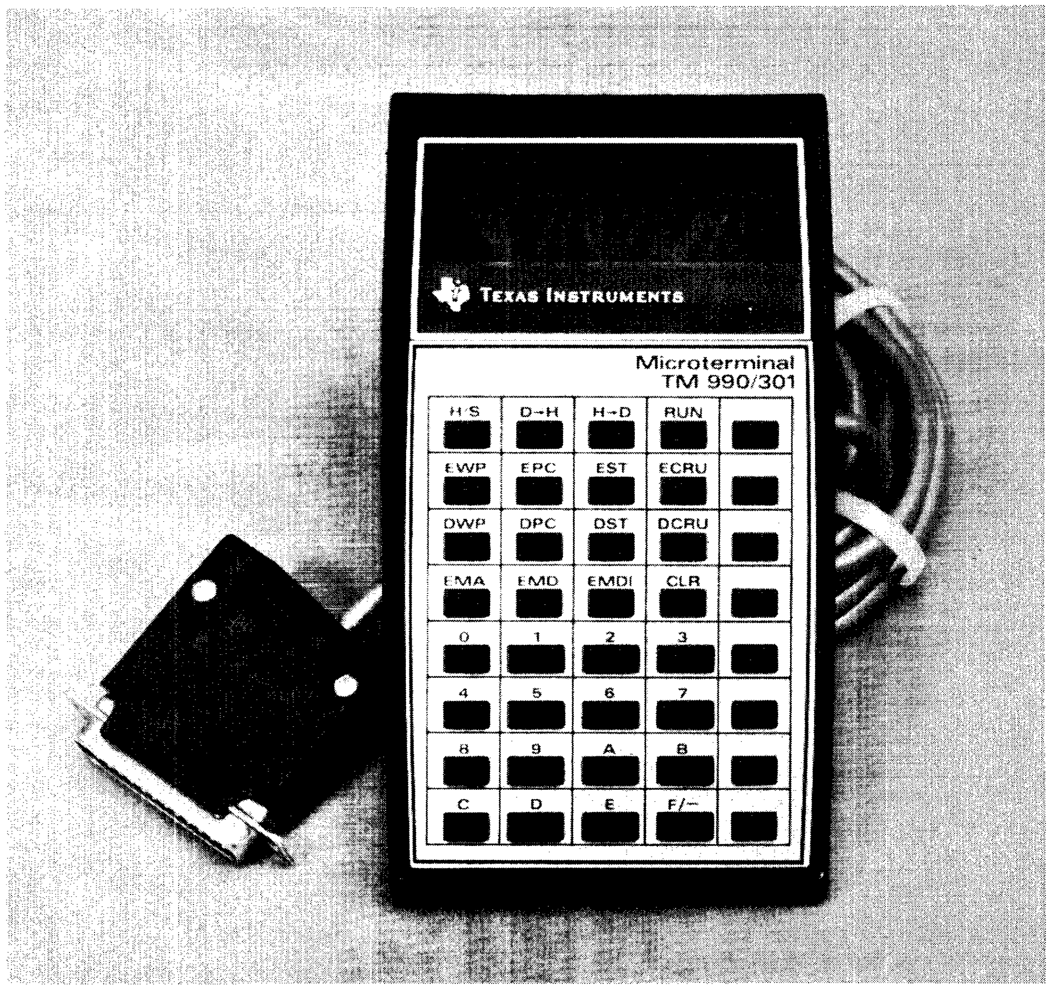
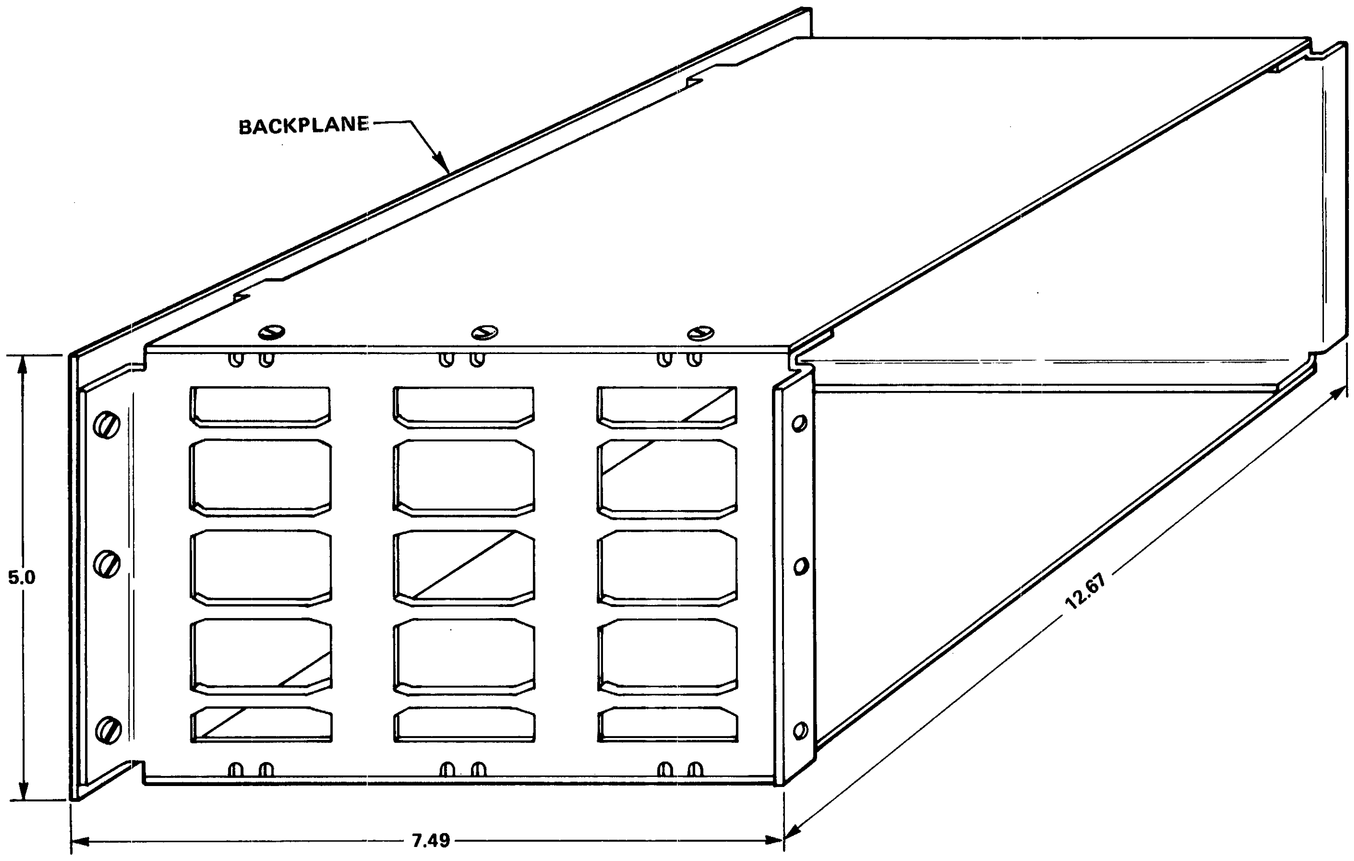


FIGURE 7-5. TM 990/301 MICROTERMINAL

7.10 INTERRUPT FROM TMS 9902

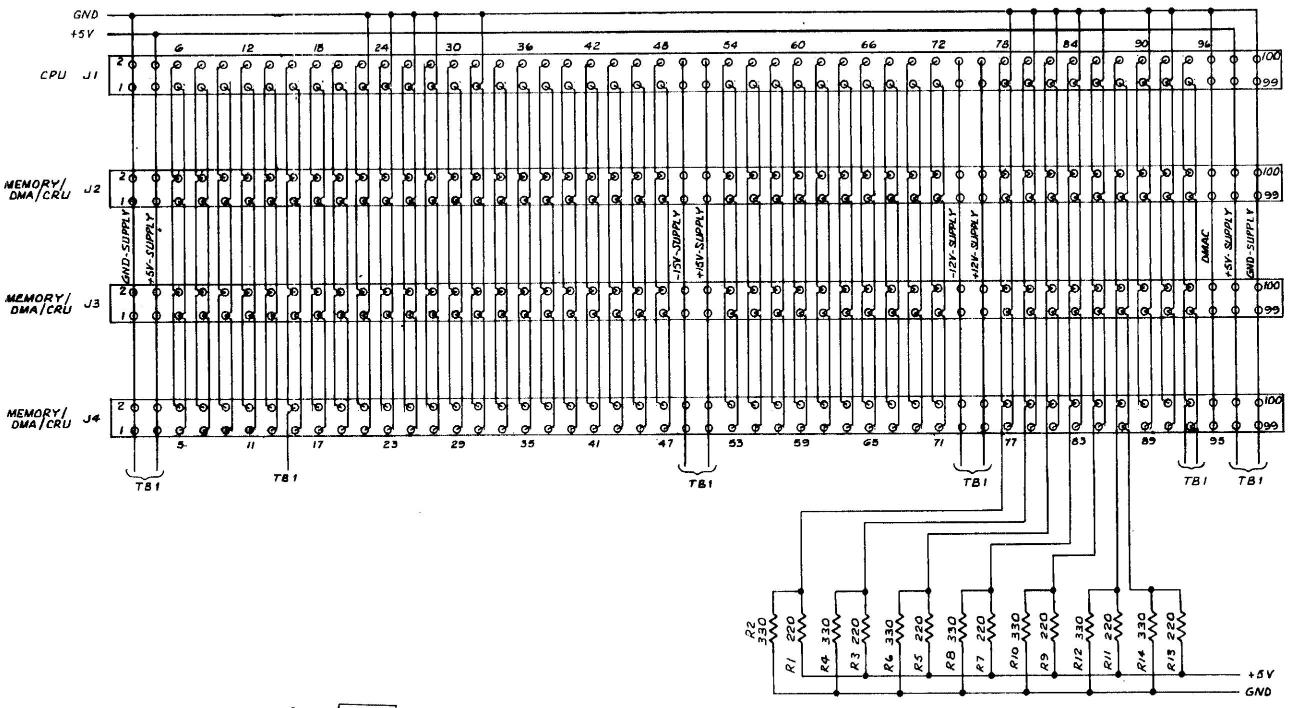
An on-board communications interrupt is issued by the TMS 9902 as explained in paragraph 6.6. When operating under the *TIBUG* monitor, place jumper J1 in position "P1-18."

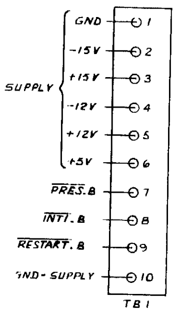


NOTES:
1. **DIMENSIONS IN INCHES**
2. **DISTANCE BETWEEN SLOTS IS 1 INCH.**

A0001463

FIGURE 7-6. OEM CHASSIS





TERMINAL STRIP
IN BACK OF CHASSIS

NOTE: BACKPLANE PIN ASSIGNMENTS LISTED IN TABLE H-1 (APPENDIX H).

FIGURE 7-7. OEM CHASSIS BACKPLANE SCHEMATIC

APPENDIX A

WIRING TELETYPE MODEL 3320/5JE FOR TM 990/100M

A-1 GENERAL

Figure A-1 shows the wiring configuration required to connect a 3320/SJE Teletype in a 20 mA current loop with a TM 990/100M. Other teletypewriter models may require different connections; therefore, consult the manufacturer for correct wiring of other models. Teletypewriters can be used with Assembly No. 999211-0001 only.

CAUTION

Note the 117 Vac connection at pins 1 and 2. Be sure that this voltage is not accidently wired to the TM 990/100M board.

A-2 CONNECTIONS

The following assumes that the teletypewriter is wired as it came from the factory.

(1) Locate the 151411 terminal block at the left rear (viewed from the rear) of the machine (Figure A-1).

(2) Move the white/blue wire from terminal 4 to terminal 5 on the terminal block.

(3) Move the brown/yellow wire from terminal 3 to terminal 5 on the terminal block.

(4) Move the purple wire from terminal 8 to terminal 9 on the terminal block (for 20 mA neutral signaling).

(5) Locate the power resistor behind the teletype power supply. Remove the blue wire from the 750 ohm tap and connect it to the 1450 ohm tap, as shown in Figure A-2.

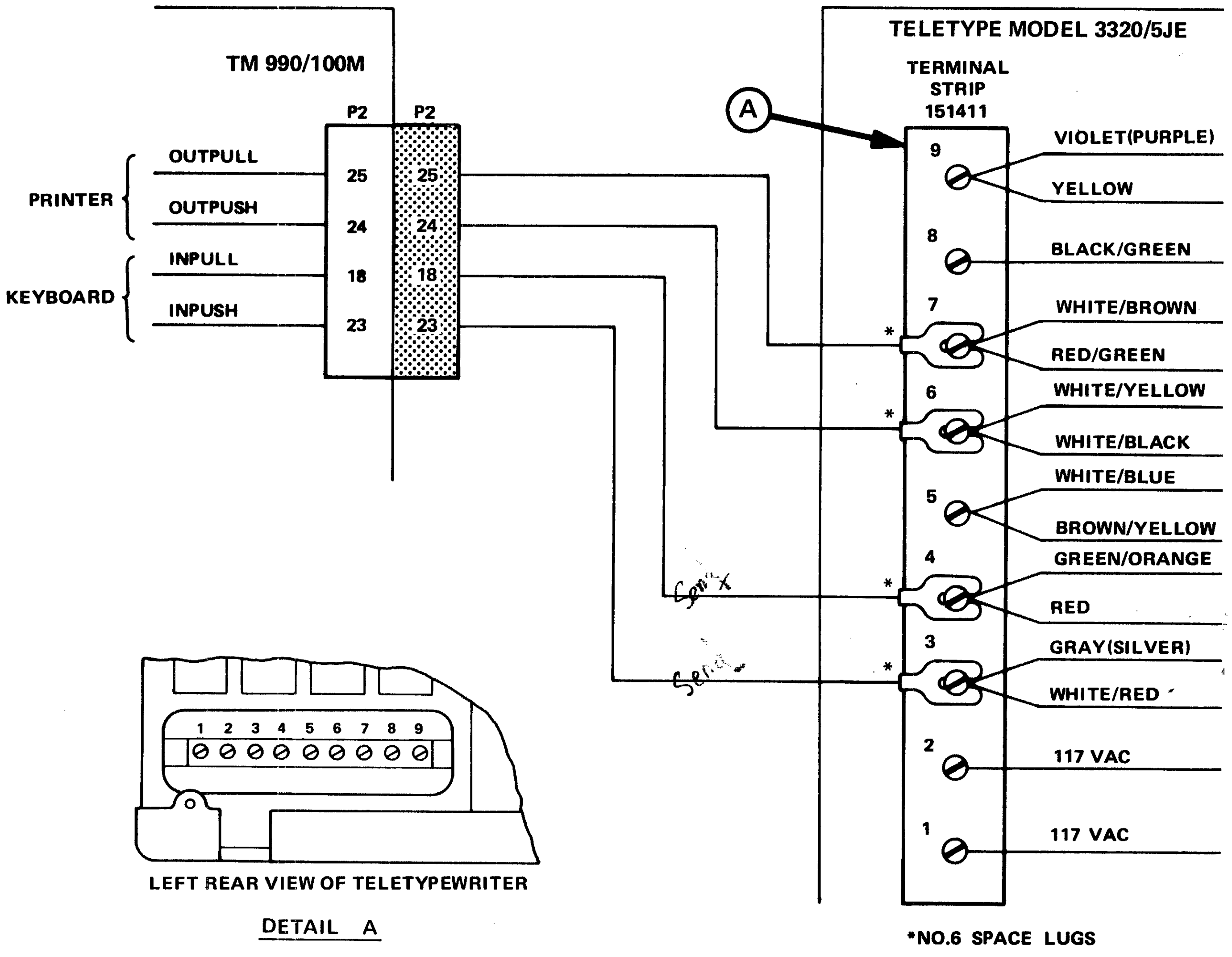
(6) Check pins 3, 4, 6, and 7 at terminal strip 151411. Voltage to ground must be zero with power applied. If not, do not connect to the TM 990/100M.

NOTE

For teletypewriter operation jumper J11 must be installed and J7 must be in the EIA position.

A-3 TROUBLESHOOTING

If the printer continues to chatter after the RESET switch on the TM 990/100M has been activated, reverse connections 6 and 7 at the terminal strip.

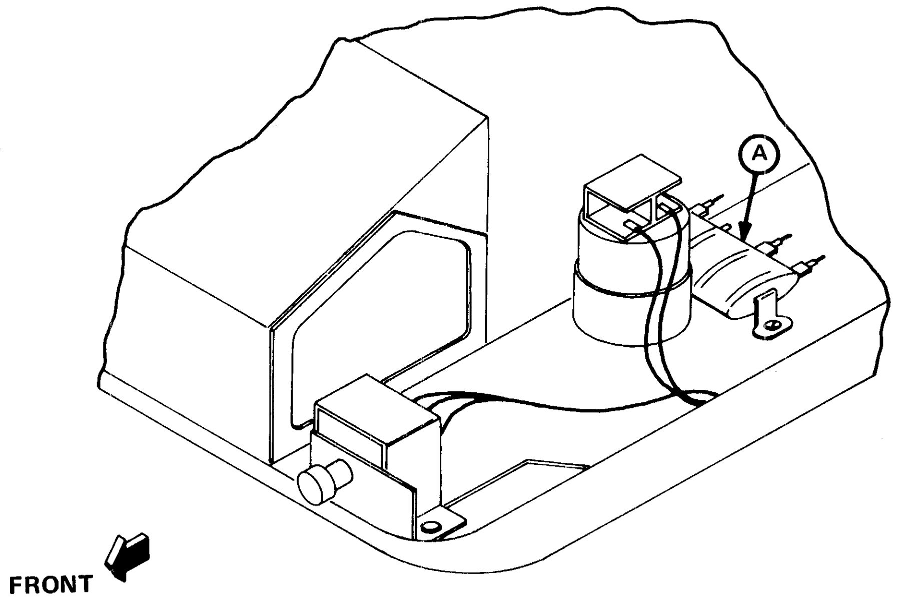


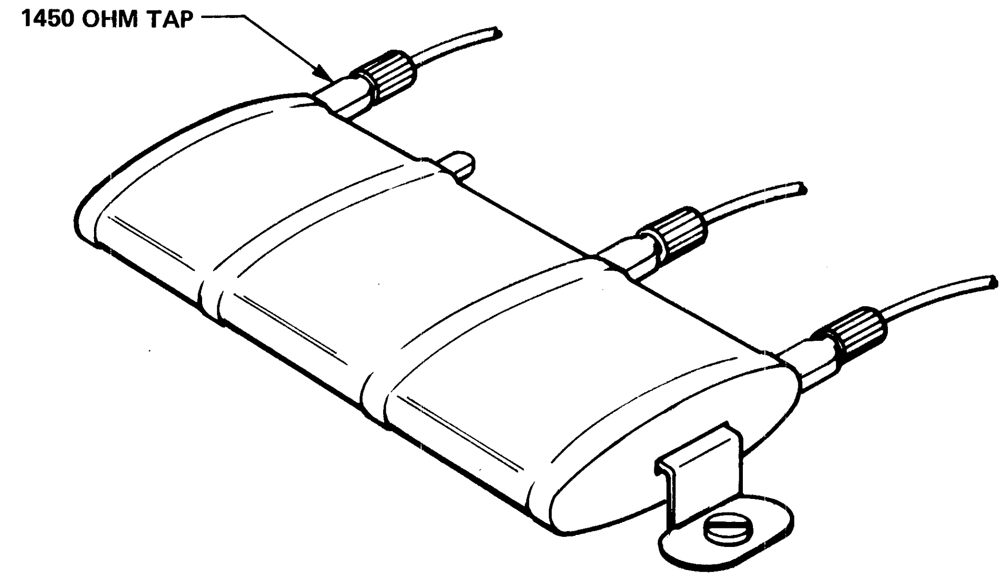
A0001412

FIGURE A-1. TELETYPEWRITER TERMINAL STRIP CONNECTIONS

TELETYPE CABLE COLOR CODE

KBD	TERM 3 - RED	PLUG	PIN 23
	TERM 4 - BRN	PLUG	PIN 18
PRINTR	TERM 6 - GRN	PLUG	PIN 24
	TERM 7 - BLU	PLUG	PIN 25





A0001413

DETAIL A

FIGURE A-2. TELETYPEWRITER RESISTOR CONNECTION

APPENDIX B

EIA RS-232-C CABLING

Figure B-1 shows the wiring for the 743 KSR cable attached between connector P2 on the TM 990/100M and a 743 KSR data terminal. Also shown is the relationship between cable wires and signals to the serial interface, the TMS 9902. Figure B-2 shows the cable configuration for the 733 data terminal.

NOTE

When using an RS-232-C device, disconnect jumper J11 and insert jumper J7 in the EIA position. See Figure 7-2.

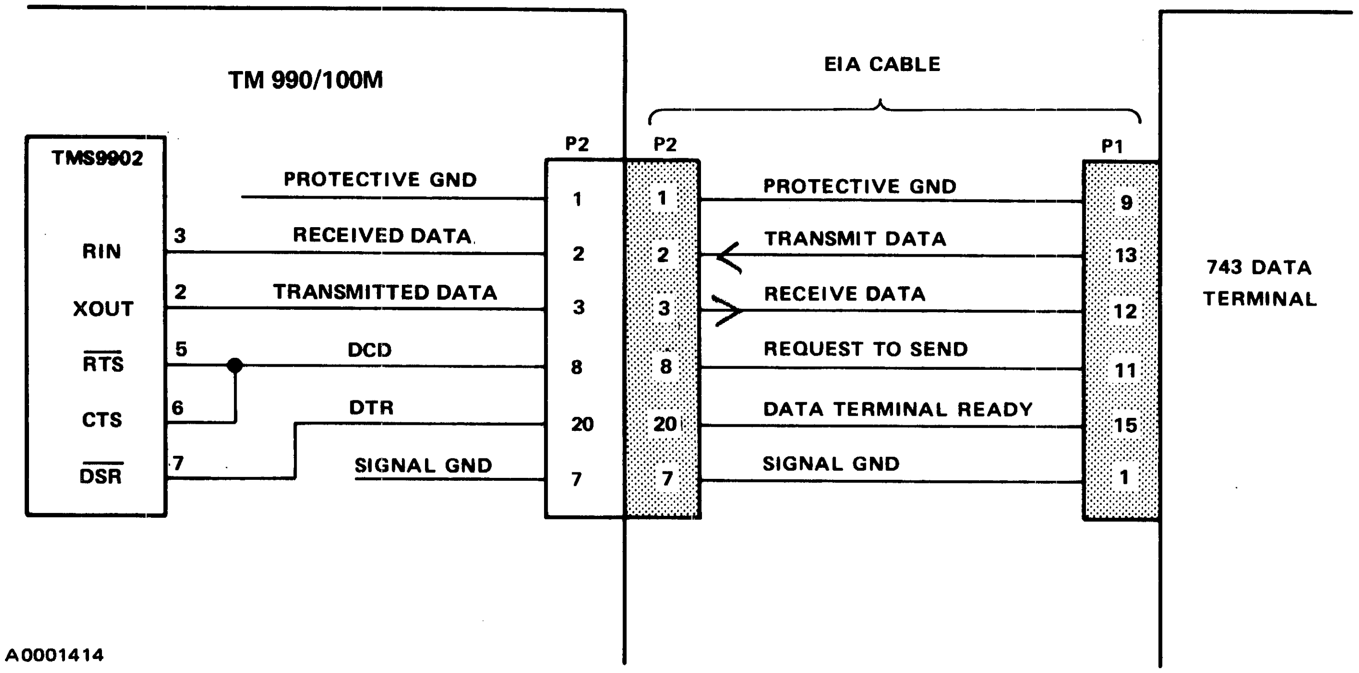


FIGURE B-1. EIA RS-232-C CABLING FOR 743 DATA TERMINAL

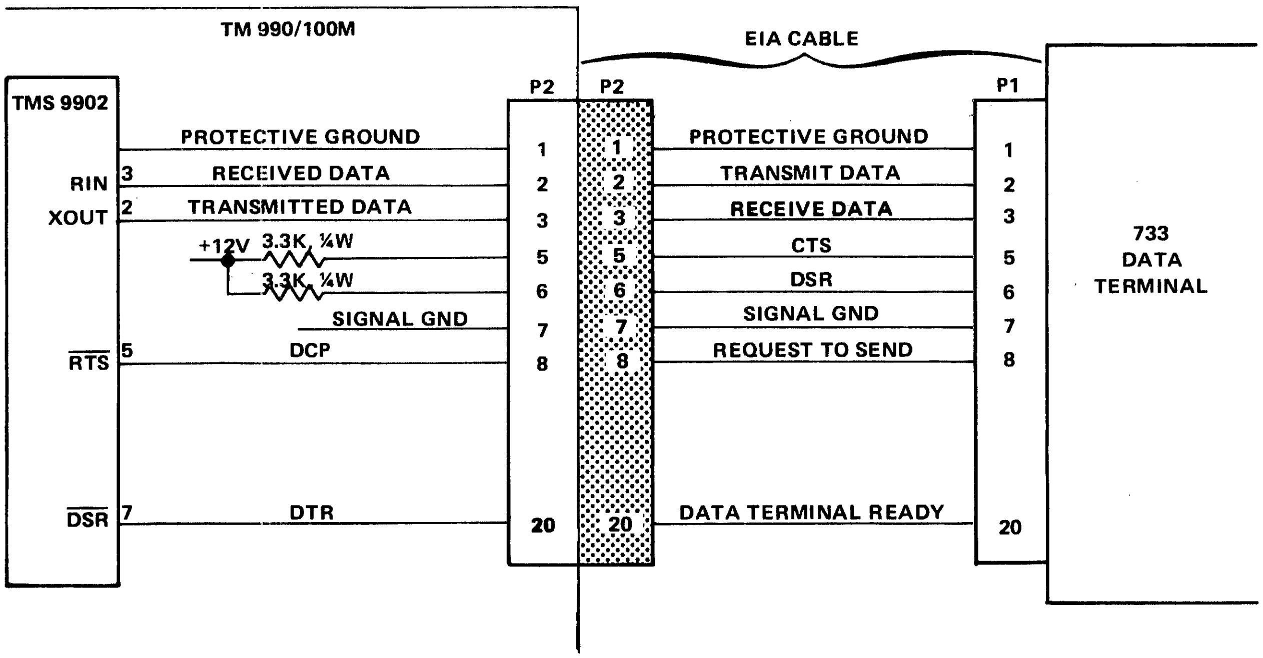


FIGURE B-2. EIA RS-232-C CABLING FOR 733 DATA TERMINAL

APPENDIX C

ASCII CODE

TABLE C-1. *ASCII CONTROL CODES

CONTROL		BINARY CODE	HEXADECIMAL CODE
NUL	– Null	000 0000	00
SOH	– Start of heading	000 0001	01
STX	– Start of text	000 0010	02
ETX	– End of text	000 0011	03
EOT	– End of transmission	000 0100	04
ENQ	– Enquiry	000 0101	05
ACK	– Acknowledge	000 0110	06
BEL	– Bell	000 0111	07
BS	– Backspace	000 1000	08
HT	– Horizontal tabulation	000 1001	09
LF	– Line feed /NL	000 1010	0A
VT	– Vertical tab	000 1011	0B
FF	– Form feed	000 1100	0C
CR	– Carriage return	000 1101	0D
SO	– Shift out	000 1110	0E
SI	– Shift in	000 1111	0F
DLE	– Data link escape	001 0000	10
DC1	– Device control 1	001 0001	11
DC2	– Device control 2	001 0010	12
DC3	– Device control 3	001 0011	13
DC4	– Device control 4 (stop)	001 0100	14
NAK	– Negative acknowledge	001 0101	15
SYN	– Synchronous idle	001 0110	16
ETB	– End of transmission block	001 0111	17
CAN	– Cancel	001 1000	18
EM	– End of medium	001 1001	19
SUB	– Substitute	001 1010	1A
ESC	– Escape	001 1011	1B
FS	– File separator	001 1100	1C
GS	– Group separator	001 1101	1D
RS	– Record separator	001 1110	1E
US	– Unit separator	001 1111	1F
DEL	– Delete, rubout	111 1111	7F

*American Standards Institute Publication X3.4-1968

TABLE C-2. *ASCII CHARACTER CODE

CHARACTER	BINARY CODE	HEXADECIMAL CODE	CHARACTER	BINARY CODE	HEXADECIMAL CODE
Space	010 0000	20	P	101 0000	50
!	010 0001	21	Q	101 0001	51
" (dbl. quote)	010 0010	22	R	101 0010	52
#	010 0011	23	S	101 0011	53
$	010 0100	24	T	101 0100	54
%	010 0101	25	U	101 0101	55
&	010 0110	26	V	101 0110	56
' (sgl. quote)	010 0111	27	W	101 0111	57
(	010 1000	28	X	101 1000	58
)	010 1001	29	Y	101 1001	59
* (asterisk)	010 1010	2A	Z	101 1010	5A
+	010 1011	2B	[	101 1011	5B
, (comma)	010 1100	2C	\	101 1100	5C
– (minus)	010 1101	2D	]	101 1101	5D
. (period)	010 1110	2E	Λ	101 1110	5E
/	010 1111	2F	_ (underline)	101 1111	5F
0	011 0000	30	`	110 0000	60
1	011 0001	31	a	110 0001	61
2	011 0010	32	b	110 0010	62
3	011 0011	33	c	110 0011	63
4	011 0100	34	d	110 0100	64
5	011 0101	35	e	110 0101	65
6	011 0110	36	f	110 0110	66
7	011 0111	37	g	110 0111	67
8	011 1000	38	h	110 1000	68
9	011 1001	39	i	110 1001	69
:	011 1010	3A	j	110 1010	6A
;	011 1011	3B	k	110 1011	6B
<	011 1100	3C	l	110 1100	6C
=	011 1101	3D	m	110 1101	6D
>	011 1110	3E	n	110 1110	6E
?	011 1111	3F	o	110 1111	6F
@	100 0000	40	p	111 0000	70
A	100 0001	41	q	111 0001	71
B	100 0010	42	r	111 0010	72
C	100 0011	43	s	111 0011	73
D	100 0100	44	t	111 0100	74
E	100 0101	45	u	111 0101	75
F	100 0110	46	v	111 0110	76
G	100 0111	47	w	111 0111	77
H	100 1000	48	x	111 1000	78
I	100 1001	49	y	111 1001	79
J	100 1010	4A	z	111 1010	7A
K	100 1011	4B	{	111 1011	7B
L	100 1100	4C	¦	111 1100	7C
M	100 1101	4D	}	111 1101	7D
N	100 1110	4E	~	111 1110	7E
O	100 1111	4F			

*American Standards Institute Publication X3.4-1968

APPENDIX D

BINARY, DECIMAL AND HEXADECIMAL NUMBERING

D-1 GENERAL

This appendix covers numbering systems to three bases (2, 10, and 16) which are used throughout this manual.

D-2 POSITIVE NUMBERS

D-2.1 DECIMAL (BASE 10). When a numerical quantity is viewed from right to left, the right-most digit represents the base number to the exponent 0. The next digit represents the base number to the exponent 1, the next to the exponent 2, then exponent 3, etc. For example, using the base 10 (decimal):

$$\begin{array}{ccccccc} 10^6 & 10^5 & 10^4 & 10^3 & 10^2 & 10^1 & 10^0 \\ X, & X & X & X, & X & X & X \end{array}$$

or

```
1,000,000
|      100,000
|      |  10,000
↓      ↓  ↓ 1000   100  10   1
X ,    X  X X   ,   X    X    X
```

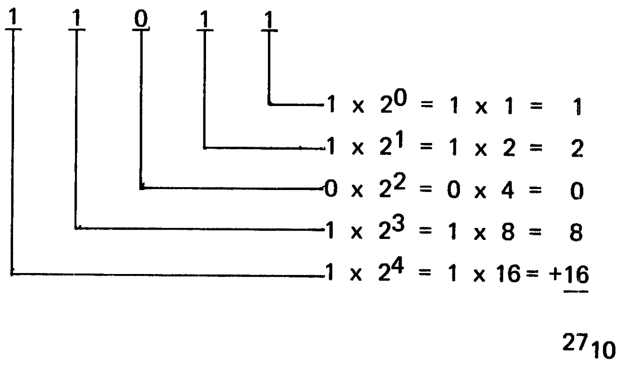
For example, 75,264 can be broken down as follows:

$$\begin{array}{lcr} 4 \times 10^0 = 4 \times 1 & = & 4 \\ 6 \times 10^1 = 6 \times 10 & = & 60 \\ 2 \times 10^2 = 2 \times 100 & = & 200 \\ 5 \times 10^3 = 5 \times 1000 & = & 5000 \\ 7 \times 10^4 = 7 \times 10{,}000 & = & +70000 \\ \hline & & 75264_{10} \end{array}$$

D-2.2 **BINARY (BASE 2).** As base 10 numbers use ten digits, base 2 numbers use only 0 and 1. When viewed from right to left, they each represent the number 2 to the powers 0, 1, 2, etc., respectively as shown below:

2^{15}		2^6	2^5	2^4	2^3	2^2	2^1	2^0
(32,768)	• • •	(64)	(32)	(16)	(8)	(4)	(2)	(1)
X	• • •	X	X	X	X	X	X	X

For example, 11011_2 can be translated into base 10 as follows:



1 1 0 1 1
$1 \times 2^0 = 1 \times 1 = 1$
$1 \times 2^1 = 1 \times 2 = 2$
$0 \times 2^2 = 0 \times 4 = 0$
$1 \times 2^3 = 1 \times 8 = 8$
$1 \times 2^4 = 1 \times 16 = +16$
27_{10}

or 11011_2 equals 27_{10}.

Binary is the language of the digital computer. For example, to place the decimal quantity 23 (23_{10}) into a 16-bit memory cell, set the bits to the following:

0															15
0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1

which is $1 + 2 + 4 + 16 = 23_{10}$.

D-2.3 **HEXADECIMAL (BASE 16).** Whereas binary uses two digits and decimal uses ten digits, hexadecimal uses 16 (0 to 9, A, B, C, D, E, and F).

The letters A through F are used to represent the decimal numbers 10 through 15 as shown on the following page.

N_{10}	N_{16}	N_{10}	N_{16}
0	0	8	8
1	1	9	9
2	2	10	A
3	3	11	B
4	4	12	C
5	5	13	D
6	6	14	E
7	7	15	F

When viewed from right to left, each digit in a hexadecimal number is a multiplier of 16 to the powers 0, 1, 2, 3, etc., as shown below:

16^3	16^2	16^1	16^0
(4096)	(256)	(16)	(1)
X	X	X	X

For example, 7 B A 5_{16} can be translated into base 10 as follows:

7 B A 5

$5 \times 16^0 = 5 \times 1 = 5$

$10 \times 16^1 = 10 \times 16 = 160$

$11 \times 16^2 = 11 \times 256 = 2\,816$

$7 \times 16^3 = 7 \times 4096 = \underline{28\,672}$

$31\,653_{10}$

or 7 B A 5_{16} equals $31{,}653_{10}$.

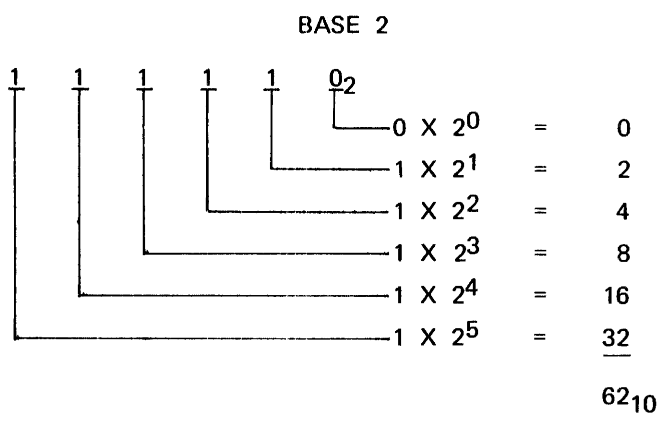
Because it would be awkward to write out 16-digit binary numbers to show the contents of a 16-bit memory word, hexadecimal is used instead. Thus

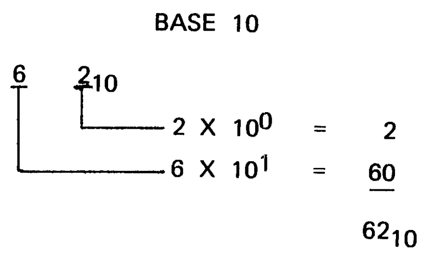
$003E_{16}$ or > 003E (> indicates hexadecimal)

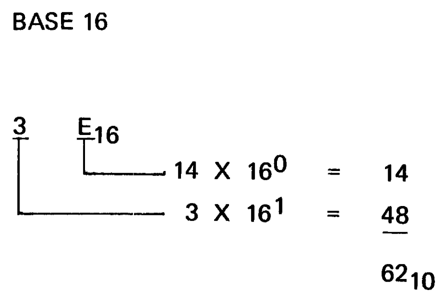
is used instead of

$0000\ 0000\ 0011\ 1110_2$

to represent 62_{10} as computed below:







Note that separating the 16 binary bits into four-bit parts facilitates recognition and translation into hexadecimal.

$$\begin{array}{cccc} 0000 & 0000 & 0011 & 1110_2 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 0 & 0 & 3 & E_{16} \end{array} \quad \text{or} \quad \begin{array}{cccc} C & 7 & B & F_{16} \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 1100 & 0111 & 1011 & 1111_2 \end{array}$$

Table D-1 is a conversion chart for converting decimal to hexadecimal and vice versa. Table D-2 shows binary, decimal and hexadecimal equivalents for numbers 0 to 15. Note that Table D-1 is divided into four parts, each part representing four of the 16-bits of a memory cell or word (bits 0 to 15 with bit 0 being the most significant bit (MSB) and bit 15 being the least significant bit (LSB). Note that the MSB is on the left and represents the highest power of 2 and the LSB on the right represents the 0 power of 2 ($2^0 = 1$). As explained later, the MSB can also be used to signify number polarity (+ or –).

NOTE

To convert a binary number to decimal or hexadecimal, convert the *positive* binary value as described in Section D-4.

TABLE D-1. HEXADECIMAL/DECIMAL CONVERSION CHART

	MSB						LSB	
	16^3		16^2		16^1		16^0	
BITS	0 1 2 3		4 5 6 7		8 7 8 11		12 13 14 15	
	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
	0	0	0	0	0	0	0	0
	1	4 096	1	256	1	16	1	1
	2	8 192	2	512	2	32	2	2
	3	12 288	3	768	3	48	3	3
	4	16 384	4	1 024	4	64	4	4
	5	20 480	5	1 280	5	80	5	5
	6	24 576	6	1 536	6	96	6	6
	7	28 672	7	1 792	7	112	7	7
	8	32 768	8	2 048	8	128	8	8
	9	36 864	9	2 304	9	144	9	9
	A	40 960	A	2 560	A	160	A	10
	B	45 056	B	2 816	B	176	B	11
	C	49 152	C	3 072	C	192	C	12
	D	53 248	D	3 328	D	208	D	13
	E	57 344	E	3 584	E	224	E	14
	F	61 440	F	3 840	F	240	F	15

To convert a number from hexadecimal, add the decimal equivalents for each hexadecimal digit. For example, $7A82_{16}$ would equal in decimal 28,672 + 2,560 + 128 + 2. To convert hexadecimal to decimal, find the nearest decimal number in the above table less than or equal to the number being converted. Set down the hexadecimal equivalent then subtract this number from the nearest decimal number. Using the remainder(s), repeat this process. For example:

$31{,}362_{10} = 7000_{16} + 2690_{10}$
$2{,}690_{10} = A00_{16} + 130_{10}$
$130_{10} = 80_{16} + 2_{10}$
$2_{10} = 2_{16}$

$$\begin{array}{r} 7000 \\ A00 \\ 80 \\ \underline{2} \\ 7A82_{16} \end{array}$$

TABLE D-2. BINARY, DECIMAL, AND HEXADECIMAL EQUIVALENTS

BINARY (N_2)	DECIMAL (N_{10})	HEXADECIMAL (N_{16})
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F
10000	16	10
10001	17	11
10010	18	12
10011	19	13
10100	20	14
10101	21	15
10110	22	16
10111	23	17
11000	24	18
11001	25	19
11010	26	1A
11011	27	1B
11100	28	1C
11101	29	1D
11110	30	1E
11111	31	1F
100000	32	20

D-3 ADDING AND SUBTRACTING BINARY

Adding and subtracting in binary uses the same conventions for decimal: carrying over in addition and borrowing in subtraction.

Basically,

$$\begin{array}{r} 0 \\ +\,1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\,1 \\ \hline 10 \end{array} \quad \text{(the carry, 1, is carried to the left)}$$

$$\begin{array}{r} 10 \\ -\,1 \\ \hline 01 \end{array} \quad \text{(1 is borrowed from top left)}$$

$$\begin{array}{rl} \left.\begin{array}{r} 1 \\ 1 \end{array}\right\} & = 0 + \text{carry } 1 \\ +\,1 & = 0 \text{ (from above)} + 1 = 1 \\ \hline \underline{1}1 & \\ \llcorner\text{carry} & \end{array}$$

$$\begin{array}{r} 11 \\ 1 \\ +\;\;1 \\ \hline \underline{10}1 \end{array} \qquad \text{carry } 1 + 1 = 10$$

$$\begin{array}{rl} \left.\begin{array}{r} 1 \\ 1 \end{array}\right\} & = 0 + 1 \text{ carry} \\ \left.\begin{array}{r} 1 \\ +\,1 \end{array}\right\} & = 0 + 1 \text{ carry} \\ \hline \underline{10}0 & \\ & 0 + 0 = 0 \\ & \text{carry } 1 + \text{carry } 1 \end{array}$$

$$\left.\begin{array}{r} 1000 \\ -\,1 \\ \hline 0111 \end{array}\right\} \text{ Borrow the 1 } \left\{\begin{array}{r} \overset{1}{0110} \\ -\,1 \\ \hline 0111 \end{array}\right.$$

D-4 POSITIVE/NEGATIVE CONVERSION (BINARY). To compute the negative equivalent of a positive binary or hexadecimal number, or interpret a binary or hexadecimal negative number (determine its positive equivalent) use the two's complement of the binary number.

NOTE

To convert a binary number to decimal, convert the *positive* binary value (*not* the negative binary value) and add the sign.

Two's complementing a binary number includes two simple steps:

a. Obtain one's complement of the number (1's become 0's, 0's becomes 1's) (invert bits).

b. Add 1 to the one's complement.

For example, with the MSB (left-most bit) being a sign bit:

010	$(+2_2)$	111	(-1_2)	110	(-2_2)	101	(-3_2)
101	Invert	000	Invert	001	Invert	010	Invert
+ 1	Add 1	+ 1	Add 1	+ 1	Add 1	+ 1	
110	(-2_2)	001	$(+1_2)$	010	$(+2_2)$	011	$(+3_2)$

This can be expanded to 16-bit positive numbers:

$(=39F6_{16})$	0011	1001	1111	0110	$(39F6_{16}$	=	$+14{,}838_{10})$	
	1100	0110	0000	1001	Invert			
				+1	Add 1			
$(=C60A_{16})$	1100	0110	0000	1010	$(C60A_{16}$	=	$-14{,}838_{10})$	Two's Complement
	SIGN BIT(−)							

And to 16-bit negative numbers:

$(=C60A_{16})$	1100	0110	0000	1010	$(C60A_{16}$	=	$-14{,}838_{10})$	
	0011	1001	1111	0101	Invert			
				+1	Add 1			
$(=39F6_{16})$	0011	1001	1111	0110	$(39F6_{16}$	=	$+14{,}838_{10})$	Two's Complement
	SIGN BIT(+)							

APPENDIX E

PARTS LIST (TM990/100M-1)

TABLE E-1. PARTS FOR ALL BOARDS

SYMBOL	DESCRIPTION	QTY.
C1 to C4	Capacitor, 22 μF, tantalum electrolytic	4
C7 to C22, C24 to C42	Capacitor, 0.047 μF, axial lead	35
C23	Capacitor, 18 pF, ceramic disc	1
CR1	Diode, 1N914B	1
L1	Inductor, 0.33 μH	1
P2	Connector, EIA, 25-pin socket	1
R1, R4, R5	Resistor, 68 ohms, 1/4 W, 5%	3
R2, R9, R11	Resistor, 220 ohms, 1/4 W, 5%	3
R3, R8, R10	Resistor, 330 ohms, 1/4 W, 5%	3
R6, R12, R13, R14, R19	Resistor, 4.7 kilohms, 1/4 W, 5%	5
R7	Resistor, 1 kilohm , 1/4 W, 5%	1
R15 to R18	Resistor, 10 ohms, 1/4 W, 5%	4
R20, R34, R35	Resistor, 3.3 kilohms, 1/4 W, 5%	3
R21	Resistor, 33 kilohms, 1/4 W, 5%	1
S1	Switch, SPDT	1

U1	Resistor Pack, 4.7 kilohms, 16-pin	1
U2	74LS241N, octal buffer	1
U3 to U10	74LS243N, quad bidirectional buffer	8
U11, U14	7438N, quad, 2-input NAND gate, open collector	2
U12	75140N, receiver	1
U13, U21, U27	74LS04N, hex inverter	3
U15	TMS 9901, programmable systems interface	1
U16	TMS 9900, central processing unit	1
U17	74S287N, PROM, 256 x 4 bits	1
U18	74LS20N, dual 4-input NAND gate	1
U19	74LS362N, clock generator	1
U20	74LS138N, 3 to 8 decoder	1
U22, U26, U30, U31	74LS74AN, dual D flip-flip	4
U23	74S288N, PROM, 32 x 8	1
U25	Resistor pack, 4.7 kilohms, 14 pin	1
U28	74LS132N, quad, 2-input NAND gate, Schmitt trigger	1
U29	74LS08N, quad, 2-input AND gate	1
U32, U34, U36, U38	TMS 4042-2 RAM, 256 x 4 bits	4
U40	TMS 9902, asynchronous communications controller	1
U41	75189N, EIA driver	1
U46	75188N, EIA driver	1

VR1	Converter, −5 V, LM7905C	1
XU15	40-pin socket, low profile	1
XU16	64-pin socket, low profile	1
XU17, XU23	16-pin socket, low profile	2
XU19, XU40	20-pin socket, low profile	2
XU32 to XU39	18-pin socket, low profile	8
XU42 to XU45	24-pin socket, low profile	4
Y1	Crystal, 48 MHz, 3 overtone	1

TABLE E-2. ADDITIONAL PARTS FOR ASSEMBLY 999211-0001 (TTY INTERFACE)

SYMBOL	DESCRIPTION	QTY.
Q1	Transistor, 2N2905A, PNP	1
R30	Resistor, 560 ohms, 1/2 W, 5%	1
R31	Resistor, 2.7 kilohms, 1/2 W, 5%	1
R32	Resistor, 330 ohms, 1/2 W, 5%	1
U42, U44	TMS 2708 EPROM (1024 x 8 bits each) with TIBUG monitor	2

TABLE E-3. ADDITIONAL PARTS FOR ASSEMBLIES 999211-0002 AND 999211-0003 (MULTIDROP INTERFACE)

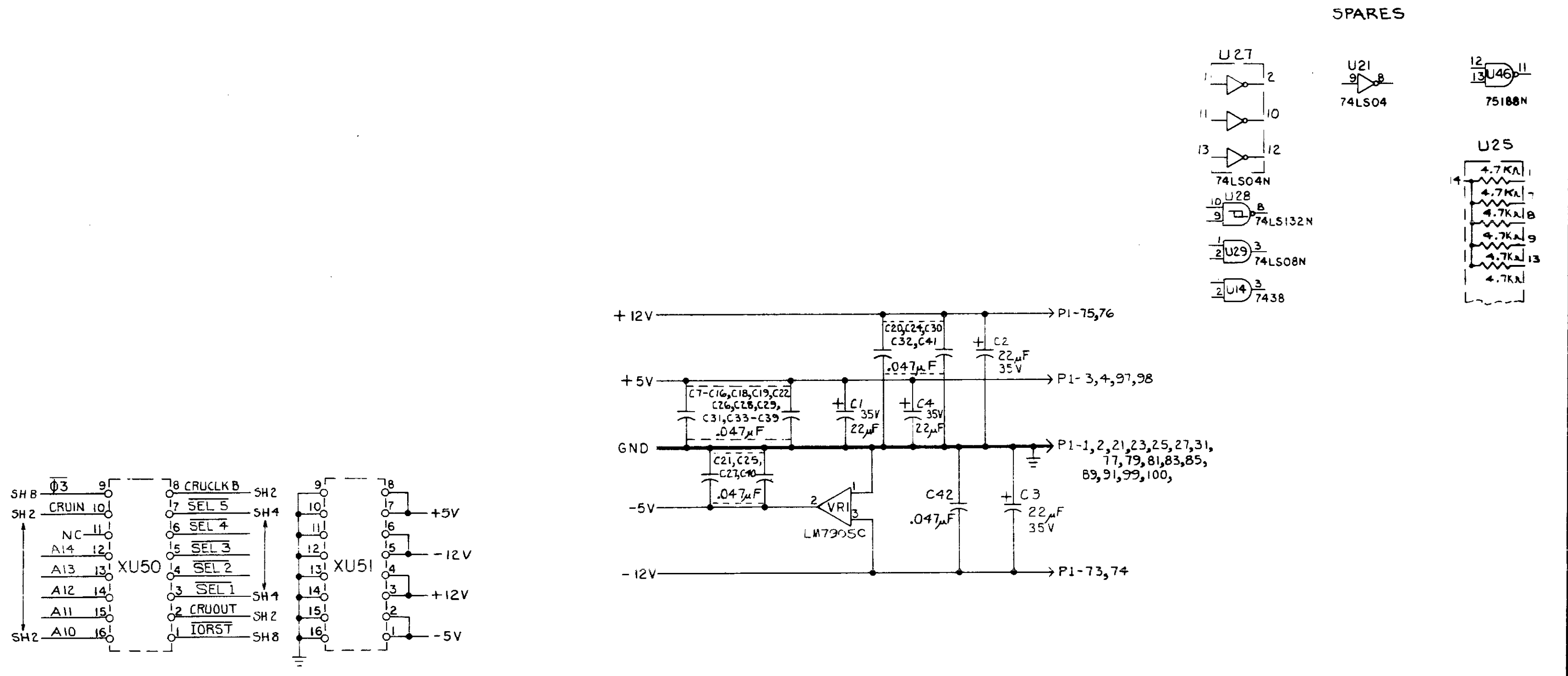
SYMBOL	DESCRIPTION	QTY.
CR2, CR3	Zener diode, 3.3 V	2
R22, R24, R26, R28	Resistor, 330 ohms, 1/4 W, 5%	4
R23, R25, R27, R29	Resistor, 27 kilohms, 1/4 W, 5%	4
U42, U44	TMS 2708 EPROM (1024 x 8 bits each)	2
U47	75112, balanced line transmitter	1
U48	75107, balanced line receiver	1

TABLE E-4. ADDITIONAL PARTS FOR ASSEMBLY 999211-0003 ONLY (MULTIDROP INTERFACE)

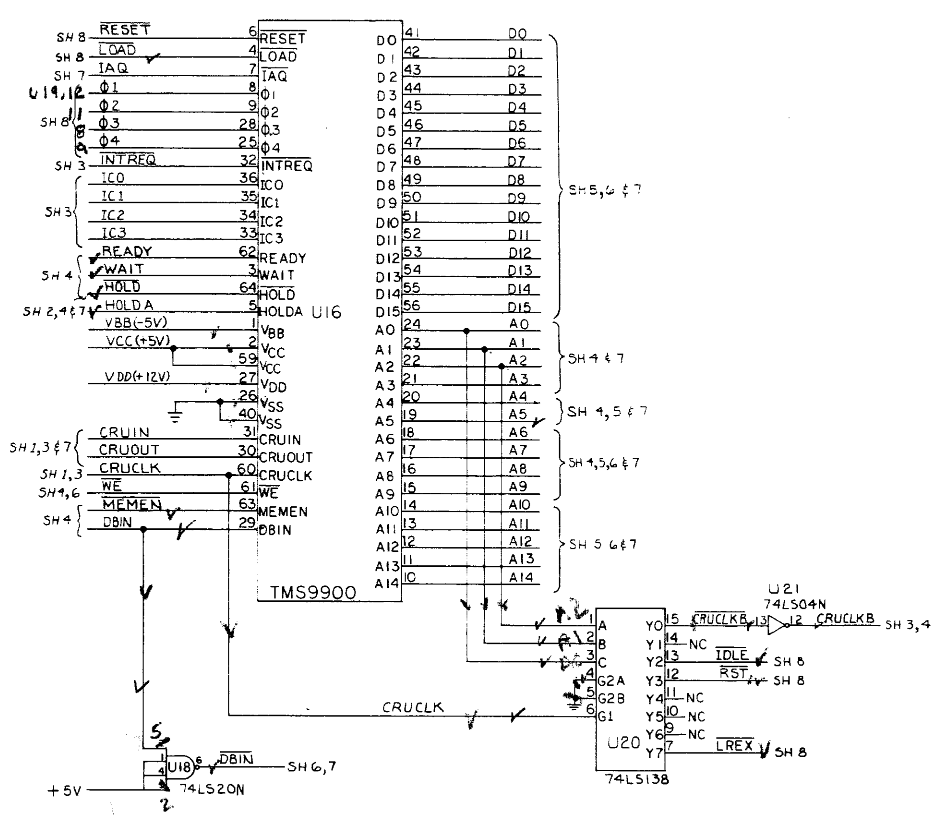
SYMBOL	DESCRIPTION	QTY.
U33, U35, U37, U39	TMS 4042-2 RAM, 256 x 4 bits each (expansion RAM)	4
U43, U45	TMS 2708 EPROM, 1024 x 8 bits each (expansion EPROM)	2

NOTES: *UNLESS OTHERWISE SPECIFIED*

1. CAPACITANCE VALUES ARE IN MICROFARADS
2. INDUCTANCE VALUES ARE IN MICROFARADS
3. 39μF C5 & C6 ELECTROLYTIC CAPACITORS ARE USER INSTALLABLE. THESE SHOULD BE TANTALUM CAPACITORS, 15V MINIMUM.
4. ON THE TM990/100M-1 ASSEMBLY, THE TTY INTERFACE IS POPLULATED. ON THE TM990/100M-2 ASSEMBLY, THE MULTI-DROP INTERFACE IS POPULATED

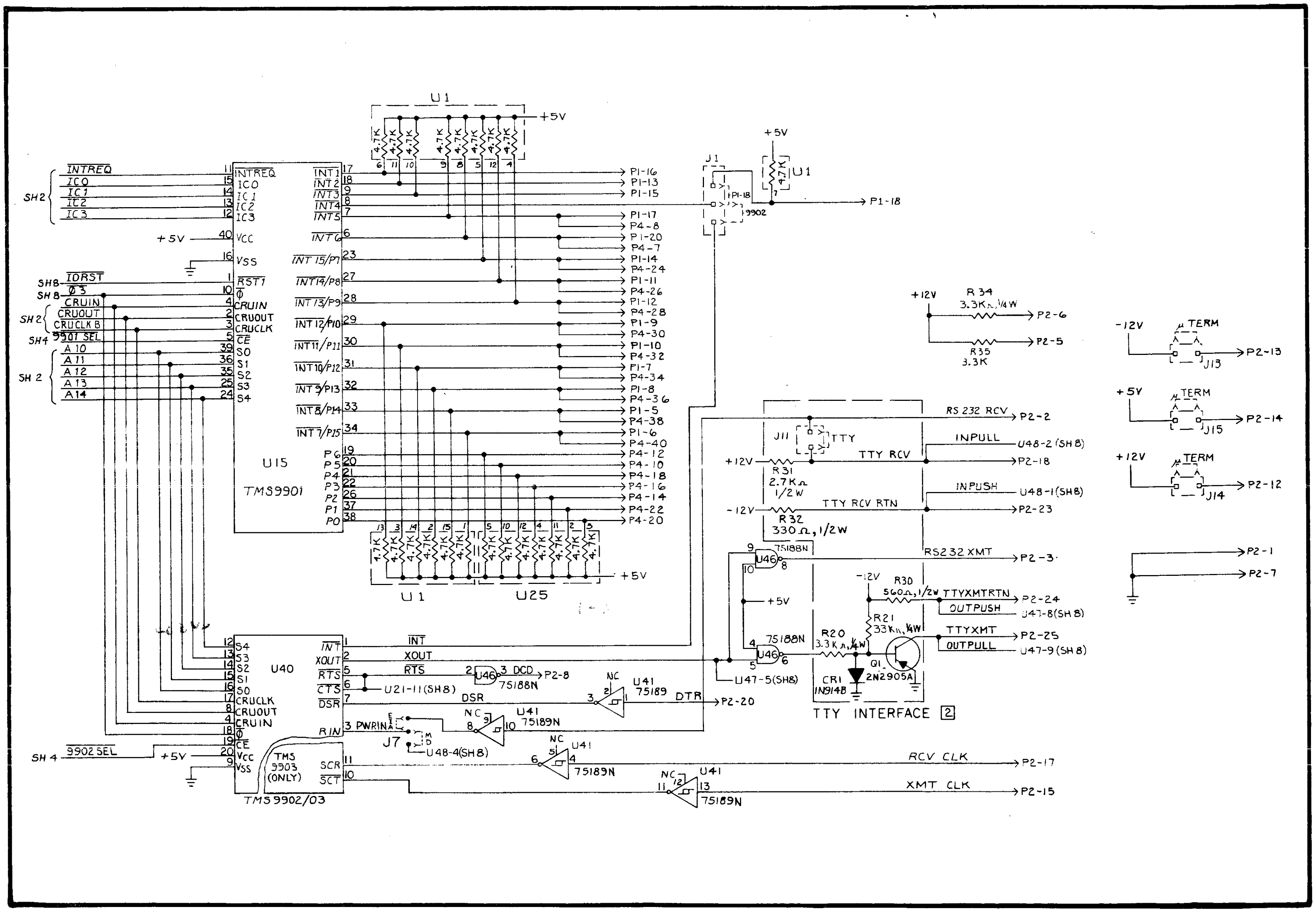


LOGIC DIAGRAM, TM 990/100M
SHEET 1 of 8



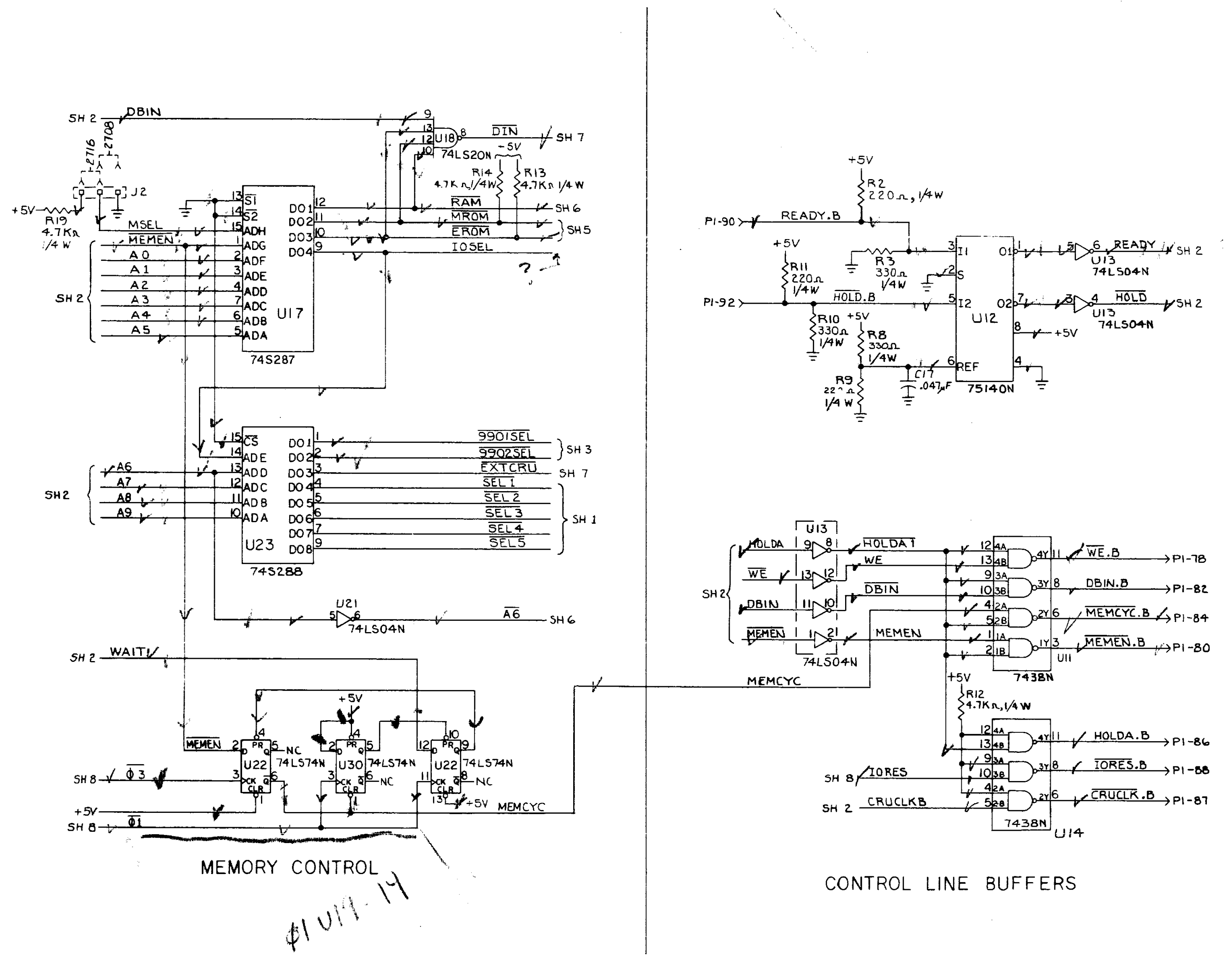
CENTRAL PROCESSOR UNIT

LOGIC DIAGRAM TM 990/100M
SHEET 2 of 8



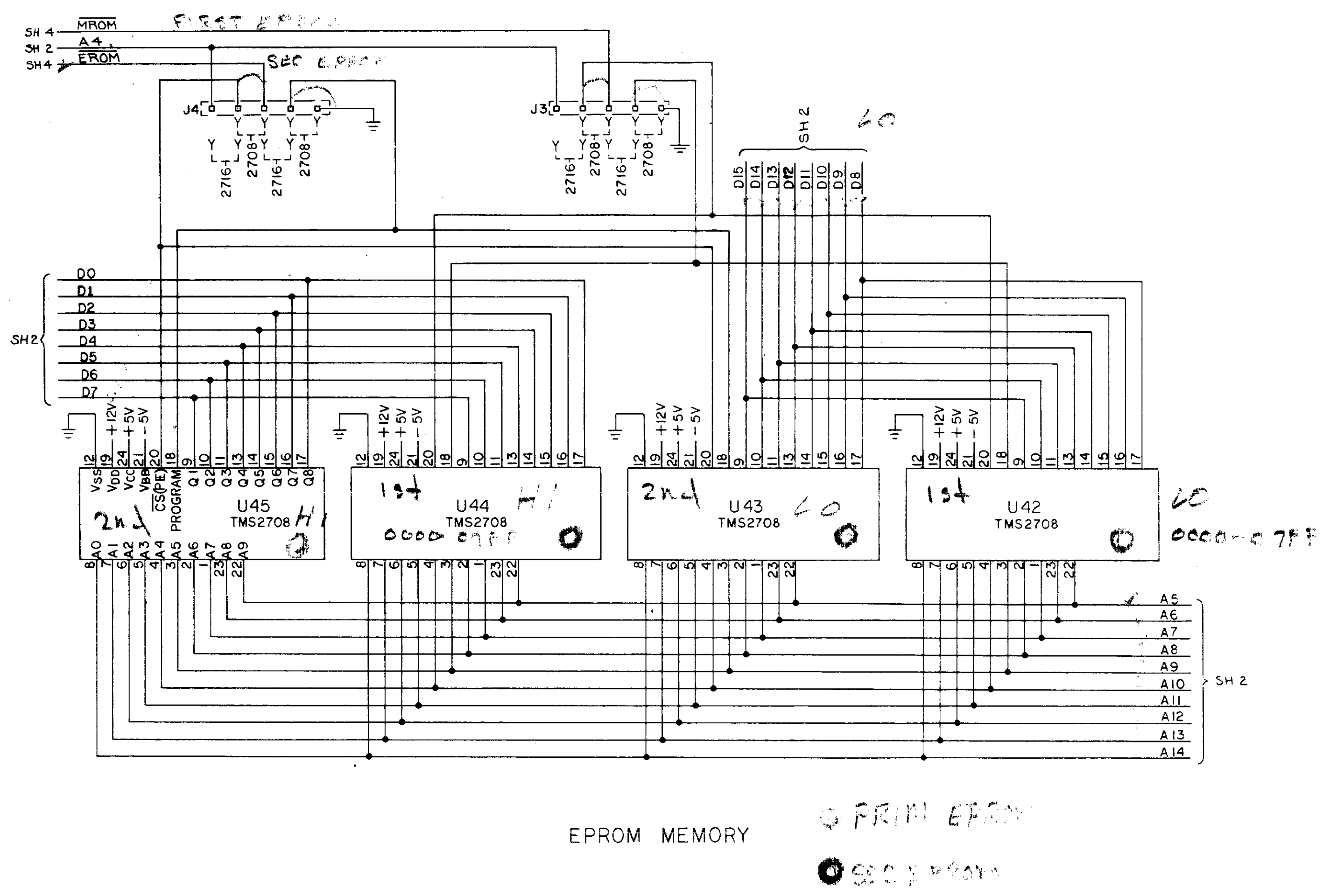
F-3

LOGIC DIAGRAM TM 990/100M
SHEET 3 of 8

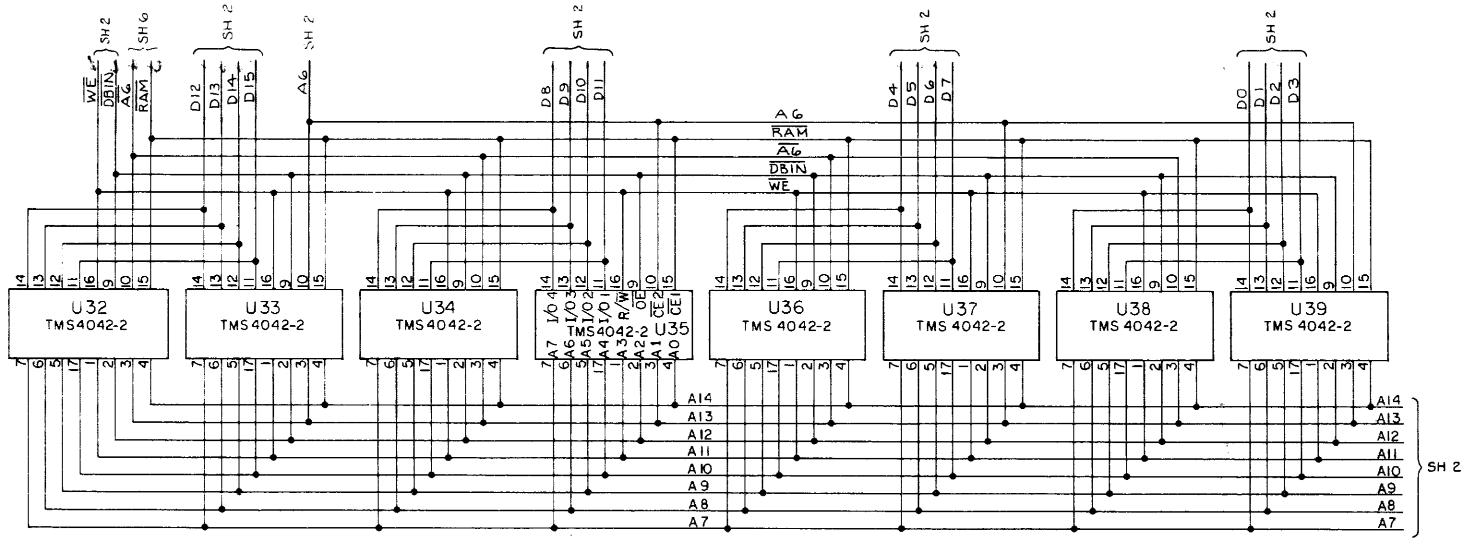


LOGIC DIAGRAM, TM 990/100M

SHEET 4 of 8

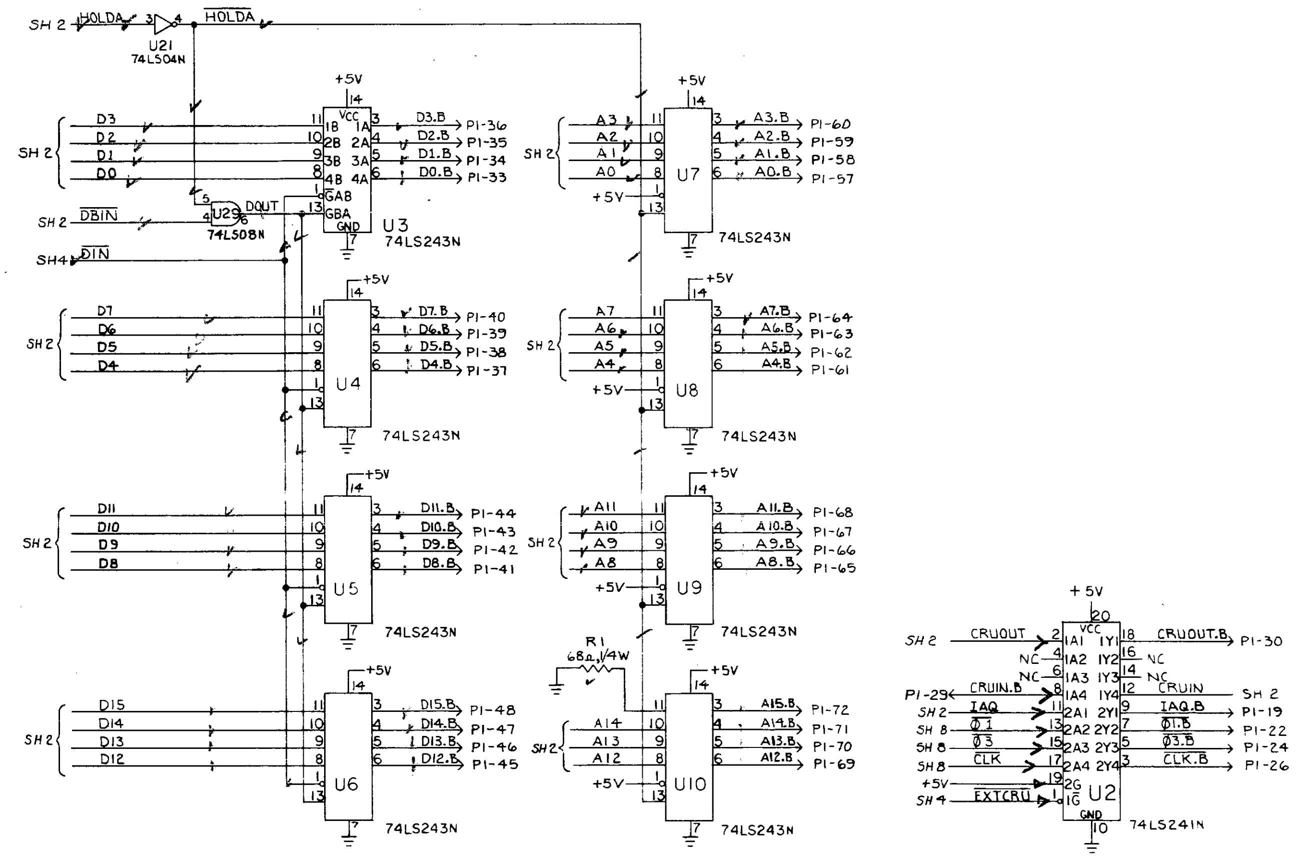


LOGIC DIAGRAM, TM 990/100M
SHEET 5 of 8



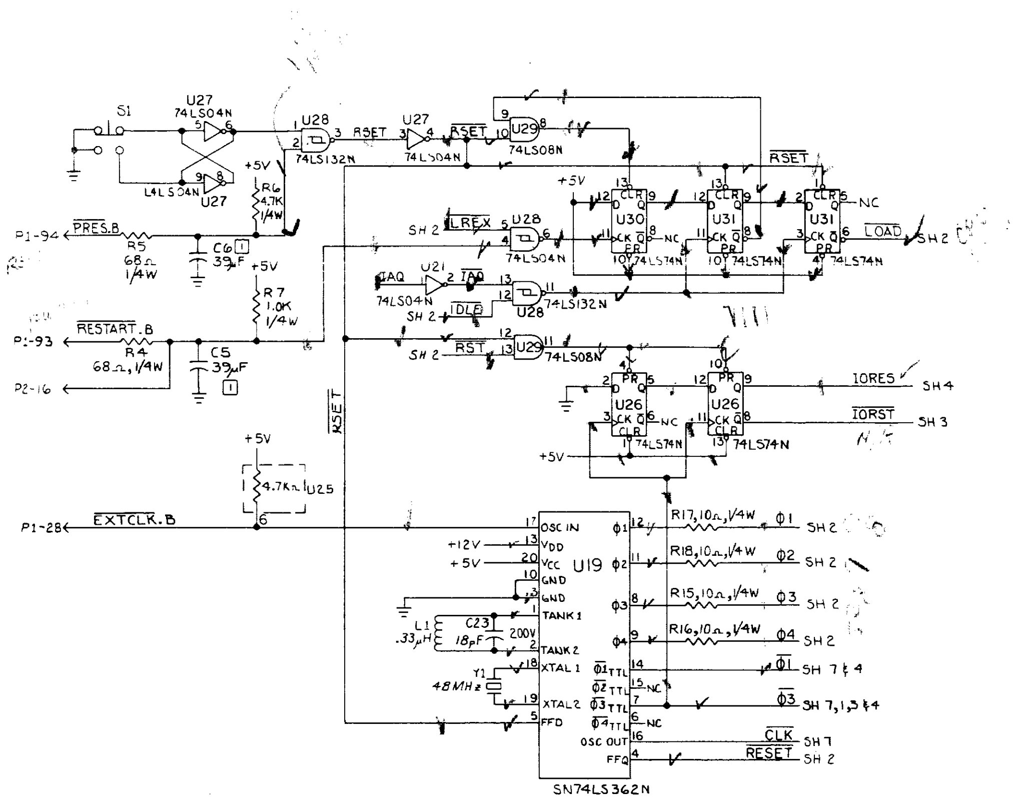
SH 2
SH 6
WE
DBIN
A6
RAM
D12
D13
D14
D15
D8
D9
D10
D11
D4
D5
D6
D7
D0
D1
D2
D3
U32
TMS4042-2
U33
TMS4042-2
U34
TMS4042-2
U35
TMS4042-2
U36
TMS 4042-2
U37
TMS 4042-2
U38
TMS 4042-2
U39
TMS 4042-2
I/O4
I/O3
I/O2
I/O1
R/W
OE
CE2
CE
A7
A6
A5
A4
A3
A2
A1
A0
A14
A13
A12
A11
A10
A9
A8
A7

RAM MEMORY

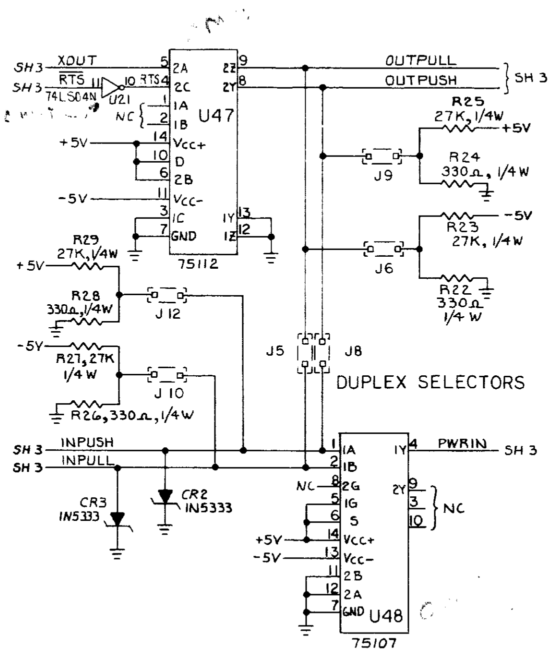


DATA/ADDRESS BUS BUFFERS

LOGIC DIAGRAM, TM 990/100M
SHEET 7 of 8



RESET/LOAD/CLOCK



MULTI-DROP INTERFACE 2

LOGIC DIAGRAM, TM 990/100M
SHEET 8 of 8

APPENDIX G

990 OBJECT CODE FORMAT

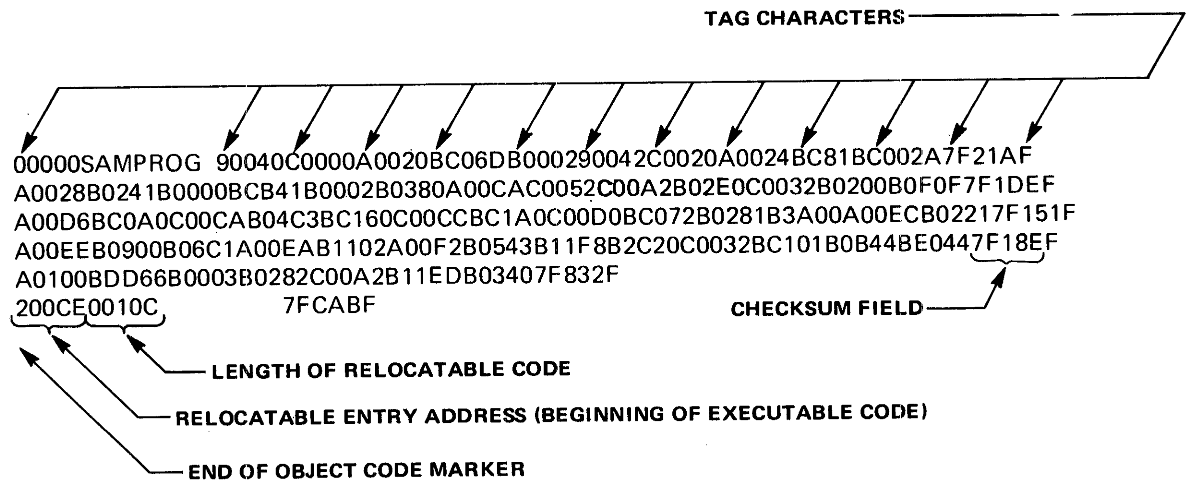
G.1 GENERAL

In order to correctly load a program into memory using a loader, the program in hexadecimal machine code must be in a particular format called object format. Such a format is required by the *TIBUG* loader (paragraph 3.2.7 explains loader execution). This object format has a tag character for each 16-bit word of coding which flags the loader to perform one of several operations. These operations include:

- Load the code at a user-specified absolute address and resolve relative addresses. (Most assemblers assemble a program as if it was loaded at memory address 0000_{16}; thus, relative addresses have to be resolved.)
- Load entire program at a specific address.
- Set the program counter to the entry address after loading.
- Check for checksum errors that would indicate a data error in an object record.

G.2 STANDARD 990 OBJECT CODE

Standard 990 object code consists of a string of hexadecimal digits, each representing four bits, as shown in Figure G-1.



```
00000SAMPROG  90040C0000A0020BC06DB000290042C0020A0024BC81BC002A7F21AF
A0028B0241B0000BCB41B0002B0380A00CAC0052C00A2B02E0C0032B0200B0F0F7F1DEF
A00D6BC0A0C00CAB04C3BC160C00CCBC1A0C00D0BC072B0281B3A00A00ECB02217F151F
A00EEB0900B06C1A00EAB1102A00F2B0543B11F8B2C20C0032BC101B0B44BE0447F18EF
A0100BDD66B0003B0282C00A2B11EDB03407F832F
200CE0010C           7FCABF
```

TAG CHARACTERS
CHECKSUM FIELD
LENGTH OF RELOCATABLE CODE
RELOCATABLE ENTRY ADDRESS (BEGINNING OF EXECUTABLE CODE)
END OF OBJECT CODE MARKER

A0001462

FIGURE G-1. OBJECT CODE EXAMPLE

The object record consists of a number of tag characters, each followed by one or two fields as defined in Table G-1. The first character of a record is the first tag character, which tells the loader which field or pair of fields follows the tag. The next tag character follows the end of the field or pair of fields associated with the preceding tag character. When the assembler has no more data for the record, the assembler writes the tag character 7 followed by the checksum field, and the tag character F, which requires no fields. The assembler then fills the rest of the record with blanks, and begins a new record with the appropriate tag character.

Tag character 0 is followed by two fields. The first field contains the number of bytes of relocatable code, and the second field contains the program identifier assigned to the program by an IDT assembler directive. When no IDT directive is entered, the field contains blanks. The loader uses the program identifier to identify the program, and the number of bytes of relocatable code to determine the load bias for the next module or program. The PX9ASM assembler is unable to determine the value for the first field until the entire module has been assembled, so PX9ASM places a tag character 0 followed by a zero field and the program identifier at the beginning of the object code file. At the end of the file, PX9ASM places another tag character zero followed by the number of bytes of relocatable code and eight blanks.

Tag characters 1 and 2 are used with entry addresses. Tag character 1 is used when the entry address is absolute. Tag character 2 is used when the entry address is relocatable. The hexadecimal field contains the entry address. One of these tags may appear at the end of the object code file. The associated field is used by the loader to determine the entry point at which execution starts when the loading is complete.

Tag characters 3 and 4 are used for external references. Tag character 3 is used when the last appearance of the symbol in the second field is in relocatable code. Tag character 4 is used when the last appearance of the symbol is absolute code. The hexadecimal field contains the location of the last appearance. The symbol in the second field is the external reference. Both fields are used by the linking loader to provide the desired linking to the external reference.

For each external reference in a program, there is a tag character in the object code, with a location, or an absolute zero, and the symbol that is referenced. When the object code field contains absolute zero, no location in the program requires the address that corresponds to the reference (an IDT character string, for example). Otherwise, the address corresponding to the reference will be placed in the location specified in the object code by the linking loader. The location specified in the object code similarly contains absolute zero or another location. When it contains absolute zero, no further linking is required. When it contains a location, the address corresponding to the reference will be placed in that address by the linking loader. The location of each appearance of a reference in a program contains either an absolute zero or another location into which the linking loader will place the referenced address.

TABLE G-1. OBJECT OUTPUT TAGS SUPPLIED BY ASSEMBLERS

TAG CHARACTER	HEXADECIMAL FIELD (FOUR CHARACTERS)	SECOND FIELD	MEANING
0	Length of all relocatable code	8-character program identifier	Program start
1	Entry address	None	Absolute entry address
2	Entry address	None	Relocatable entry address
3	Location of last appearance of symbol	6-character symbol	External reference last used in relocatable code
4	Location of last appearance of symbol	6-character symbol	External reference last used in absolute code
5	Location	6-character symbol	Relocatable external definition
6	Location	6-character symbol	Absolute external definition
7	Checksum for current record	None	Checksum
8	Ignore checksum	None	Do not checksum for error
9	Load address	None	Absolute load address
A	Load address	None	Relocatable load address
B	Data	None	Absolute data
C	Data	None	Relocatable data
D	Load bias value*	None	Load point specifier
F	None	None	End-of-record
G	Location	6-character symbol	Relocatable symbol definition
H	Location	6-character symbol	Absolute symbol definition

*Not supplied by assembler.

Tag characters 5 and 6 are used for external definitions. Tag character 5 is used when the location is relocatable. Tag character 6 is used when the location is absolute. Both fields are used by the linking loader to provide the desired linking to the external definition. The sec ›nd field contains the symbol of the external definition.

Tag character 7 precedes the checksum, which is an error detection word. The checksur is formed as the record is being written. It is the 2's complement of the sum of the 8-bit ASCII values of the characters of the record from the first tag of the record through the checksum tag 7. If the tag character 7 is replaced by an 8, the checksum will be ignored. The 8 tag can be used when object code is changed in editing and it is desired to ignore checksum.

Tag characters 9 and A are used with load addresses for data that follows. Tag character 9 is used when the load address is absolute. Tag character A is used when the load address is relocatable. The hexadecimal field contains the address at which the following data word is to be loaded. A load address is required for a data word that is to be placed in memory at some address other than the next address. The load address is used by the loader.

Tag characters B and C are used with data words. Tag character B is used when the data is absolute; an instruction word or a word that contains text characters or absolute constants, for example. Tag character C is used for a word that contains a relocatable address. The hexadecimal field contains the data word. The loader places the word in the memory location specified in the preceding load address field, or in the memory location that follows the preceding data word.

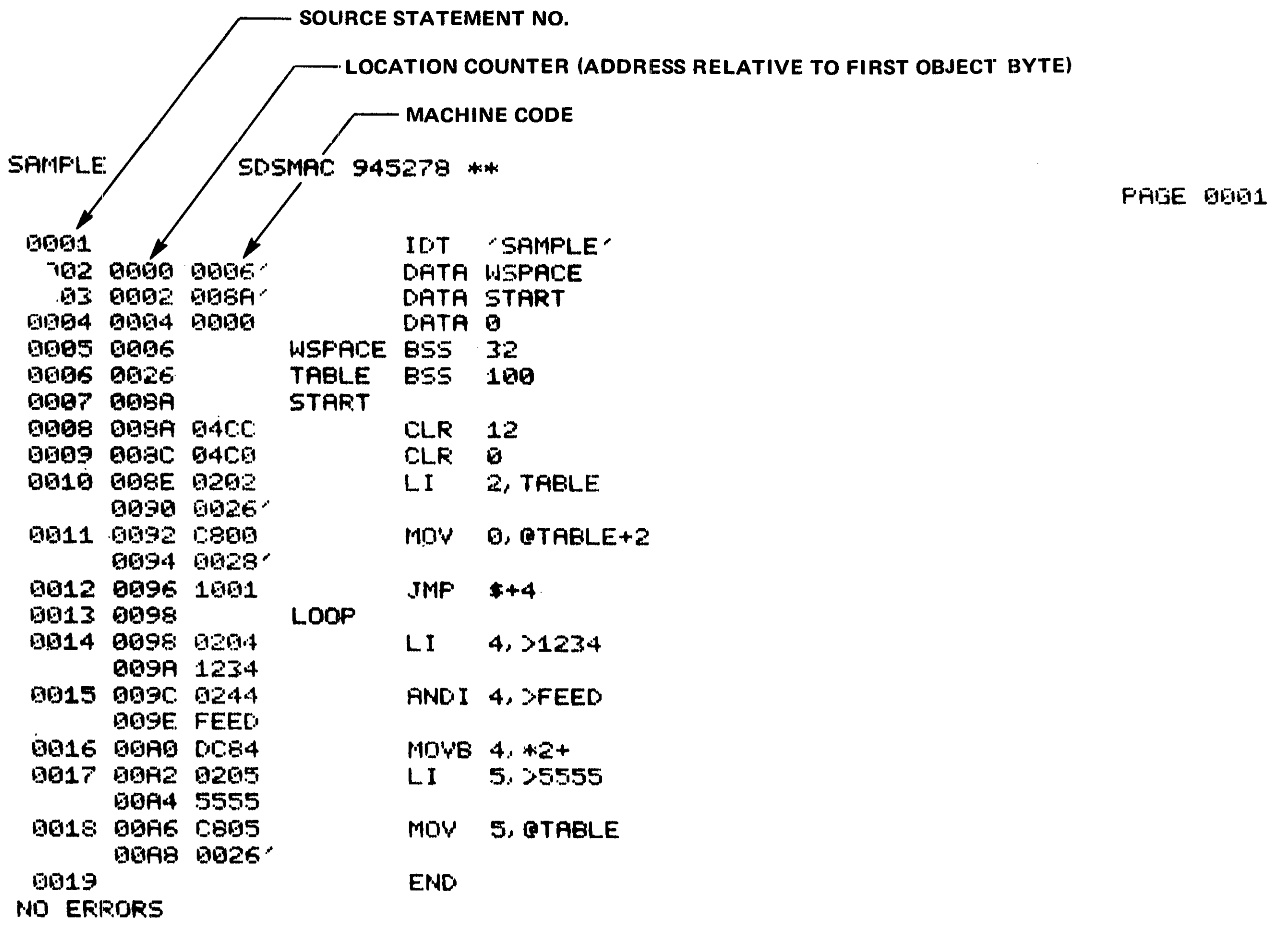
To have object code loaded at a specific memory address, precede the object program with the D tag followed by the desired memory address (e.g., DFD00).

Tag character F indicates the end of record. It may be followed by blanks.

Tag characters G and H are used when the symbol table option is specified with other 990 assemblers. Tag character G is used when the location or value of the symbol is relocatable, and tag character H is used when the location or value of the symbol is absolute. The first field contains the location or value of the symbol, and the second field contains the symbol to which the location is assigned.

The last record of an object code file has a colon (:) in the first character position of the record, followed by blanks. This record is referred to as an end-of-module separator record.

Figure G-2 is an example of an assembler source listing and corresponding object code. A comparison of the object tag characters and fields with the machine code in the source listing will show how object code is constructed for use by the loader.



```
SAMPLE          SDSMAC 945278 **
                                                                   PAGE 0001

0001                          IDT   'SAMPLE'
0002 0000 0006'               DATA WSPACE
0003 0002 008A'               DATA START
0004 0004 0000                DATA 0
0005 0006          WSPACE BSS   32
0006 0026          TABLE  BSS   100
0007 008A          START
0008 008A 04CC                CLR   12
0009 008C 04C0                CLR   0
0010 008E 0202                LI    2, TABLE
     0090 0026'
0011 0092 C800                MOV   0, @TABLE+2
     0094 0028'
0012 0096 1001                JMP   $+4
0013 0098          LOOP
0014 0098 0204                LI    4, >1234
     009A 1234
0015 009C 0244                ANDI  4, >FEED
     009E FEED
0016 00A0 DC84                MOVB  4, *2+
0017 00A2 0205                LI    5, >5555
     00A4 5555
0018 00A6 C805                MOV   5, @TABLE
     00A8 0026'
0019                          END
NO ERRORS
```

```
000AASAMPLE   A0000C0006C008AB0000A008AB04CCB04C0B0202C0026BC8007F200F         000
C0028B1001B0204B1234B0244BFEEDBDC84B0205B5555BC805C00267F3C1F                  000
:       SAMPLE     00/00/00   08:14:23                    SDSMAC 945278 **
```

FIGURE G-2. SOURCE CODE AND CORRESPONDING OBJECT CODE

APPENDIX H

P1, P2, AND P4 PIN ASSIGNMENTS

TABLE H-1. CHASSIS INTERFACE CONNECTOR (P1) SIGNAL ASSIGNMENTS

P1 PIN	SIGNAL	P1 PIN	SIGNAL	P1 PIN	SIGNAL
33	D0.B	71	A14.B	12	$\overline{\text{INT13}}$.B
34	D1.B	72	A15.B	11	$\overline{\text{INT14}}$.B
35	D2.B	22	$\overline{\text{Ø1}}$.B	14	$\overline{\text{INT15}}$.B
36	D3.B	24	$\overline{\text{Ø3}}$.B	28	EXTCLK.B
37	D4.B	92	$\overline{\text{HOLD}}$.B	3 ok	+5V
38	D5.B	86	HOLDA.B	4 ok	+5V
39	D6.B	82	DBIN.B	97 ok	+5V
40	D7.B	26	$\overline{\text{CLK}}$.B	98 ok	+5V
41	D8.B	80	$\overline{\text{MEMEN}}$.B	75 B	+12V
42	D9.B	84	$\overline{\text{MEMCYC}}$.B	76 T	+12V
43	D10.B	78	$\overline{\text{WE}}$.B	73 B	−12V
44	D11.B	90	READY.B	74 T	−12V
45	D12.B	87	$\overline{\text{CRUCLK}}$.B	1 ok	GND >
46	D13.B	30	CRUOUT.B	2 ok	GND
47	D14.B	29	CRUIN.B	21	GND
48	D15.B	19	IAQ.B	23	GND
57	A0.B	94	$\overline{\text{PRES}}$.B	25	GND
58	A1.B	88	$\overline{\text{IORST}}$.B	27	GND
59	A2.B	16	$\overline{\text{INT1}}$.B	31	GND
60	A3.B	13	$\overline{\text{INT2}}$.B	77 ok	GND
61	A4.B	15	$\overline{\text{INT3}}$.B	79	GND
62	A5.B	18	$\overline{\text{INT4}}$.B	81	GND
63	A6.B	17	$\overline{\text{INT5}}$.B	83	GND
64	A7.B	20	$\overline{\text{INT6}}$.B	85	GND
65	A8.B	6	$\overline{\text{INT7}}$.B	89	GND
66	A9.B	5	$\overline{\text{INT8}}$.B	91 ok	GND
67	A10.B	8	$\overline{\text{INT9}}$.B	99 ok	GND
68	A11.B	7	$\overline{\text{INT10}}$.B	100 ok	GND
69	A12.B	10	$\overline{\text{INT11}}$.B	93	$\overline{\text{RESTART}}$.B
70	A13.B	9	$\overline{\text{INT12}}$.B		

TABLE H-2. SERIAL I/O INTERFACE (P2) PIN ASSIGNMENTS

P2 PIN	SIGNAL	DESCRIPTION
1	GND	
7	GND	
3	RS232 XMT	RS232 Serial Data Out
2	RS232 RCV	RS232 Serial Data In
5	CTS	Clear to Send (3.3KΩ pull-up to +12 V)
6	DSR	Data Set Ready (3.3KΩ pull-up to +12 V)
8	DCD	Carrier Detect
20	DTR	Data Terminal Ready
18,23	TTY XMT	TTY Receive Loop/Private Wire Receive Pair
24,25	TTY RCV	TTY Transmit Loop/Private Wire Transmit Pair
17	RCV CLK	Receive Clock
15	XMT CLK	Transmit Clock
12*	+12 V	Jumper Option for Microterminal
13*	−12 V	Jumper Option for Microterminal
14*	+5 V	Jumper Option for Microterminal
16	RESTART	Invokes the Load Interrupt to the TMS 9900 CPU

*When using the Microterminal, these voltages are jumpered to the corresponding pin in connector P2. Else, the voltages are not connected.

TABLE H-3. PARALLEL I/O INTERFACE (P4) SIGNAL ASSIGNMENT

P4 PIN	SIGNAL	P4 PIN	SIGNAL
20	P0	17	GND
22	P1	15	GND
14	P2	13	GND
16	P3	11	GND
18	P4	9	GND
10	P5	39	GND
12	P6	37	GND
24	$\overline{\text{INT15}}$ or P7	35	GND
26	$\overline{\text{INT14}}$ or P8	33	GND
28	$\overline{\text{INT13}}$ or P9	31	GND
30	$\overline{\text{INT12}}$ or P10	29	GND
32	$\overline{\text{INT11}}$ or P11	27	GND
34	$\overline{\text{INT10}}$ or P12	25	GND
36	$\overline{\text{INT9}}$ or P13	23	GND
38	$\overline{\text{INT8}}$ or P14	21	GND
40	$\overline{\text{INT7}}$ or P15	19	GND
7	$\overline{\text{INT 6}}$	1–6	Spares
8	$\overline{\text{INT 5}}$		

APPENDIX I

TM 990/301 MICROTERMINAL

I.1 GENERAL

The Texas Instruments Microterminal offers all of the features of a minicomputer front panel at reduced cost. The Microterminal, intended primarily to support the Texas Instruments TM 990/100M and TM 990/180M microcomputers, allows the user to do the following:

- Read from ROM or read/write to RAM
- Enter/display Program Counter
- Execute user program in free running mode or in single instruction mode
- Halt user program execution
- Enter/display Status Register
- Enter/display Workspace Pointer (this term is unique to the Texas Instruments 9900 microprocessor)
- Enter/display CRU data (this term is unique to the Texas Instruments 9900 microprocessor)
- Convert hexadecimal quantity to signed decimal quantity
- Convert signed decimal quantity to hexadecimal quantity

I.2 SPECIFICATIONS

- Power Requirements
  +12V (± 3%), 50 mA
  −12V (± 3%), 50 mA
  + 5V (± 3%),150 mA
- Operating Temperature: 0°C to 50°C (+32° to +122°F)
- Operating Humidity: 0 to 95 percent, non-condensing
- Shock: Withstand 2 foot vertical drop

I.3 INSTALLATION

To install the Microterminal onto a TM 990/100M or TM 990/180M microcomputer, do the following:

- Attach jumpers to J13, J14, and J15 on the TM 990/100M or to J4, J5, and J6, on the TM 990/180M board to route voltages to the Microterminal.
- Attach the EIA cable from the Microterminal to connector P2. Signals between the Microterminal and the microcomputer are listed as in Table I-1.

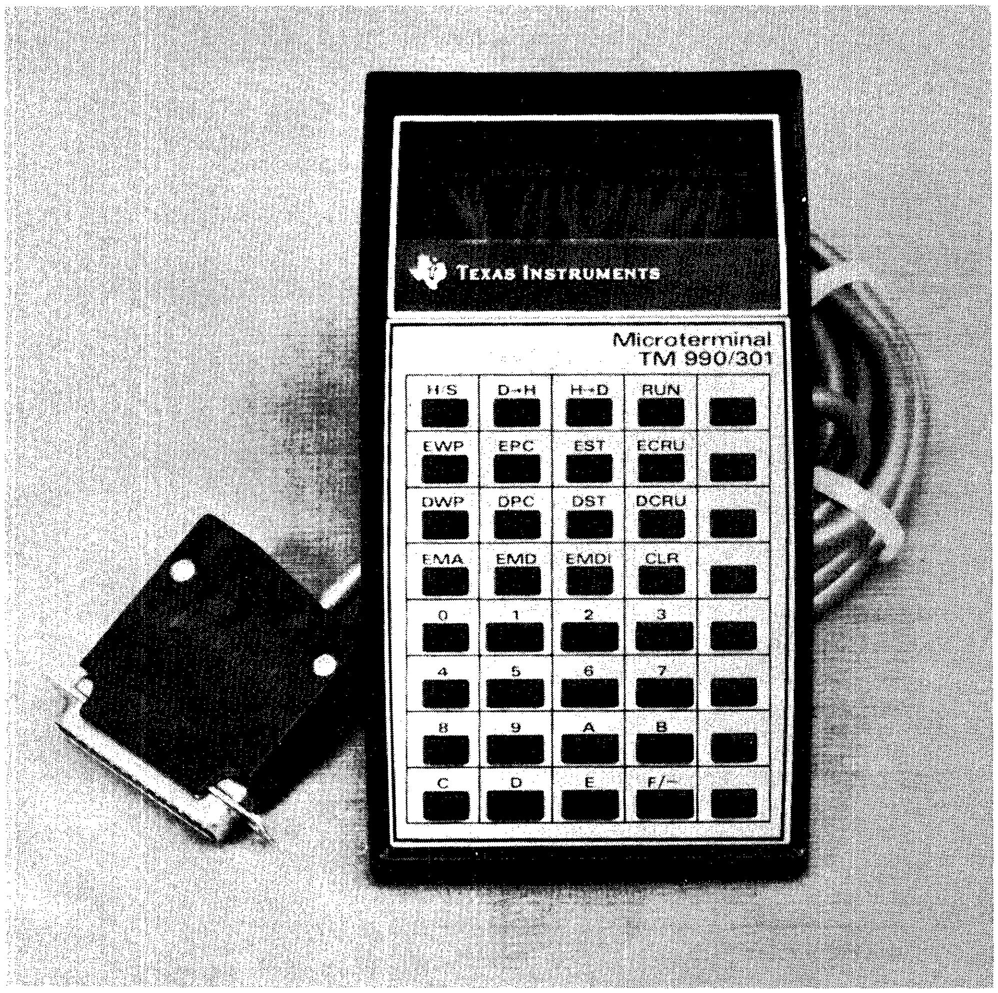


FIGURE I-1. TM 990/301 MICROTERMINAL

TABLE I-1. EIA CABLE SIGNALS

EIA Connector Pin	Interface Signal	At TM 990/100M/180M	
		P2 Pin	Signal
2	$\overline{\text{TERMINAL DATA OUT}}$	−2	RS232 RCV
3	$\overline{\text{TERMINAL DATA IN}}$	−3	RS232 XMT
7	GND	−7	GND
12	+12V	−12	+12V
13	−12V	−13	−12V
14	+ 5V	−14	+ 5V
16	$\overline{\text{HALT}}$	−16	$\overline{\text{RESTART}}$

CAUTION

Before attaching the Microterminal to a power source, verify voltage levels between ground and EIA connector pins 12, 13, and 14 at connector P2 on the board. Voltage should not exceed values in Table I-1.

I.4 KEY DEFINITIONS

I.4.1 DATA KEYS

CLR — Clear Key – Depressing this key blanks display, initializes and sends initialization message (ASCII code for A and ASCII code for Z) to host microcomputer.

0
1
.
.
.
F/– — Hexadecimal Data Keys – Depressing any one of these keys shifts that value into the right-hand display digit. All digits already in the data display are left shifted. For all operations other than decimal to hexadecimal conversion, the fourth digit from the right is shifted off the end of the right-hand display field when a data key is depressed. For a decimal to hexadecimal conversion, the fifth display digit from the right, rather than the fourth, is shifted off the end of the data field.

I.4.2 INSTRUCTION EXECUTION

H/S — Pressing this key while a program is running (run displayed) will halt program execution. The address of the next instruction will be displayed in the four left-hand display digits, and the contents of that address will be displayed in the four right-hand digits. Pressing this key while the program is halted, will execute a single instruction using the values in the Workspace Pointer (WP), Program Counter (PC), and Status Register (ST), and the displays will be updated to the next memory address and contents at that address.

RUN — Pressing this key initiates program execution at the current values in the WP, PC; run is displayed in the three right-hand display digits.

I.4.3 ARITHMETIC

H→D — The signed hexadecimal data contained in the four right-hand display digits is converted to signed decimal data. Note that the fourth display digit from the right is the sign bit (1 = negative). The conversion limits are minus $32{,}768_{10}$ (8000_{16}) to plus 32,767 ($7FFF_{16}$). Two H→D key depressions are required. The sequence is:

1. Depress H→D .
2. Enter data via hex data key depressions.
3. Depress H→D . The results of the conversion are displayed in the five right-hand display digits.

D→H — The decimal data contained in the five right-hand display digits is converted to hexadecimal. The conversion limits are the same as for hexadecimal to decimal conversion. The sequence is:

1. Depress D→H .
2. Enter data via hex data key depressions.
3. Depress D→H . The results of the conversion are displayed in the four right-hand display digits.

I.4.4 REGISTER ENTER/DISPLAY

$\boxed{\text{EWP}}$ Pressing this key causes the value displayed in the four right-hand digits to be entered into the WP.

$\boxed{\text{DWP}}$ Pressing this key causes the WP contents to be displayed in the four right-hand display digits.

$\boxed{\text{EPC}}$ Pressing this key causes the value displayed in the four right-hand digits to be entered into the PC.

$\boxed{\text{DPC}}$ Pressing this key causes the PC contents to be displayed in the four right-hand display digits.

$\boxed{\text{EST}}$ Pressing this key causes the value displayed in the four right-hand digits to be entered into the ST.

$\boxed{\text{DST}}$ Pressing this key causes the ST contents to be displayed in the four right-hand display digits.

I.4.5 CRU DISPLAY/ENTER

$\boxed{\text{DCRU}}$ Pressing this key causes the data at the designated Communications Register Unit (CRU) addresses to be displayed. Designate from one to 16 CRU bits at a specified CRU address by using four hexadecimal digits. The first digit is the count of bits to be displayed. The next three digits are the CRU address (equal to bits 3 to 14 in register 12 for CRU addressing). When $\boxed{\text{DCRU}}$ is depressed, the bit count and address are shifted to the left-hand display, and the right-hand display will contain the values at the selected CRU output addresses. The output value will be zero-filled on the left, depending upon bit count entered. If less than nine bits, the value will be contained in the left two hexadecimal digits. If nine or more, the value will be right justified in all four hexadecimal digits.

$\boxed{\text{ECRU}}$ Pressing this key enters a new value at the CRU addresses and bit count shown in the left display after depressing $\boxed{\text{DCRU}}$. The new value is entered from the keyboard and displayed in the right-hand display. Pressing $\boxed{\text{ECRU}}$ enters this value onto the CRU at the address shown in the left display.

CAUTION

Avoid setting new values at the TMS 9902 on the TM 990/100M/180M through the CRU (TMS 9902 is at CRU address 0040_{16}), as this device controls I/O functions.

I.4.6 MEMORY ENTER, DISPLAY, INCREMENT

$\boxed{\text{EMA}}$ Pressing this key will cause (1) the memory address (MA) in the right-hand display to be shifted to the left-hand display and (2) the contents of that memory address to be displayed in the right-hand display.

$\boxed{\text{EMD}}$ Pressing this key causes the value in the right-hand display to be entered into the memory address contained in the left-hand display. The contents of that location will then be displayed in the four right-hand display digits (entered then read back).

$\boxed{\text{EMDI}}$ Pressing this key causes the same action as described for the $\boxed{\text{EMD}}$ key; it also increments the memory address by two and displays the contents at that new address. The memory address is displayed on the left and the contents at that address is displayed on the right.

I.5 EXAMPLES

I.5.1 EXAMPLE 1, ENTER PROGRAM INTO MEMORY

Enter the following program starting at RAM location $FE00_{16}$. Set the workspace pointer to $FF00_{16}$ and the status register to 2000_{16}. Single step through the program and verify execution. Then execute the program in free run mode and verify execution. Then halt program execution.

NOTE
In the following examples, XXXX indicates memory contents at current value in Memory Address Register.

```
OPCODE    INSTRUCTIONS

 04C0     CLR     R0          CLEAR WORKSPACE REGISTER 0
 0580     INC     R0          INCREMENT WORKSPACE REGISTER 0
 0280     CI      R0, >00FF   CHECK FOR COUNT 255
 00FF
 16FC     JNE     $-6         JUMP TO INC R0 IF NOT DONE
 10FF     JMP     $-0         STAY HERE WHEN FINISHED
```

	KEY ENTRIES		DISPLAY
Clear Display	Depress	CLR	
Enter PC Value	Depress	F/– E 0 0	FE00
Enter into PC	Depress	EPC	FE00
Display PC	Depress	DPC	FE00
Enter ST Value	Depress	2 0 0 0	2000
Enter into ST	Depress	EST	2000
Display ST	Depress	DST	2000
Enter WP Value	Depress	F/– F/– 0 0	FF00
Enter Into WP	Depress	EWP	FF00
Display WP	Depress	DWP	FF00
Enter MA Value	Depress	F/– E 0 0	FE00
Enter Into MA	Depress	EMA	FE00 xxxx
Enter CLR 0 Opcode	Depress	0 4 C 0	FE00 04C0
Enter data, increment MA	Depress	EMDI	FE02 xxxx
Enter INC 0 Opcode	Depress	0 5 8 0	FE02 0580
Enter Data, Increment MA	Depress	EMDI	FE04 xxxx
Enter CI Opcode	Depress	0 2 8 0	FE04 0280
Enter Data, Increment MA	Depress	EMDI	FE06 xxxx

		KEY ENTRIES	DISPLAY
Enter CI Immediate Operand	Depress	0 0 F F	FE06 00FF
Enter Data, Increment MA	Depress	EMDI	FE08 xxxx
Enter JNE $-6 Opcode	Depress	1 6 F C	FE08 16FC
Enter Data, Increment MA	Depress	EMDI	FE0A xxxx
Enter JMP $-0 Opcode	Depress	1 0 F F	FE0A 10FF
Enter Data, Increment MA	Depress	EMDI	FE0C xxxx

The program has now been entered into RAM. Since the PC, ST and WP values have been previously set, the program can be executed in single step mode by depressing the H/S key.

		DISPLAY (AFTER)	EXECUTES INSTRUCTION
Depress	H/S	FE02 0580	CLR RO
Depress	H/S	FE04 0280	INC RO
Depress	H/S	FE08 16FC	CI RO,>00FF
Depress	H/S	FE02 0580	JNE $–6

This cycle will continue until R0 reaches the count of 255 at which point the program will continuously execute at location FE0A$_{16}$ because it is a jump to itself.

To verify this, depress:	DISPLAY
RUN	run

The program should now be "looping to self" at location FE0A$_{16}$. To verify this, depress:

H/S	FE0A 10FF

Now examine the memory location corresponding to Register 0.

Depress	F F 0 0	FE0A FF00
Depress	EMA	FF00 00FF

This illustrates that FF$_{16}$ did become the final contents of WP0. Note that, when the program was being entered into RAM, $\boxed{\text{EMDI}}$ was used rather than $\boxed{\text{EMD}}$ because of the rather desirable feature of automatic address incrementing. The advantage of using $\boxed{\text{EMD}}$ is that the actual contents of the addressed memory location are displayed after key depression (echoed back after being entered).

I.5.2 EXAMPLE 2, HEXADECIMAL TO DECIMAL CONVERSIONS

Convert 8000_{16} to a decimal number

Depress	$\boxed{\text{CLR}}$		
Depress	$\boxed{\text{H}\rightarrow\text{D}}$		
Depress	$\boxed{8}$ $\boxed{0}$ $\boxed{0}$ $\boxed{0}$		8000
Depress	$\boxed{\text{H}\rightarrow\text{D}}$	−3	2768

Convert 0020_{16} to a decimal number

Depress	$\boxed{\text{CLR}}$		
Depress	$\boxed{\text{H}\rightarrow\text{D}}$		
Depress	$\boxed{2}$ $\boxed{0}$		20
Depress	$\boxed{\text{H}\rightarrow\text{D}}$		32

I.5.3 EXAMPLE 3, DECIMAL TO HEXADECIMAL CONVERSIONS

Convert 45_{10} to hex

Depress	$\boxed{\text{CLR}}$		
Depress	$\boxed{\text{D}\rightarrow\text{H}}$		
Depress	$\boxed{4}$ $\boxed{5}$		45
Depress	$\boxed{\text{D}\rightarrow\text{H}}$		2D

Convert -1024_{10} to hex

Depress	$\boxed{\text{CLR}}$		
Depress	$\boxed{\text{D}\rightarrow\text{H}}$		
Depress	$\boxed{\text{F/}-}$ $\boxed{1}$ $\boxed{0}$ $\boxed{2}$ $\boxed{4}$	−	1024
Depress	$\boxed{\text{D}\rightarrow\text{H}}$		FC00

I.5.4 EXAMPLE 4, ENTER VALUE ON CRU

Send a bit pattern to the CRU at CRU address (bits 3 to 14 of R12) $0E0_{16}$ with a bit count of 9 containing a value of 5 (000000101_2).

Depress	$\boxed{\text{CLR}}$	$\boxed{}\boxed{}$
Depress	$\boxed{9}$ $\boxed{0}$ $\boxed{\text{E}}$ $\boxed{0}$	$\boxed{}\boxed{\text{90E0}}$
Depress	$\boxed{\text{DCRU}}$	$\boxed{\text{90E0}}\boxed{\text{YYYY}}$
Depress	$\boxed{0}$ $\boxed{0}$ $\boxed{0}$ $\boxed{5}$	$\boxed{\text{90E0}}\boxed{\text{0005}}$
Depress	$\boxed{\text{ECRU}}$	

YYYY indicates value at the current CRU address. Note that a $\boxed{\text{DCRU}}$ operation is always required to specify bit count/CRU address.

I.5.5 EXAMPLE 5. ENTER, VERIFY VALUE AT MEMORY ADDRESS

Enter 0040_{16} into location FE20 and verify that it got there.

Depress	$\boxed{\text{CLR}}$	
Depress	$\boxed{\text{F}}$ $\boxed{\text{E}}$ $\boxed{2}$ $\boxed{0}$	$\boxed{}\boxed{\text{FE20}}$
Depress	$\boxed{\text{EMA}}$	$\boxed{\text{FE20}}\boxed{\text{xxxx}}$
Depress	$\boxed{0}$ $\boxed{0}$ $\boxed{4}$ $\boxed{0}$	$\boxed{\text{FE20}}\boxed{\text{0040}}$
Depress	$\boxed{\text{EMD}}$	$\boxed{\text{FE20}}\boxed{\text{0040}}$

The contents of address FE20 are verified by an echo of data from memory to display following the pressing of $\boxed{\text{EMD}}$. If it is desired to view and enter data at address FE22, depress $\boxed{\text{EMD}}$.

ALPHABETICAL INDEX

INTRODUCTION

The following index lists key words and concepts from the subject material of the manual together with the area(s) in the manual that supply major coverage of the listed concept. The numbers along the right side of the listing reference the following manual areas:

- Sections – References to Sections of the manual appear as "Section x" with the symbol x representing any numeric quantity.
- Appendixes – References to Appendixes of the manual appear as Appendix y" with the symbol y representing any capital letter.
- Paragraphs – References to paragraphs of the manual appear as a series of alphanumeric or numeric characters punctuated with decimal points. Only the first character of the string may be a letter; all subsequent characters are numbers. The first character refers to the section or appendix of the manual in which the paragraph is found.
- Tables – References to tables in the manual are represented by the capital letter T followed immediately by another alphanumeric character (representing the section or appendix of the manual containing the table). The second character is followed by a dash (-) and a number:

  Tx-yy
- Figures – References to figures in the manual are represented by the capital letter F followed immediately by another alphanumeric character (representing the section or appendix of the manual containing the figure). The second character is followed by a dash (-) and a number:

  Fx-yy

INDEX

INDEX (Continued)

INDEX (Concluded)

TEXAS INSTRUMENTS
INCORPORATED
POST OFFICE BOX 1443 • HOUSTON, TEXAS 77001

January 1978

Dear Customer:

We are pleased that you have chosen a member of the TM990 product line for your microcomputer system.

We would like to introduce you to our user's organization at this time. As the owner of a TM990 microcomputer module you are eligible for membership to TI-MIX, the Texas Instruments Minicomputer Information Exchange. This is a group of mini and microcomputer users who are interested in exchanging ideas, programs, etc., with each other. We have enclosed an application to TI-MIX and hope that you will consider joining.

We are working to expand and improve our product line and would appreciate your help. Any suggestions or problems on the general product line should be addressed to:

Texas Instruments, Inc.
8600 Commerce Park Drive
Suite 700, MS 653
Houston, Texas 77036

On questions concerning the installation or warranty of the module, please contact your local TI field sales office or authorized distributor. For detailed technical questions please call our Microprocessor Customer Support Line at (713) 776-6511, Ext. 632.

Sincerely,

Al Lofthus

Al Lofthus
Microcomputer Program Manager
MOS Microprocessor

AL:bdr
Enclosure