73001BA-02

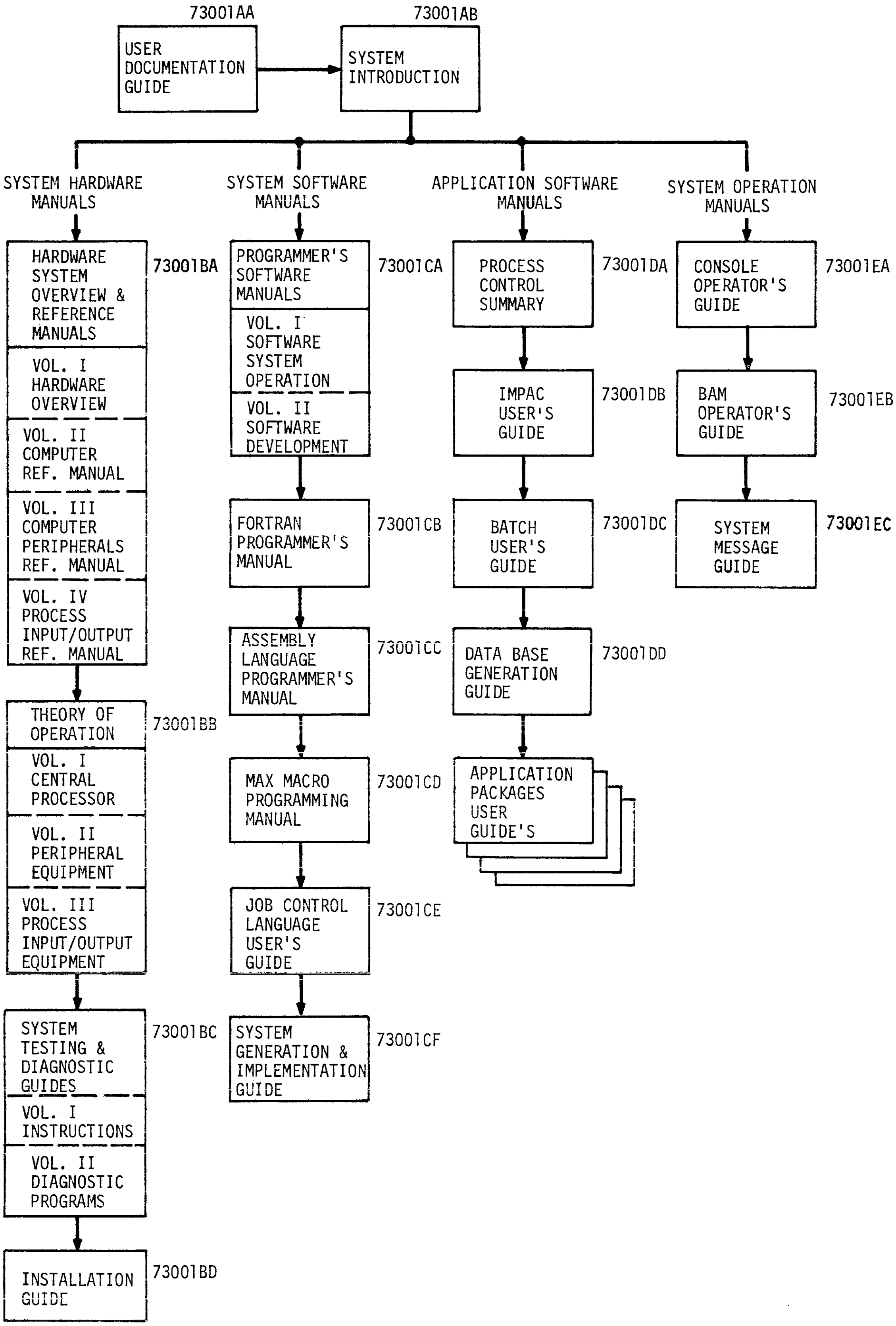
# FOX 1

# Hardware System Overview and Reference Manual

## VOLUME 2
## COMPUTER REFERENCE MANUAL

FOXBORO

# FOX 1 USER DOCUMENTATION


73001AA
USER DOCUMENTATION GUIDE
73001AB
SYSTEM INTRODUCTION
SYSTEM HARDWARE MANUALS
SYSTEM SOFTWARE MANUALS
APPLICATION SOFTWARE MANUALS
SYSTEM OPERATION MANUALS
HARDWARE SYSTEM OVERVIEW & REFERENCE MANUALS
73001BA
VOL. I HARDWARE OVERVIEW
VOL. II COMPUTER REF. MANUAL
VOL. III COMPUTER PERIPHERALS REF. MANUAL
VOL. IV PROCESS INPUT/OUTPUT REF. MANUAL
THEORY OF OPERATION
73001BB
VOL. I CENTRAL PROCESSOR
VOL. II PERIPHERAL EQUIPMENT
VOL. III PROCESS INPUT/OUTPUT EQUIPMENT
SYSTEM TESTING & DIAGNOSTIC GUIDES
73001BC
VOL. I INSTRUCTIONS
VOL. II DIAGNOSTIC PROGRAMS
INSTALLATION GUIDE
73001BD
PROGRAMMER'S SOFTWARE MANUALS
73001CA
VOL. I SOFTWARE SYSTEM OPERATION
VOL. II SOFTWARE DEVELOPMENT
FORTRAN PROGRAMMER'S MANUAL
73001CB
ASSEMBLY LANGUAGE PROGRAMMER'S MANUAL
73001CC
MAX MACRO PROGRAMMING MANUAL
73001CD
JOB CONTROL LANGUAGE USER'S GUIDE
73001CE
SYSTEM GENERATION & IMPLEMENTATION GUIDE
73001CF
PROCESS CONTROL SUMMARY
73001DA
IMPAC USER'S GUIDE
73001DB
BATCH USER'S GUIDE
73001DC
DATA BASE GENERATION GUIDE
73001DD
APPLICATION PACKAGES USER GUIDE'S
CONSOLE OPERATOR'S GUIDE
73001EA
BAM OPERATOR'S GUIDE
73001EB
SYSTEM MESSAGE GUIDE
73001EC

73001BA-02

# Hardware System Overview and Reference Manual

## VOLUME 2
## COMPUTER REFERENCE MANUAL

**Preliminary**

$17.50

The information contained in this document is general in nature and not intended for specific application purposes. The Foxboro Company reserves the right to make changes in specifications shown herein and to make product changes at any time without notice or obligation.

HISTORY

| Date | Issue | Copies |
|---|---|---|
| 3-72 | Original | 800 |

# Preface

This publication is a comprehensive description of the FOX 1 hardware system and the hardware system components. It can be used as an introductory textbook for hardware system analysts, engineers, and machine-language programmers learning the basic equipment structure and operation. It can also be used as a "quick look-up" reference manual for basic machine functions. Details of hardware logic and maintenance, or software that drives and utilizes the features of the hardware units are not in this book. Details of FOX 1 hardware configuration and operation at the machine-language level are presented here.

Volume 1 of this manual presents system overview information describing the FOX 1 hardware components, the system hardware configurations, and the participation of each hardware component in system functions. In general, sections on system hardware components are described in terms of a functional description, physical description, standard and optional features, interface considerations, and specifications.

Volumes 2, 3, and 4 present reference-level information describing the FOX 1 system hardware components (units). To simplify location of specific information, Volume 2 describes the computer components, Volume 3 describes the computer-peripheral type components, and Volume 4 describes the process-oriented type components. Volume 2 is divided into sections related to functions of the computer or to elements of the computer subsystem performing a specific task. Volumes 3 and 4 are divided into sections for each system hardware component. Individual sections describe each of these components in terms of a functional summary, block diagram discussion, instruction complement, and any other basic information peculiar to the subject component.

## DOCUMENTATION STRUCTURE

Readers should be aware of the relationship between this document and other books in the FOX 1 anthology, as described in the FOX 1 User Documentation Guide, 73001AA.

The FOX 1 Hardware System Overview and Reference Manuals, 73001BA, are divided into four volumes which may or may not be bound under separate covers. Each volume is subdivided into modular, topical sections (equivalent to chapters). Pages are assigned a two-part number in which the section number precedes and is separated from the page sequence number by a hyphen. All headings and paragraphs are assigned a two-part number in which the section number precedes and is separated from a sequence number by a decimal point. The span of these numbers is indicated at the top of each page, allowing subjects to be referenced (such as in the index) not only by page, but also by paragraph number.

Tables and illustrations bear a three-part designation consisting of the volume number, the section number, and the sequence number, all separated by hyphens.

Information contained herein is preliminary in nature, being printed in advance of exhaustive formal technical and corporate review; it is subject to change without notice.

## SYMBOLS AND NOTATION

Logic equations, in the descriptions of programmed instructions and throughout this manual, use the following symbols and notation. All abbreviations used in this book are not listed here, but are defined with their first use.

| Symbol Or Notation | Interpretation |
|---|---|
| A | The A register (accumulator) or its contents. |
| A, E | The combined contents of both the A and E registers in double-word format. |
| CPI | The central processor indicator register or its contents. |
| CPS | The central processor status register or its contents. |
| $\text{DISP}_e$ | The displacement field of relative instructions extended left to form a 15-bit address. |
| E | The arithmetic extender register or its contents. |
| EA | The effective address register, its contents , or the location specified by the effective address. |
| EA1 | The effective address derived from the first word of a two-word instruction. |
| EA2 | The effective address derived from the second word of a two-word instruction. |
| (EAdp) | The contents of the location specified by the effective address expanded to form a double-precision floating-point number. |
| (EA, EA + 1) | The contents of two consecutive memory locations, the first of which has the address specified by the effective address of an instruction. |
| $\text{EA}_F$ | Final computed effective address. |
| $\text{EA}_I$ | Intermediate effective address. (An effective address that has received some, but not all, of the arithmetic operations required to compute a final effective address.) |
| EMB | The executive mode bit (bit 00) of the privilege and level register. |
| FNC | The memory-protection fence register or its contents. |
| FNV | The fence violation bit (bit 14) of the central processor status register. |

| Symbol Or Notation | Interpretation |
|---|---|
| FPL | The floating-point long indicator (bit 01) of the central processor indicator bit. |
| FPO | The floating-point overflow indicator (bit 19) of the central processor status register. |
| FPU | The floating-point underflow indicator (bit 00) of the central processor indicator register. |
| IR | The instruction register or its contents. |
| IR06-08 | The mode field of the instruction register or its contents. |
| IR'06-08 | The mode field of an indirect vector. |
| IR12-23e | The displacement field of relative instructions extended left to form a 15-bit address (same as $DISP_e$). |
| IR'09-23 | The indirect address supplied by an indirect vector. |
| OV | The overflow indicator (bit 18) of the central processor status register. |
| PC | The program counter register, its contents, or the location specified by the program count. |
| PL | The privilege and level register or its contents. |
| RMP | The register manipulation privilege indicator (bit 01) of the privilege and level register. |
| SHC | The shift counter or its contents. |
| XA | The index A register or its contents. |
| XB | The index B register or its contents. |
| XC | The index common register or its contents. |
| XE | The index executive register. |
| XT | The index top-of-stack register or its contents. |

| Symbol Or Notation | Interpretation |
|---|---|
| (XT)OLD | The former or initial contents of the index top-of-stack register. |
| → | Is copied into, or replaces the contents of. |
| B → A | The contents of the B register are transferred into, or replace the contents of, the A register. |
| PC + 1 → PC | The contents of the PC are incremented by one. |
| (EA) → A | The contents of the EA specify a location whose contents are transferred into the A register. |
| + | The arithmetic sum or the logical inclusive OR operation. |
| - | The arithmetic difference or the logical inversion (when placed immediately after a line or register symbol) operation. |
| ⊕ | The logical exclusive OR operation. |
| & | The logical AND operation. |
| A23 → E00 | Bit 23 of register A is shifted (right) into bit 00 of register E. |
| XA → (EA09-23) | The contents of the XA register are transferred into bits 09 through 23 of the location specified by the contents of the EA. |
| 0 → (EA00-08) | Zeros are transferred into bits 00 through 08 of the location specified by the contents of the EA. |

viii Preface

# Contents

CONTENTS *(contd)*

CONTENTS *(contd)*

Section 3. FORMATS AND ADDRESSING

Section 4. INSTRUCTION REPERTOIRE

## CONTENTS *(contd)*

CONTENTS *(contd)*

## CONTENTS *(contd)*

Section 5. INTERRUPT STRUCTURE

Section 6. OPERATING PROCEDURES

CONTENTS *(contd)*

Section 7. DRUM

Section 8. REAL TIME CLOCKS

## CONTENTS *(contd)*

## FIGURES

## TABLES

CONTENTS *(contd)*

# 1/INTRODUCTION

The FOX 1 central processor integrates, synchronizes, and controls the functions of all other components of the system. Triggered by an internal real-time clock or external process events, foreground programs in the processor scan process instrumentation inputs, perform control algorithm calculations, and transmit the results to process control devices. Background programs perform data processing and software development tasks on a time-available basis. Advanced hardware design features provide reliable real-time processing, extensive general purpose capability, and rapid, efficient switching between foreground and background work.

Some of these features are:

a. Word-oriented memory, 24-bit (plus parity)

b. Information manipulation on single words, double words, bytes (variable size), and bits.

c. Expandable memory from 16,384 to 32,768 words, in increments of 8,192 or 16,384 words.

d. Automatic shutdown and startup procedures, and memory protection for power fail-safe operation.

e. Flexible addressing techniques:

   (1) Direct addressing of all 32,768 words (without need of indirect, indexing, or displacement addressing).

(2) Immediate addressing of operands (space- and time-saving).

(3) Two-level indexing (preindexing, postindexing, or both).

(4) Two-level indirect addressing.

f. Sixteen addressable hardware registers for arithmetic and control operation.

g. Selective register and memory protection.

h. Address stop capability (optional) for on-line program debugging.

i. Privileged assignment of system resources.

j. Priority interrupt system offering 12 standard or 24 optional priority interrupt levels. Each level is capable of handling software, hardware, and IO interrupts simultaneously. Up to 24 devices may be assigned to each interrupt level.

k. Automatic, rapid-context switching for fast interrupt service.

l. A trapping system that enables use of operating system resources without interference with real-time operation.

m. Multiple real-time clocks for timing programmed events.

n. Complete and powerful instruction repertoire consisting of the following:

(1) Bit and byte manipulation.

(2) Multiple word data transfers.

(3) Register-to-register operation.

(4) Fixed point arithmetic operations, in single-word and double-word data formats, with overflow detection and control.

(5) Floating point arithmetic operations in long and short data formats, with overflow, underflow, and normalization control.

(6) All floating point arithmetic operations are performed in double-precision form for greater accuracy. Single-precision operands are converted to double-precision format; the arithmetic is performed; then the results are rounded and returned to single-precision format.

(7) Logical arithmetic operations (AND, OR, exclusive OR, masked store).

(8) Arithmetic operations in byte manipulation.

(9) Comparison operations, on fixed point and floating point data.

(10) Shifting operations on both long and short data: arithmetic, logical, and normalize.

(11) Push-down stack manipulating instructions for dynamic control of memory allocation, subroutine calls, and subroutine exits.

o. Complete IO system, providing the following features:

(1) Programmed IO control of all IO operations.

(2) Privileged IO usage on basis of need.

(3) Up to 24 devices assigned to each interrupt level.

(4) Eight IO channels (two high-speed, six low-speed) with direct access to memory.

(5) IO command chaining for uninterrupted channel IO transfers.

(6) IO status checking without activity disturbance.

(7) Direct IO without use of channels.

The major elements of the FOX 1 central processor, as show in Figure 2-1-1, are:

a. A heirarchical storage system consisting of

(1) A core memory optionally expandable from the basic 16,384 words to 24,576 or 32,768 words.

(2) A drum memory providing 256 or 512 tracks (1024 words per track) of bulk data storage.

(3) Hardware registers (16 of which are program addressable) to store address and control information pertinent to individual programs, and data in process.

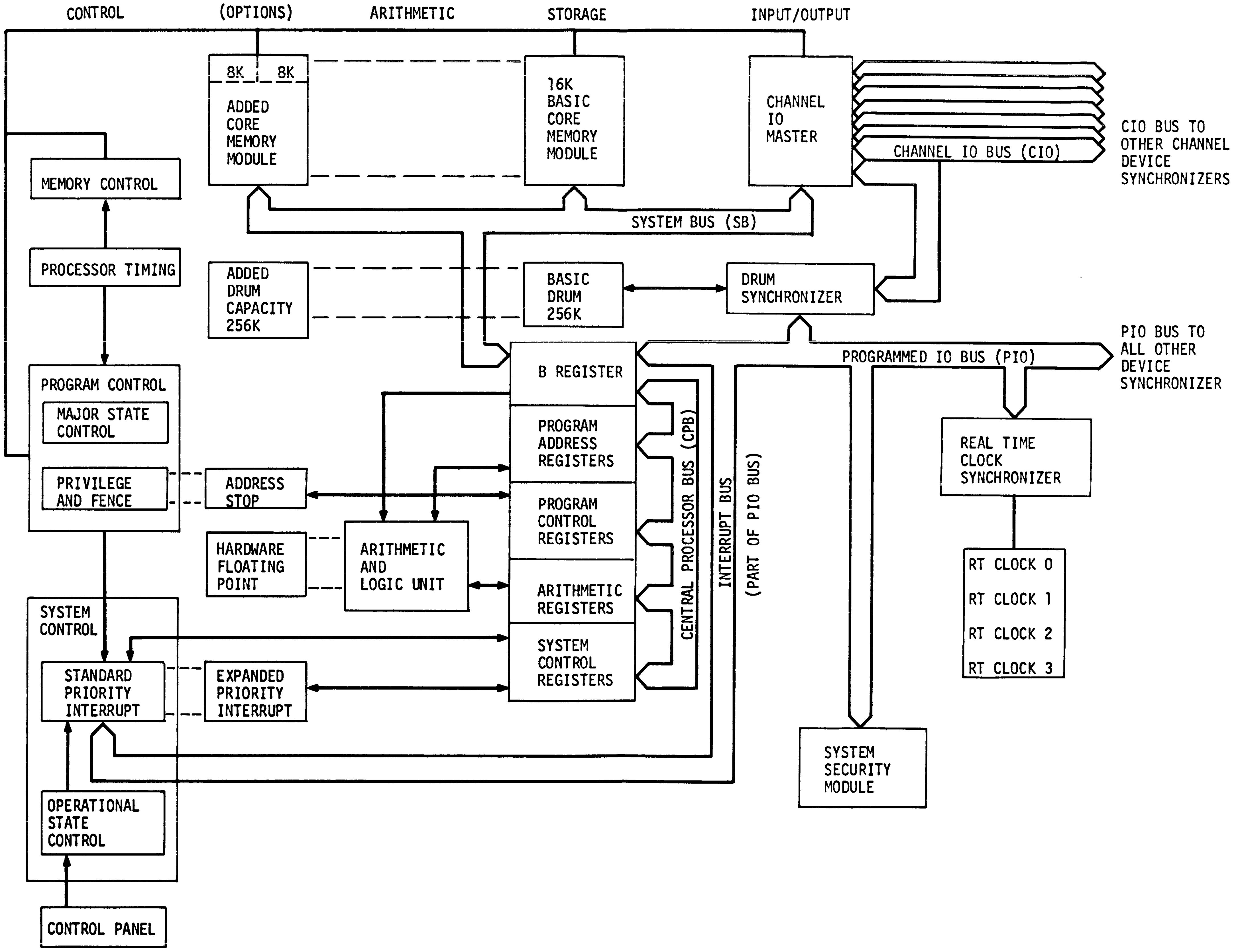


Figure 2-1-1. Central Processor Simplified Block Diagram

(4) A bus system interconnecting storage media, the processor, and the input/output subsystem.

b. A processor that controls all system activity, including the following:

(1) Memory Access Control -- priority allocation and cycle control.

(2) Program Control -- major state control.

(3) System Control -- operational state control and priority interrupt.

c. An input/output section including:

(1) Programmed IO -- one common data bus using the processor's arithmetic registers to receive or transmit data.

(2) Channel IO -- using up to eight dedicated data channels interfacing with an IO master control unit that provides direct memory access.

# 2/FUNCTIONAL HARDWARE

## STORAGE

Core memory stores the operating system software and several user programs, drum memory provides a bulk storage[1] pool for additional programs and data, and hardware registers contain program operating limits, address modification constants, and arithmetic operands and results. All media use the same 24-bit data word length, but core and drum also store a parity bit for each word. Parity is checked at the storage interface -- the parity bit is not transmitted to the processor or channel IO master.

A storage protection system limits the area of core available to individual programs (memory fence and address stop protection) and requires an appropriate privilege to use the drum (system IO privilege) or to alter the contents of addressable registers (executive mode bit or register manipulation privilege, software interrupt privilege, or trace control privilege, depending upon the specific register involved).

### Core Memory

Core memory operates asynchronously from the processor logic, performing a full cycle (read and restore, or clear and write) operation in 960 nanoseconds. Data read from memory are available to the processor or channel IO master 640 nanoseconds after the start of the memory cycle.

[1] Bulk storage operations mentioned here as being performed by the drum can, at times, be performed by the disk. The optional disk memory subsystem is described in detail in Volume 3, Section 6.

Core Capacity - Storage capacity of the basic 2 1/2 D, 22-mil ferrite-core memory is 16,384 words, and can be expanded optionally to 24,576 words or 32,768 words. Memory locations are identified by octal numbers from 00000 through 37777 for 16K words, through 57777 for 24K words, or through 77777 for 32K words. All memory locations can be directly addressed. Indirect-addressing, relative-addressing, and address-indexing facilities are provided to simplify address manipulation and to make programs relocatable.

Memory Allocation Features - The central processor hardware provides very flexible methods for allocating memory areas for software systems operations. These methods are a combination of hardware and software address manipulation, using the index top-of-stack register (XT register), the index common register (XC register), and the index executive register (XE register). (Refer to the description of these registers under Program Address Registers.)

Dedicated Core Locations - Areas of core memory are reserved for, or dedicated to, special use as shown in Table 2-2-1. Sixteen program-addressable CP hardware registers are assigned addresses 77760 through 77777. In systems containing 32K of memory, the processor is inhibited from accessing the last 16 core locations. (Channel IO master access to these locations -- although permitted -- is of no practical use, since the processor cannot access them for IO error correction.)

## Drum Memory

The following deals with the drum only as an extension of the storage capacity of core memory. Details of drum hardware and programming are covered in Section 7 of this volume.

Table 2-2-1. Dedicated Memory Allocation

| OCTAL ADDRESS FIELD | DECIMAL LOCATIONS | USAGE |
|---|---|---|
| 00000 to 00027 | 24 | Interrupt Block Vectors |
| 00030 to 00031 | 2 | Nondedicated Memory |
| 00032 to 00037 | 6 | IO Data Address Registers for CIO Low-Speed Channels* |
| 00040 to 00041 | 2 | Nondedicated Memory |
| 00042 to 00047 | 6 | IO Last Address Registers for CIO Low-Speed Channels** |
| 00050 to 00057 | 8 | IO Next Instruction Address Registers for ALL CIO Channels; provides IO Linkage*** |
| 00060 | 1 | Nondedicated Memory |
| 00061 | 1 | Floating Point Temporary Storage |
| 00062 to 00067 | 6 | Nondedicated Memory |
| 00070 | 1 | Restart Trap |
| 00071 to 00077 | 7 | Nondedicated Memory |
| 00100 | 1 | Fence Violation Trap (BSP, BSR) |
| 00101 | 1 | Literal Trap (BSP, BSR) |
| 00102 to 77757 | --- | Nondedicated Memory**** |
| 77760 to 77767 | 8 | Protected Registers |
| 77770 to 77777 | 8 | Unprotected Registers |

NOTES

*Used to store memory address of first transmitted word.

**Computed from IO data address register, plus word count (address of last word used in data transmission).

***Indirect pointers to channel IO command and three-word information block (set by program).

****Depending upon memory size, this block is 16,302 words, 24,494 words, or 32,686 words, yielding a total nondedicated memory size of 16,314 words, 24,506 words, or 32,698 words.

Drum memory provides rapid access to relatively large amounts of data. Two choices of drum capacity are available: 256 tracks or 512 tracks (1 track = 1024 words). The use of 50 or 60 cycle primary power determines drive motor speed, fixing the time required for one drum revolution at 20.4 milliseconds or 16.7 milliseconds, respectively. During one drum revolution, 1024 words (one track) can be read or written. Thus, the continuous data rate (up to one track) is one computer word transmitted every 15.2 microseconds. The average track access time is 8.3 milliseconds (one-half revolution). Since channel IO transfers data to or from the drum by direct memory access, other program instructions can be executed while a drum transfer is taking place.

Program Swapping - FOX 1 programs may be separated into core-resident and drum-resident. The operating system and associated tables, common subroutines, and time-critical foreground (process control) programs are core-resident. But the majority of the programs are drum-resident, read into core before execution time by the operating system.

Program Relocation - Space in core memory is partitioned to receive drum-resident programs of several different priorities. If the core partition for the priority level of a given program is not available immediately prior to its execution, the operating system reassigns the program to an available partition of lower priority. Therefore, all programs controlled by the operating system must be relocatable.

Program Address Registers

In contrast to core and drum, which provide general purpose storage, most registers are limited to specific types of information. Therefore, register storage can be separated according to use into four groups:

a. Program Address Registers: those registers involved in address arithmetic or supplying operating parameters for individual instructions or programs.

b. Program Control Registers: those registers containing operating limits for individual programs.

c. Arithmetic and Logic Registers: those registers involved in computation or decision logic.

d. System Control Registers: those registers controlling the multiprogramming environment.

Operating system software and relocation hardware use the registers in the first group, the program address registers, to make programs fully relocatable. Relative addressing uses the contents of one of four dedicated base registers as a reference point for locating program instructions, common operating data, initialization data, or subroutine linkage information. The address field of a relocatable instruction can provide a displacement up to ±2048 from the value designated by the base register. Either of two index registers can be used to change the sum of base value and displacement up to ±16,384.

Nine registers (two of which are also counters) make up the program address group. The six relocation registers in the group are directly addressable by the program; only the XE requires a register privilege to permit changing its contents. The three nonaddressable registers included in this group are:

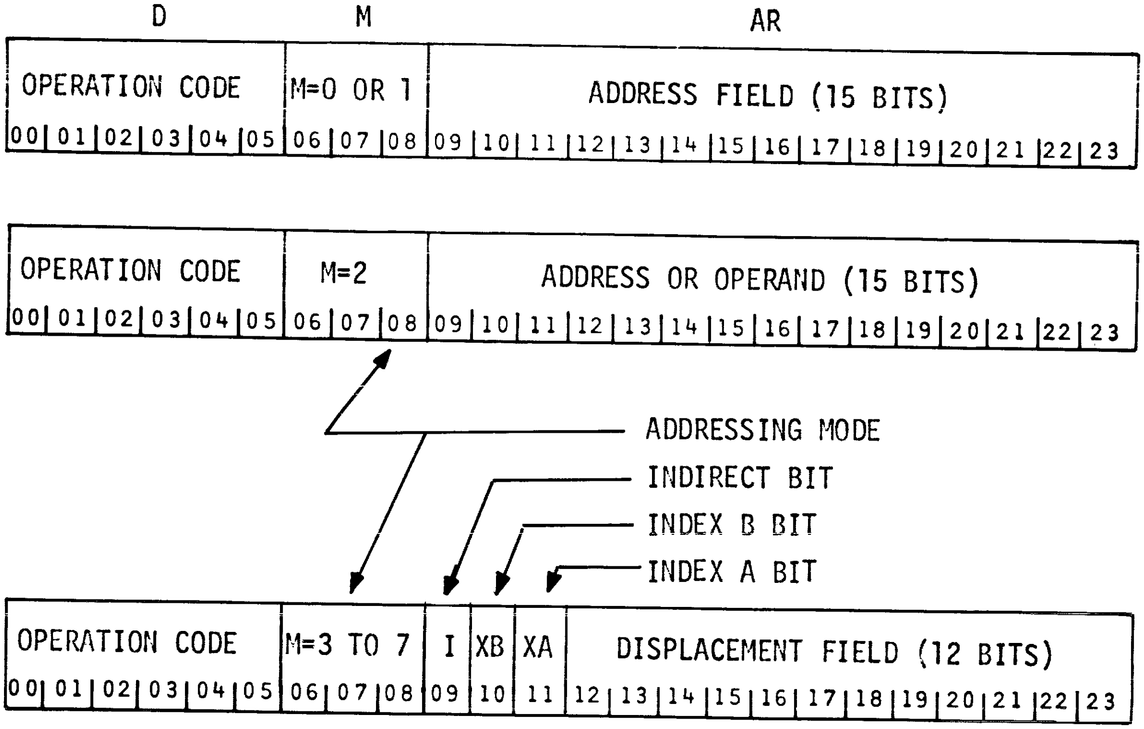
a. Instruction Register
b. Shift Counter
c. Effective Address Register

The following six address modification registers are program addressable:

d. Program Counter
e. Index Common
f. Index Executive
g. Index Top-of-Stack
h. Index A
i. Index B

Addressing, arithmetic, and control registers are shown in detail in Figure 2-2-1, and their characteristics summarized in Table 2-2-2.

Instruction Register (IR, Not Addressable, 24 Bits) - Hardware directs the instruction word read from core into the instruction register (IR), which, in turn, is separated into three sub-registers: the D register, containing the 6-bit operation code, the M register, containing the 3-bit mode field; and the address register (AR), containing a 15-bit field.



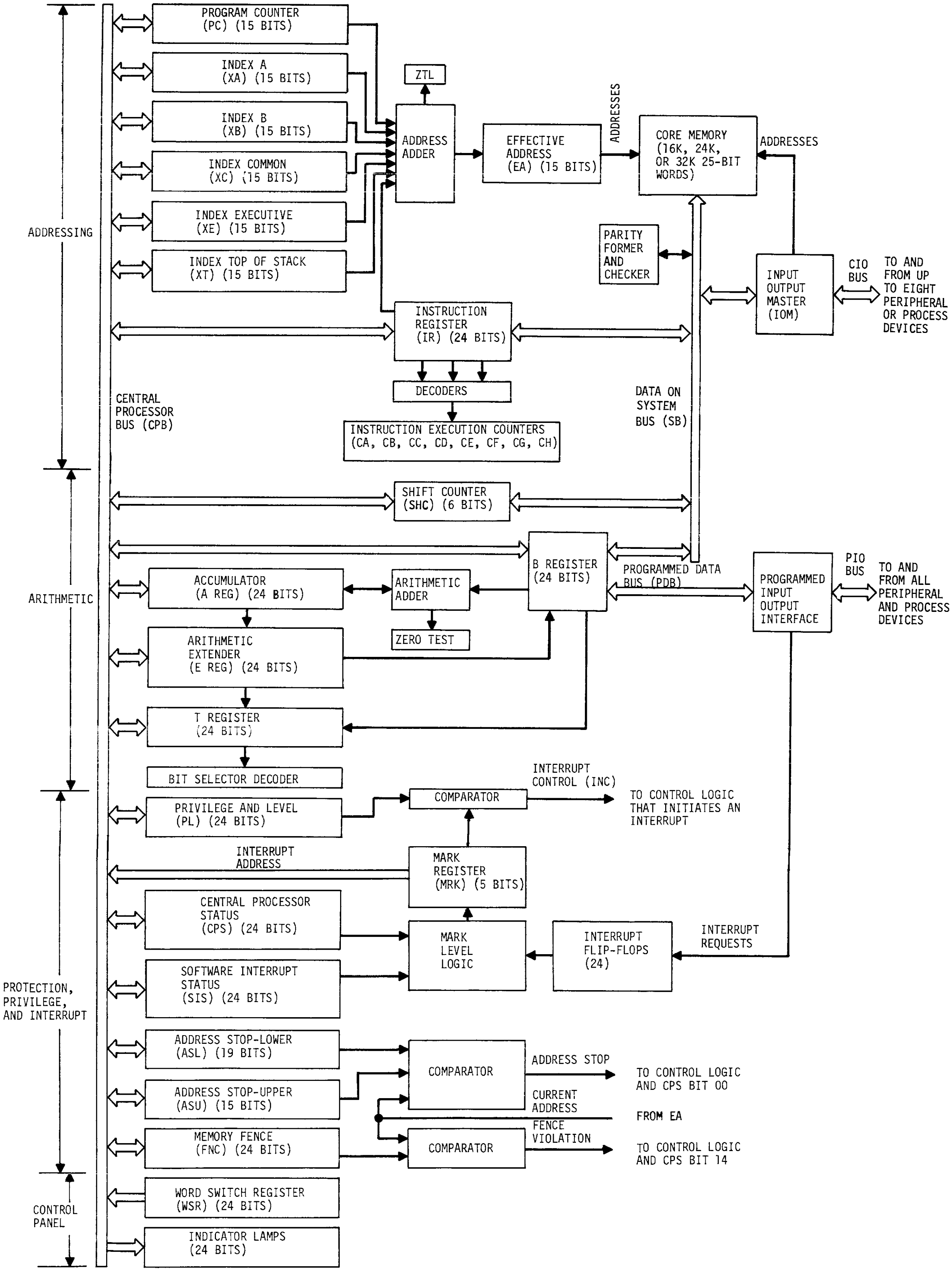

PROGRAM COUNTER (PC) (15 BITS)
INDEX A (XA) (15 BITS)
INDEX B (XB) (15 BITS)
INDEX COMMON (XC) (15 BITS)
INDEX EXECUTIVE (XE) (15 BITS)
INDEX TOP OF STACK (XT) (15 BITS)
ZTL
ADDRESS ADDER
EFFECTIVE ADDRESS (EA) (15 BITS)
ADDRESSES
CORE MEMORY (16K, 24K, OR 32K 25-BIT WORDS)
ADDRESSES
PARITY FORMER AND CHECKER
INPUT OUTPUT MASTER (IOM)
CIO BUS
TO AND FROM UP TO EIGHT PERIPHERAL OR PROCESS DEVICES
ADDRESSING
INSTRUCTION REGISTER (IR) (24 BITS)
DECODERS
INSTRUCTION EXECUTION COUNTERS (CA, CB, CC, CD, CE, CF, CG, CH)
CENTRAL PROCESSOR BUS (CPB)
DATA ON SYSTEM BUS (SB)
SHIFT COUNTER (SHC) (6 BITS)
B REGISTER (24 BITS)
PROGRAMMED DATA BUS (PDB)
PROGRAMMED INPUT OUTPUT INTERFACE
PIO BUS
TO AND FROM ALL PERIPHERAL AND PROCESS DEVICES
ARITHMETIC
ACCUMULATOR (A REG) (24 BITS)
ARITHMETIC ADDER
ZERO TEST
ARITHMETIC EXTENDER (E REG) (24 BITS)
T REGISTER (24 BITS)
BIT SELECTOR DECODER
PRIVILEGE AND LEVEL (PL) (24 BITS)
COMPARATOR
INTERRUPT CONTROL (INC)
TO CONTROL LOGIC THAT INITIATES AN INTERRUPT
INTERRUPT ADDRESS
MARK REGISTER (MRK) (5 BITS)
CENTRAL PROCESSOR STATUS (CPS) (24 BITS)
MARK LEVEL LOGIC
INTERRUPT FLIP-FLOPS (24)
INTERRUPT REQUESTS
SOFTWARE INTERRUPT STATUS (SIS) (24 BITS)
PROTECTION, PRIVILEGE, AND INTERRUPT
ADDRESS STOP-LOWER (ASL) (19 BITS)
COMPARATOR
ADDRESS STOP
TO CONTROL LOGIC AND CPS BIT 00
ADDRESS STOP-UPPER (ASU) (15 BITS)
CURRENT ADDRESS
FROM EA
MEMORY FENCE (FNC) (24 BITS)
COMPARATOR
FENCE VIOLATION
TO CONTROL LOGIC AND CPS BIT 14
CONTROL PANEL
WORD SWITCH REGISTER (WSR) (24 BITS)
INDICATOR LAMPS (24 BITS)

Figure 2-2-1. Central Processor Block Diagram

The D register is decoded to answer the following questions:

a. Is a memory operand access (MOA) required for this instruction?

b. If the literal mode is specified, is the value in the AR an operand (literal class 1), or an address (class 2)? For class 1 literals (operands), bit 09 of the AR is left-propagated to form a 24-bit signed data word in the B register. Class 2 literals, memory (or register) addresses, are left at 15 bits. Literals used in subroutine linkage or long form floating point instructions are placed in a third category, class special.

c. Which execution counter controls this instruction?

d. Which control lines provided by the selected execution counter are appropriate to this specific instruction?

For absolute modes, the M register indicates whether AR contains a direct address (M = 0), an indirect address (M = 1), or a literal value (M = 2). For relative modes, the M register indicates the source of the value to be added to the contents of the AR: PC (M = 4), XC (M = 5), XT (M = 6), or XE (M = 7). When M = 3, a value of 00000 is added to AR.

For absolute modes 0 and 1, AR contains a 15-bit memory (or register) address; for relative modes 3 through 7, the AR is subdivided into four fields: I, the indirect bit, which identifies the contents of AR as the address of an address; XB, the bit specifying the use of the XB index register for address computation; and XA, the bit specifying the use of the XA index register. The remaining 12-bit value represents a signed displacement from the base reference. AR provides one of the inputs to the address adder during effective address computation.

Table 2-2-2. Addressable Registers

| USE | NAME | MNEMONIC | SIZE (IN BITS) | ADDRESS | INTERRUPT ACTION (S=SAVED I=INITIAL.) | PROTECTION PRIVILEGE[1] | VIOLATION | FORMAT AND BIT POSITION ASSIGNMENT<br>00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 |
|---|---|---|---|---|---|---|---|---|
| PROGRAM TESTING | Address Stop - Lower | ASL | 19 | 77760 | | TCP | RPV | * * * * * IR OR SM PC ADDRESS STOP (Lower) |
| | Address Stop - Upper | ASU | 15 | 77761 | | TCP | RPV | * * * * * * * * * ADDRESS STOP (Upper) |
| OPERATING SYSTEM | Index Executive | XE | 15 | 77762 | | RMP | RPV | * * * * * * * * * INDEX EXECUTIVE |
| MANUAL CONTROL | Word Switch | WSR | 24 | 77763 | | Read Only | None | † † † † † † † † † † † † † † † † † † † † † † † †<br>WORD SWITCH REGISTER |
| SYSTEM CONTROL | Software Interrupt Status | SIS | 24 | 77764 | | RMP, SIP | RPV, SIV | Software Interrupt Status<br>Interrupt Levels<br>00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 |
| | Central Processor Status | CPS | 24 | 77765 | | RMP | RPV | ASI 32K 24K 16K 8K * * * * * * * IOT IOU FNV RPV SIV ILS OV FPO * MPF MPE HLT |
| | Memory Fence | FNC | 24 | 77766 | S I | RMP | RPV | HIGH FENCE LOW FENCE |
| INDIVIDUAL PROGRAM CONTROL | Privilege and Level | PL | 24 | 77767 | S I | RMP[2] | RPV | EMB RMP SIP SIOP CIOP NIOP TCP ASE IAOK LOWEST PRIORITY UL HIGHEST PRIORITY LL INTERRUPT STATUS IS |
| ADDRESS MODIFICATION | Program Counter | PC | 15 | 77770 | S I | None | None | * * * * * * * * * PROGRAM COUNTER |
| | Index Common | XC | 15 | 77771 | S I | None | None | * * * * * * * * * INDEX COMMON |
| | Index Top-of-Stack | XT | 15 | 77772 | S I | None | None | * * * * * * * * * INDEX TOP-OF-STACK |
| | Index B | XB | 15 | 77773 | S I | None | None | * * * * * * * * * INDEX B |
| | Index A | XA | 15 | 77774 | S | None | None | * * * * * * * * * INDEX A |
| ARITHMETIC AND LOGIC | Accumulator (Arithmetic Register) | A | 24 | 77775 | S | None | None | ACCUMULATOR |
| | Arithmetic Extender | E | 24 | 77776 | S | None | None | ARITHMETIC EXTENDER |
| | Central Processor Indicator | CPI | 5 | 77777 | S | None | None | FPU FPL BP BZ BO * * * * * * * * * * * * * * * * * * *<br>00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 |

*Indicates bits that cannot be changed by any instruction. Also, if a register containing * is used as an operand, the value of * is 0.
†Indicates bits that can be read but not altered.

[1] EMB = 1 grants all privilege.
[2] SPL instruction can change IS within UL ≤ IS ≤ LL.

Shift Counter (SHC, Not Addressable, 6 Bits) - As a register, the shift counter extends the capacity of the IR during instruction access for the following types of information:

a. Secondary operation code (micro-code) for the Bit and Compare With Memory instructions.

b. Number of words to be moved by the Move Multiple instruction.

During instruction execution, SHC, used as a counter, performs the following functions:

a. Counts the number of bit positions shifted during the execution of Shift, Rotate, Bit, and Byte instructions.

b. Counts the number of words handled during execution of the Move Multiple instruction.

c. Controls the execution of arithmetic or logic operations that shift data in the A or E registers (the only addressable registers capable of shifting data).

d. Times bus exchanges during execution of the Programmed IO instruction to detect any abnormal delay.

Effective Address Register (EA Register, Not Addressable, 15 Bits) - The effective address register is the result register for the address adder, and supplies the address lines to the memory module for all memory access by the processor. Comparators in the fence and address stop logic check the output of the EA. The contents may be transferred between EA and PC directly, or through the central processor bus (CPB). At the end of each instruction execution, PC → EA (direct transfer) provides the memory module with the address for the next instruction. In the execution of a branch operation, EA → CPB → PC provides the starting point for a new sequence of instructions. (The branch address remains in the EA, as well, supplying memory with the address for the first instruction in the new sequence.)

Program Counter (PC Register, 77770, 15 Bits) - The PC register controls the sequence in which instructions are fetched from memory. At the end of each address modification state, the PC is incremented by one. During each execution state, the PC retains the address of the next instruction word. PC is used as a base register for effective address computation when the value of the instruction mode field is 4.

There is no method for privileged protection of the PC register. Instructions that generate a skip signal increment PC during execution (if the condition for a skip is satisfied), causing the next sequential instruction word to be omitted. Branch instructions (if the condition for a branch is satisfied) replace the contents of the PC with the effective address of the branch instruction. The Return From Subroutine instruction replaces the PC contents with a return vector previously stored in the push-down stack. PC may be addressed (77770 used as an effective address) by any memory-referencing instruction; however, alteration of PC must be done with care to maintain program control.

Information in the PC is stored when interrupt occurs, and a new address is loaded into the register. This accomplishes a branch to the program that handles the interrupt. The saved information is restored in PC by a Return From Interrupt instruction.

Index Common (XC Register, 77771, 15 Bits) - The XC register is one of the base registers used in relative addressing for effective address computation. XC-relative addressing is specified by the value 5 in the mode field of an instruction.

In operation under control of the real time executive system, the XC register contains an address that points to a common storage area within each user program. By operating convention, XC is established and maintained by the executive system and is not normally altered or manipulated by the user's program.

There is no method for privileged protection of the XC register. No instructions specifically use the XC register; however, the XC register address (77771) may be used as the effective address of any memory-referencing instruction.

The address in the XC register is stored when an interrupt occurs, and new information is loaded into the register. The saved information is reloaded into XC by a Return From Interrupt instruction.

Index Executive (XE Register, 77762, 15 Bits) - The XE register is used as a base register for effective-address computation. XE-relative addressing is specified by the value 7 in the mode field of an instruction.

The real time executive system reserves the XE register for its exclusive use. It is used primarily to point to header information for each program in core.

XE is protected from modification by the RMP bit of the PL register. Contents of XE are normally controlled by the real time operating system. No instructions specifically use the XE register; however, the XE register address (77762) may be the effective address of any memory-referencing instruction. The address in XE is neither stored nor altered when interrupt occurs.

Index Top-of-Stack (XT Register, 77772, 15 Bits) - The XT register is one of the base registers used in relative addressing for effective address computation. XT-relative addressing is specified by the value 6 in the mode field of an instruction.

The XT register normally contains an address that identifies the top (numerically highest address) of a push-down data stack. This data stack provides dynamic storage allocation for subroutine linkages. Three instructions (BSP, BSR, and BRU) manipulate the XT register to expand and contract the stack.

There is no method of privileged protection of the XT register. It is altered by the BSP, BSR, and BRU instructions. Its address (77772) may be used as the effective address by any memory-referencing instruction.

The address in the XT register is stored when interrupt occurs, and new information is loaded into the register. The saved information is reloaded into XT by a Return From Interrupt instruction.

Index A (XA Register, 77774, 15 Bits) - The XA register is used as a general or post-index register for effective address computation. Bit 11 of relative addressing instructions specifies use of XA.

A complete set of instructions is provided for control of XA. There is no method for privileged protection of the XA register. XA may be addressed (77774 used as an effective address) by any memory-referencing instruction. The information in the XA register is stored but not altered when interrupt occurs. The saved information is restored in XA by a Return From Interrupt instruction.

Index B (XB Register, 77773, 15 Bits) - The XB register is used as a general, pre-index, or post-index register for effective-address computation. Bit 10 of relative-addressing instructions specifies use of XB. Bits 09 and 11 of relative-addressing instructions determine if XB is to be used for pre-indexing or post-indexing.

A complete set of instructions is provided for control of XB. There is no method for privileged protection of XB. XB may be addressed (77773 used as an effective address) by any memory-referencing instruction. The information in the XB register is stored when interrupt occurs, and new information is loaded into the register. The saved information is restored in XB by a Return From Interrupt instruction.

Program Control Registers

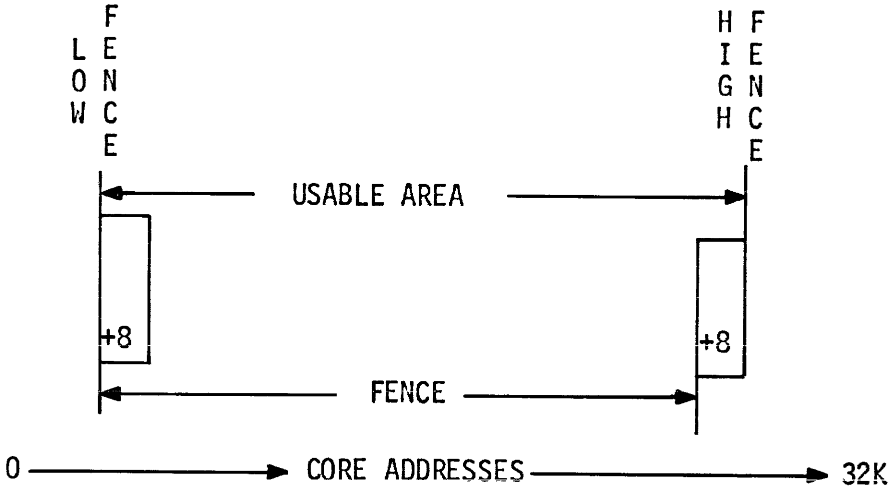
Four addressable registers, all of which are protected, make up the program control group. The first two registers, which provide the normal operating limits for each program, are:

a. Fence Register
b. Privilege and Level Register

Two optional registers providing special limits for program testing and development are:

c. Address Stop Upper
d. Address Stop Lower

Memory Protection Fence (FNC Register, 77766, 24 Bits) - FNC provides high (bits 00 through 11) and low (bits 12 through 23) address limits of memory that may be referenced by the operating program. These 12-bit fields hold the high-order 12 bits of the limit addresses (with the three low-order bits assumed to contain zeros). Thus, the limit addresses are always modulo 8, and the smallest area that can be limited by the fence is eight locations.



FNC prevents intrusion of one program on portions of memory reserved for other programs. When operation is under control of the real time executive system, fence limits are established for each user program. A hardware interrupt request is generated if the fence is violated.

Table 2-2-3 shows fence violation interrupt and trap conditions that can result from instruction fetch (PC violation), data store, or branch operations.

FNC is protected from modification by the RMP and EMB bits of the privilege and level register (PL) and may only be modified if the RMP or EMB bit (or both) of the PL register is set.

Table 2-2-3. Effect of Crossing Fence Boundaries[1]

| INSTRUCTION ACCESS STATE | | | | INSTRUCTION EXECUTION STATE | | | |
|---|---|---|---|---|---|---|---|
| LOCATION OF CURRENT INSTRUCTION COMPARED WITH FENCE | LOCATION OF PREVIOUS INSTRUCTION COMPARED WITH FENCE | RESULTING ACTION | ADDRESS MODIFICATION STATE | INSTRUCTION TYPE | LOCATION OF CURRENT INSTRUCTION COMPARED WITH FENCE | EFFECTIVE ADDRESS OF CURRENT INSTRUCTION COMPARED WITH FENCE | RESULTING ACTION |
| Outside | Inside | Interrupt | Operand fetch does not cause fence violation.<br>Indirect fetch does not cause fence violation, except that M' = 2 forces FNV. | Branch | Inside | Outside | Interrupt |
| Inside | Outside | Reset EMB. | | | Either | Inside | Execute |
| | | | | BSR/BSP (Except Literal) | Inside | Outside | Trap to 00100 |
| | | | | | Either | Inside | Execute |
| | | | | BSR/BSP (literal) | Either | Either | Trap to 00101 |
| | | | | Store | Either | Inside | Execute |
| | | | | | Either | Outside | Interrupt[2] |

NOTES: [1] If the executive mode bit (EMB) is set, the fence is ineffective.

[2] Data is not written, but instruction execution runs to completion before interrupt occurs.

The information in the FNC register is stored when interrupt occurs, and new information is loaded into the register. The saved information is restored in FNC by a Return From Interrupt instruction.

Privilege and Level (PL Register, 77767, 24 Bits) - Each program is granted certain privileges and program control levels. These privileges and levels are generally assigned by the real time executive system and implemented at the start of each program by loading of the PL register.

The PL register contains a 9-bit privilege field and three 5-bit level fields as follows:

| Privilege | | | | | | | | | Upper Level (UL) (00-$37_8$) (Lowest Priority) | | | | | Lower Level (LL) (00-$37_8$) (Highest Priority) | | | | | Interrupt Status (IS) (00-$37_8$) (Current Priority) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| EMB | RMP | SIP | SIOP | CIOP | NIOP | TCP | ASE | IAOK | | | | | | | | | | | | | | | |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Privileges assigned to the current program are loaded into bits 00 through 07 of the PL register. Bits set signify privileges allowed, and bits cleared signify privileges not allowed. Attempts to use a privilege that is not allowed cause an associated bit of the central processor status register to be set. This, in turn, initiates a level 01 interrupt. Thus, violation of privilege will be detected by the central processor.

Bits in the privilege field have the following meaning:

EMB - Executive Mode Bit. A mask used to interpret bits 01 through 06. If EMB or one of the bits 01 through 06 (or both) is set, the specified privilege is enabled; i.e., if EMB = 1, all privileges are enabled; and if EMB = 0, only those privileges that are set are enabled. EMB also specifies if the memory fence (FNC) is active (EMB = 0) or inactive (EMB = 1) and, thus, acts as the FNC privilege bit.

RMP - Register Manipulation Privilege. If RMP or EMB is set, the FNC, PL, CPS, SIS, and XE registers may be modified. If RMP and EMB are reset, this privilege is not allowed, and any attempt to modify these registers causes the RPV bit of the CPS register to be set.

SIP - Software Interrupt Privilege. If SIP or EMB is set, the bit manipulation instruction may alter a bit in the SIS register. If SIP and EMB are reset, this privilege is not allowed, and any attempt to modify SIS by the BIT instruction causes the SIV bit of the CPS register to be set. (An attempt to modify SIS by a store class instruction is dependent on RMP.) This privilege allows setting bits in SIS even if register manipulation privilege is not allowed.

SIOP
CIOP
NIOP - IO privilege. These bits, if set, identify the categories of IO that may be used by a program:

System IO Privilege (SIOP allows use of all IO devices)

Critical IO Privilege (CIOP allows use of CIOP and NIOP devices)

Noncritical IO Privilege (NIOP allows use of NIOP devices only)

Each IO device is placed in one of the listed categories by its hardware assigned address. The real time executive system controls the assignment of IO privilege. Any attempt to use a category without privilege (i.e., the corresponding bit and EMB reset) causes the IOU bit of the CPS register to be set.

TCP - Trace Control Privilege. If TCP or EMB is set, the ASL and ASU registers may be modified. If TCP and EMB are reset, this privilege is not allowed, and any attempt to modify the ASL and ASU registers causes the RPV bit of the CPS register to be set.

ASE - Address Stop Enabled. If ASE is set, the address stop capability (if implemented) is to be effective. If ASE is reset, address stop capability is ignored.

IAOK - Instruction Adress OK. Hardware sets this bit at the end of each instruction access state if the instruction just fetched is within the fence limits. If the instruction is outside the fence, IAOK is reset. It is used during the next instruction fetch, along with the fence comparator outputs, to detect crossing from inside to outside, or from outside to inside the fence limits.

UL/LL/IS - Upper Limit/Lower Limit/Interrupt Status. Every program has a priority level. This level is set in the interrupt status (IS) field of the PL register. During execution of a program, all interrupts of equal (or of lower) priority than IS (equal to or greater than the numeric value of IS) are disabled. During execution of the program of level IS, the value of IS may be altered by an SPL instruction. If an SPL changes the setting of IS, the setting must be such that LL ≤ New IS ≤ UL. If this relationship is violated, the register privilege violation (RPV) bit of the CPS register is set, which causes an interrupt at level 01.

PL is protected from alteration if the RMP and EMB bits are reset. The IS field is also protected by the boundary limits specified by UL and LL.

The information in the PL register is stored when interrupt occurs, and new information is loaded into the register. The saved information is restored in PL by a Return From Interrupt instruction.

| NOT USED | | | | | | | | | ADDRESS STOP UPPER (ASU) | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

| NOT USED | | | | | I<br>R | O<br>R | S<br>M | P<br>C | ADDRESS STOP LOWER (ASL) | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

$\underbrace{05\ 06\ 07\ 08}$ ADDRESS STOP CONDITION FIELD

Address Stop Registers - Operations in any of four categories defined by the stop condition field may be barred inside the address limits set by the two address stop registers.

Address Stop-Upper (ASU, 77761, 15 bits) contains the fifteen bit address of the upper limit.

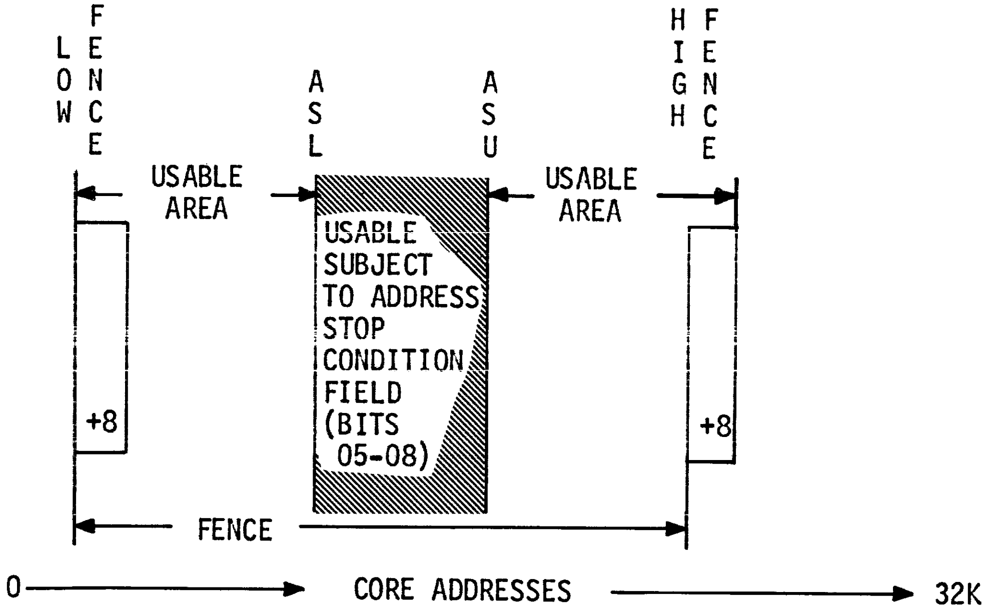
Address Stop-Lower (ASL, 77760, 19 bits) contains two fields:

a. Address Stop Condition Field (bits 05 through 08). This field specifies the circumstances that might cause an address stop condition. Each bit corresponds to a memory access function (fetch indirect vector, fetch operand, store or modify, or fetch instruction word). If the memory accessing address is within the limits specified by ASL and ASU and the bit in ASL corresponding to the memory access function is set, the address stop condition is satisfied:

| Bit | Mnemonic | Function |
|---|---|---|
| 05 | IR | Fetch indirect vector |
| 06 | OR | Fetch operand |
| 07 | SM | Store or modify |
| 08 | PC | Fetch next instruction |

The ASI bit (bit 00) of the CPS register is set if an address stop condition is satisfied, the address stop capability is implemented, and the ASE bit of the privilege and level register is set. This causes an interrupt at level 11, where 12 levels are implemented, or level 23, where 24 levels are implemented.

b. Lower Address Store Memory Boundary Address (bits 09 through 23). ASU contains the upper address stop memory boundary address.



ASU and ASL are protected from modification if the TCP and EMB bits of the PL register are reset. Neither ASU nor ASL is stored or initialized when interrupt occurs.

Arithmetic Registers

Three of the five registers making up the arithmetic group are directly addressable by the program; none is protected. The three addressable registers are:

a. Accumulator
b. Arithmetic Extender
c. Central Processor Indicator

The two nonaddressable registers are:

d. B Register
e. T Register

Accumulator (A Register, 77775, 24 Bits) - The A register is the main register of the processor arithmetic unit providing one of the inputs to the adder, and serving as the result register. The utility of the A register is illustrated by the variety of information that it can hold. Information type and format in the A register is a function of the instructions that use the register in their execution. Some types of information held in the A register are:

a. One of the operands and the signed sum or difference of single-precision add or subtract instructions.

b. The multiplicand and the high-order 24 bits (including sign) of the product of fixed-point multiply operations.

c. The 24 high-order bits (including sign) of the dividend and the signed quotient of fixed-point divide operations.

d. The 24 high-order bits of one operand (including sign and characteristic) and result of all floating-point operations.

e. One operand and the result of all logical operations.

f. The operand or the 24 high-order bits of the operand of shift/rotate and normalize operations.

g. One operand of compare operations.

h. One operand and the result of byte manipulation.

i. The test value of some branch instructions.

j. Input/output information involved in programmed IO operations.

All arithmetic overflows and floating point underflows occur in the accumulator.

The A register may be addressed (77775 used as the effective address) by any memory-referencing instruction. There is no method for privileged protection of the A register. Information in the A register is stored but not altered when interrupt occurs. The saved information is restored in the A register by a Return From Interrupt instruction.

Arithmetic Extender (E Register, 77776, 24 Bits) - The E register is an auxiliary register generally used by the arithmetic unit to extend the accumulator for double-precision arithmetic. Some types of information held in the E register are:

a. The 24 low-order bits of one operand and of the result of all double-precision arithmetic operations, both fixed point and floating-point.

b. The low-order 24 bits of the product of fixed point multiply operations.

c. The low-order 24 bits of the dividend and the remainder of fixed point divide operations.

d. The mask of masked store operations.

e. One operand of compare operations.

f. The specification parameters of byte manipulation operations.

g. The result of generate-effective-address operations.

h. The 24 low-order operand bits of double-precision shift and rotate operations.

i. The operand of rotate-left E operations.

The E register may be addressed (77776 used as an effective address) by any memory-referencing instruction. There is no method for privilege protection of the E register. Information in the E register is stored but not altered when interrupt occurs. The saved information is restored in the E register by a Return From Interrupt instruction.

<u>Central Processor Indicator (CPI Register, 77777, Five Bits)</u> - The CPI register records transient arithmetic conditions. Each bit of the register signifies a set/reset result of instruction execution. The CPI register format is:

| | | | | |
|---|---|---|---|---|
| FPU | FPL | BP | BZ | BO |
| 00 | 01 | 02 | 03 | 04 |

where:

FPU - Floating Point Underflow. FPU is set whenever the characteristic of a floating point operand or result is reduced below zero during execution of a floating point instruction.

Execution of an A register (or A and E register) load operation (LDA, LDL) resets FPU.

FPL - Floating Point Long. FPL indicates that the data format of the operand in the A register or the A and E registers is for floating point arithmetic operations. FPL is set when a Load Long, Add Long, or Subtract Long (LDL, ADL, SBL) instruction is executed and is reset when a Load A, Store Normalized and Rounded, Add Short, or Subtract Short instruction (LDA, SNR, ADD, SUB) is executed. Because all floating point arithmetic uses the long (double-precision) format, FPL = 0 specifies to the arithmetic unit that the operand in the A register must be converted to double-precision format before execution of a floating point operation; FPL = 1 specifies that no conversion is necessary; (i.e., the operand in A and E is in long format).

BP - Byte Parity. BP specifies the results of byte parity (BYT) instructions. BP set indicates odd parity in BYT results. BP reset indicates even parity in BYT results.

BZ - Byte Zero. BZ indicates the results of a BYT instruction. BZ set indicates non-zero BYT results. BZ reset indicates zero BYT results.

BO - Byte Overflow. BO indicates field overflow of the BYT instructions. BO is set by any arithmetic carry out of the high-order bit of the result field during BYT instructions.

NOTE

> BP, BZ, and BO are reset before execution of a BYT instruction, unless specified otherwise by the LV parameter of the instruction.

The CPI register may be addressed (77777 used as an effective address) by any memory-addressing instruction. There is no method for privileged protection of the CPI register. Information in the CPI register is stored but not altered when interrupt occurs, and it is restored in the CPI register by a Return From Interrupt instruction.

B Register (Not Addressable, 24 Bits) - The B register is the universal data register for the processor -- the only register connected to all the data buses in the processor (excluding the CIO bus). It receives or transmits data words from or to the core memory module (over the system bus), any of the addressable registers (over the central processor bus), and any of the device synchronizers (over the programmed IO bus). The accumulator and the B register are the only two inputs to the arithmetic adder.

T Register (Not Addressable, 24 Bits) - The T register is an auxiliary register generally used to extend the capacity of the B register for double word operands. The T register provides temporary storage for the contents of the E register when floating point hardware exchanges A,E contents with B,T. The Move Multiple instruction hardware transfers EA contents to the T register to save the initial address of the "from" block while it computes the initial address of the "to" block. The Bit instruction hardware uses positions 19 through 23 of the T register to identify the bit to be manipulated within the addressed data word.

## System Control Registers

Three of the registers included in the system control group are addressable. Two of the addressable registers are protected, the third, WSR, is a read only register.

The following registers make up the system control group:

a. Central Processor Status Register
b. Software Interrupt Status Register
c. Interrupt Flip-Flops (not addressable)
d. Mark Register (not addressable)
e. Word Switch Register (read only)

<u>Central Processor Status (CPS Register, 77765, 24 Bits)</u> - The CPS register contains four hard-wired bits specifying the implemented memory capacity, eleven bits that are individually set when security logic detects a hardware error, program error, or an impending power failure, and one bit that is set by executing the Halt instruction.

The 16 one-bit fields and 8 unused bits of the CPS register are in the following format:

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ASI | 32K | 24K | 16K | 8K | X | X | X | X | X | X | X | IOT | IOU | FNV | RPV | SIV | ILS | OV | FPO | X | MPF | MPE | HLT |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

LEVEL 11 OR 23

CAUSE LEVEL 01 INT.

LEVEL 00

Interrupts on three different levels can be initiated by settings in the CPS register. Setting of bit 00 initiates an interrupt at either level 11 or level 23, depending upon the number of levels available in the system. If any one of bits 12 through 19 is set, an interrupt to level

01 is initiated. These bits record privilege violation and arithmetic overflow. If any one of bits 21 through 23 is set, an interrupt to level 00 is initiated. The conditions under which each of the bit fields are set are:

ASI - Address Stop Interrupt. Set when all the following are satisfied:

a. Address stop capability is implemented.

b. Any of the conditions specified by bits 05 through 08 of the ASU register is satisfied.

c. The ASE bit of the PL register is set to enable address stop.

For a system with the basic 12 levels of priority interrupt, ASI occurs at level $11_{10}$. Where the 24-level expanded priority interrupt option is implemented, ASI occurs at level $23_{10}$. Address Stop operation is temporarily inhibited during the fetch of the implicitly-addressed parameter (K) of a Branch and Save Region (BSR) instruction, and during the hardware save-load interrupt process.

32K - Amount of Available Memory. These hard-wired bits indicate the
24K amount of memory available. All pertinent addressing used by the
16K central processor is checked against these settings. Addresses
8K that exceed available memory cause ILS (bit 17) to be set, which initiates an interrupt at level 01. The settings are:

| Memory Available | Bit Settings | | | |
|---|---|---|---|---|
| | 01 | 02 | 03 | 04 |
| 8,192 words | 0 | 0 | 0 | 1 |
| 16,384 words | 0 | 0 | 1 | 1 |
| 24,576 words | 0 | 1 | 1 | 1 |
| 32,768 words | 1 | 1 | 1 | 1 |

IOT - IO Response Time Violation. Set if an IO request receives no response from the specified device within the time limits (approximately 12 μsec) set by the PIO logic. If the time limit is exceeded, the PIO instruction that initiated the IO request is aborted, bit 12 is set, and interrupt occurs.

IOU - IO Usage Violation. Set when a program requests IO service for which it does not have privilege (as specified by bits 03, 04, or 05 of the PL register).

FNV - Fence Violation. Set when one of the memory protection fence interrupt conditions occurs (with EMB = 0).

RPV - Register Privilege Violation. Set when the privilege specified by EMB and RMP of the PL register is violated. This prevents unauthorized modifications to register contents.

SIV - Software Interrupt Violation. Set when unauthorized modification of the SIS register by a BIT instruction is attempted. Authority to change SIS by a bit manipulation instruction is dependent on the EMB and SIP bits of the PL register.

ILS - Illegal Instruction or Unimplemented Memory. Set if an attempt is made to execute an unimplemented instruction or to access memory that is not specified as available by bits 01 through 04 of CPS.

OV = Accumulator Overflow. Set whenever arithmetic overflow occurs. This overflow can be the result of a fixed point add, subtract, or divide; arithmetic left shift, or normalize operation.

FPO - Floating Point Overflow. Set whenever the characteristic of a floating point number exceeds the capacity of the characteristic field during execution of a floating point instruction.

MPF - Main Power Failure. Set if power level drops below the safe limit.

MPE = Memory Parity Error. Set when a word fetched from memory indicates improper parity.

HLT - Halt Instruction Execution. Set when instruction code 00 (HLT) is executed. Use of the HLT instruction should be avoided in on-line programs.

CPS is protected from program manipulation by the RMP bit of the PL register. The information in the CPS register is neither saved nor initialized when an interrupt request is honored.

Software Interrupt Status (SIS Register, 77764, 24 Bits) - The SIS register can generate a software interrupt request on any one of the available levels: 12 basic, 24 with the priority interrupt expander option. Each bit position specifies an interrupt level; however, without the expander option bits 12 through 23 can not generate an interrupt.

The lower the position number of the bit set in SIS, the higher the priority of the interrupt requested. Thus, a program operating at level 05 (IS field of the PL register contains the value 5) could set bit 04 of the SIS register. Setting bit 04 interrupts the level 05 program immediately. The level 04 interrupt handler software provides the service requested, then returns control to the level 05 program. On the other hand, setting bit 06 of the SIS register initiates an interrupt at level 06 only after the level 05 program has been suspended or terminated and any new requests for levels 00 through 05 have been serviced.

Two separate protection privileges are maintained for the SIS register. It is protected from store operations by the RMP bit of the PL register and from bit manipulation (BIT instruction) by the SIP bit of the PL register.

SIS is neither saved nor initialized when an interrupt occurs.

Interrupt Flip-Flops (Not Addressable, 12 or 24 FF's) - Each line in the interrupt bus, a part of the PIO bus, connects to a separate interrupt flip-flop in the central processor. The basic processor has 12 interrupt lines implemented; a processor with the expanded interrupt option has 24 lines. A sample pulse occurring every pulse train (320 nanoseconds) sets each flip-flop to the state of its interrupt line.

Hardware connects each IO device to a single interrupt line corresponding with the desired device interrupt level. With as many as 24 devices sharing the same line, interrupt handler software must use a Read Interrupt Level instruction to determine which of the devices has requested an interrupt.

Mark Register (Not Addressable, 5 Bits) - Three sources of interrupt requests compete for processor time: hardware requests from the CP status register, software requests from the SIS register, and IO requests from the interrupt flip-flops. At the end of each instruction access state, mark level logic compares the priority levels of all pending requests and sets the mark register to the value of the highest active level.

A comparator matches the five bits of the mark register against the current program's interrupt status (IS) stored in the last five bits of the PL register. An interrupt is generated if the number in the mark register is a smaller absolute value (indicating a higher priority) than the number in the IS. Interrupt is generated for requests at levels 00 through 11 for the basic priority interrupt system. Requests at lower levels do not cause any hardware action. With the priority interrupt expander option, interrupt is generated for levels 00 through 23.

Word Switch Register (WSR, 7763, 24 Bits) - The word switch register consists of a row of 24 toggle switches on the control panel. The contents of the WSR are set physically by the operator and cannot be modified by a program. The contents of WSR may be referenced by any instruction that uses 77763 as an effective address, but any attempts to store information into the WSR result in no action and no alarm indication.

## Bus System

Register Busing - Within the CP two word-parallel buses transfer data: the system bus (SB) between core and a selected register, and the central processor bus (CPB) from register to register. Direct parallel transfer from register to register or between register and adder (arithmetic or address) can take place in the same cycle as a bus transfer, permitting a complete exchange or update of register information in 320 nanoseconds.

Information read from core can be transferred over the system bus to three hardware registers -- the B register, the instruction register, and the shift counter. These three non-addressable registers also connect to the central processor bus, which is common to all the addressable registers.

The B register, focal point for all data movement, provides the only direct interface between the processor and IO device synchronizers -- the programmed IO bus (PIO).

## PROCESSOR (CONTROL SECTION)

Elements of the processor control and coordinate system activities such as memory accessing, instruction processing, and priority interrupt handling.

## Memory Control Unit

Memory control logic allows one resource -- a single port core memory -- to be shared by two users -- the processor and the IO master (IOM). Instruction fetch, address modification, and instruction execution logic initiate processor requests for memory cycles. Sequence instruction fetch, input data processing, and output data processing logic initiates IOM requests for memory. User priority must be established, address input provided, and signals synchronized. The processor, IOM, and memory module each have an address register, as shown in Figure 2-2-2, and each has its own sequence control logic. The central processor clock drives both the processor and the IOM sequence logic, but the memory module generates its own internal timing.

Logic Components - The memory control unit, located in the central processor (and shown as blocks in heavy outline in the referenced figure) consists of two major elements, memory address gates, and memory control flip-flops. The memory control flip-flops include priority flip-flops, cycle interlock and control flip-flops, and a 2-stage access enable counter.

Priority Control - Priority checking logic allocates cycles on a first-come, first-served basis, except that when both the processor and the IOM have requests pending at the same time, the IOM is given preference. The priority level activated -- priority 1 (PR1) for the IOM, or priority 2 (PR2) for the processor -- enables the appropriate address register output to pass through the memory address gates to set the address register in the memory module.

Cycle Control - Since the memory module generates its own internal timing pulses, the memory control logic has the additional task of synchronizing the signal exchange. The solution to the problem of synchronizing memory

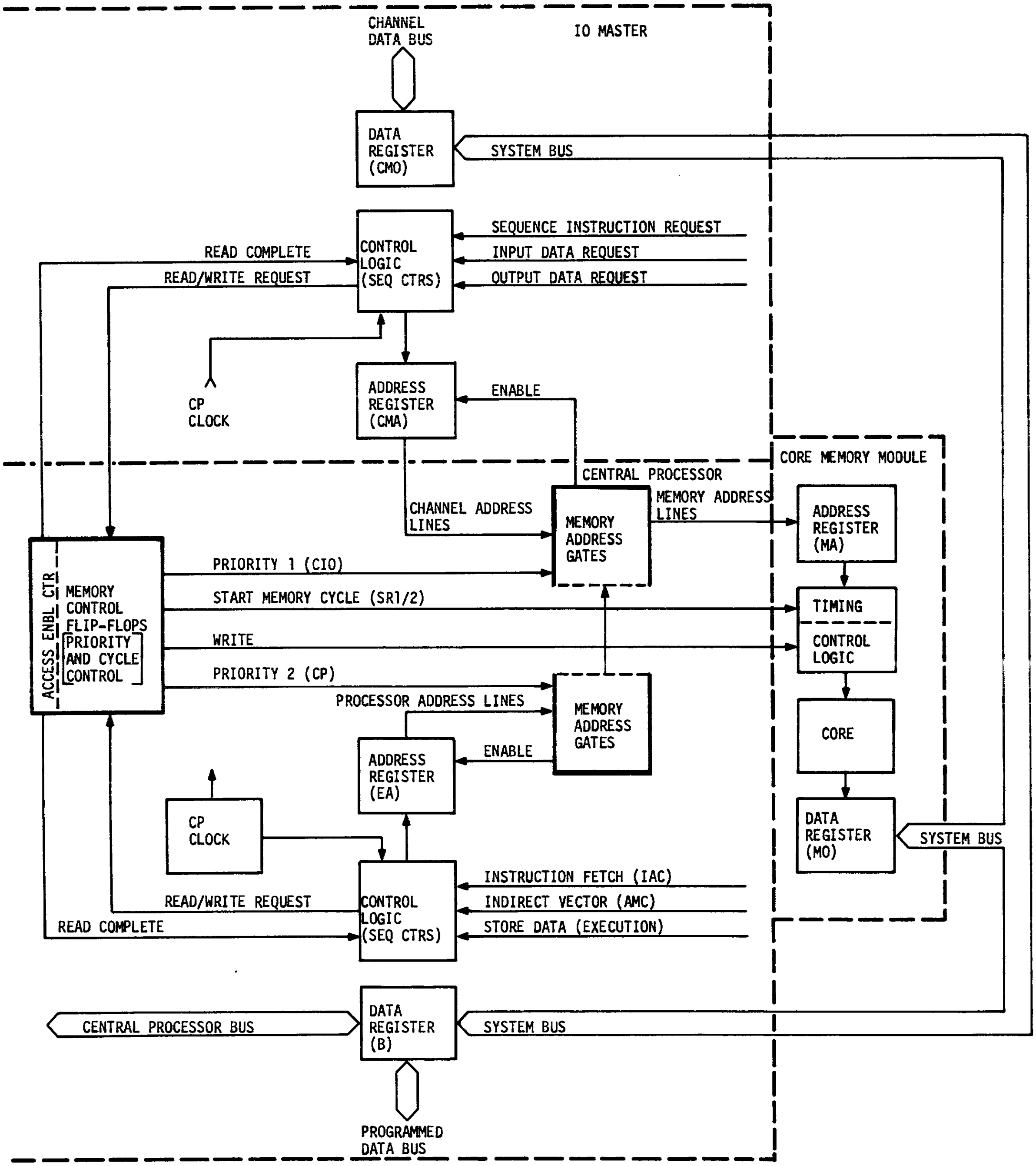


Figure 2-2-2. Memory Control Unit

and the processor or IOM is evolved from the way that the three units are controlled. Memory cycles only on demand, when its controlling delay line is pulsed from an external source. Once started, memory must inevitably run through its full cycle without interruption. The other two units, on the other hand, are more flexible. A continuously running clock in the central processor drives sequence counters in the processor and in the IOM. Counter advance in either unit may be suspended at any point by blocking the clock pulse. Synchronization is achieved in the following way. Through the memory control unit, the processor or IOM provides the external input, start read, to the memory delay line. Sequence counter advance in the requesting unit is suspended until the 2-stage access enable counter in the memory control unit sends a signal, read complete, enabling the sequence to resume.

Within the memory module, a memory cycle always consists of two parts: read (or clear), and restore (or write). If the processor or IOM requests a read operation, the internal functions are read and restore. If the request is for a write operation, the internal functions are clear and write. In either case, the data access (system bus transfer between the user's data register and the data register in the memory module) occurs during the first, or "read" half of the cycle. With the address and data stored internally, the memory module carries out the second half of the operation independently, permitting the processor or IOM to resume its interrupted sequence. Thus, the signal that releases the user is called "read complete" regardless of the operation requested.

## Major State Control

<u>Instruction Access</u> - The processor progresses through three major states for each instruction processed: instruction access, address modification, and instruction execution. The instruction access state can be entered manually from the processor control panel, or automatically at

the end of each instruction execution. Instruction processing is diagrammed in Figure 2-2-3. The program counter provides the address used to read the instruction word from memory in the instruction access state. The memory control logic gates the instruction word into the instruction register and signals read complete (RC) to advance the processor to the address modification state.

Address Modification - The mode (M) portion of the instruction register governs the events that take place during the address modification state. In the direct modes, the address register (AR) portion of the instruction register is used without change (except to propagate bit 09 for class 1 literals in mode 2). Relative modes add a base value from one of the address modification registers to the AR in the address adder, and, if required, add a value from one of the index registers to the intermediate address computed. At the end of the state, the B register contains an operand (for those instructions with MOA = 1) and the EA register contains the final effective address. The program counter is automatically incremented by one, and one of the eight execution control counters is selected.

Instruction Execution - At the end of address modification (provided that no program interrupt is generated), a signal called "set K logic 1" (SKL1) enables the gates to the first stage of each execution (K logic) counter. Lines decoding the operation in the D register (bits 00-05 of the instruction register) provide the additional input necessary to select one of the eight counters.

Execution Sequence States - The number of sequence state outputs available from an execution counter ranges from 3 to 30. Counters may advance one state at a time, jump to a nonsequential state, or loop through two or more states until some desired limit is reached. When a

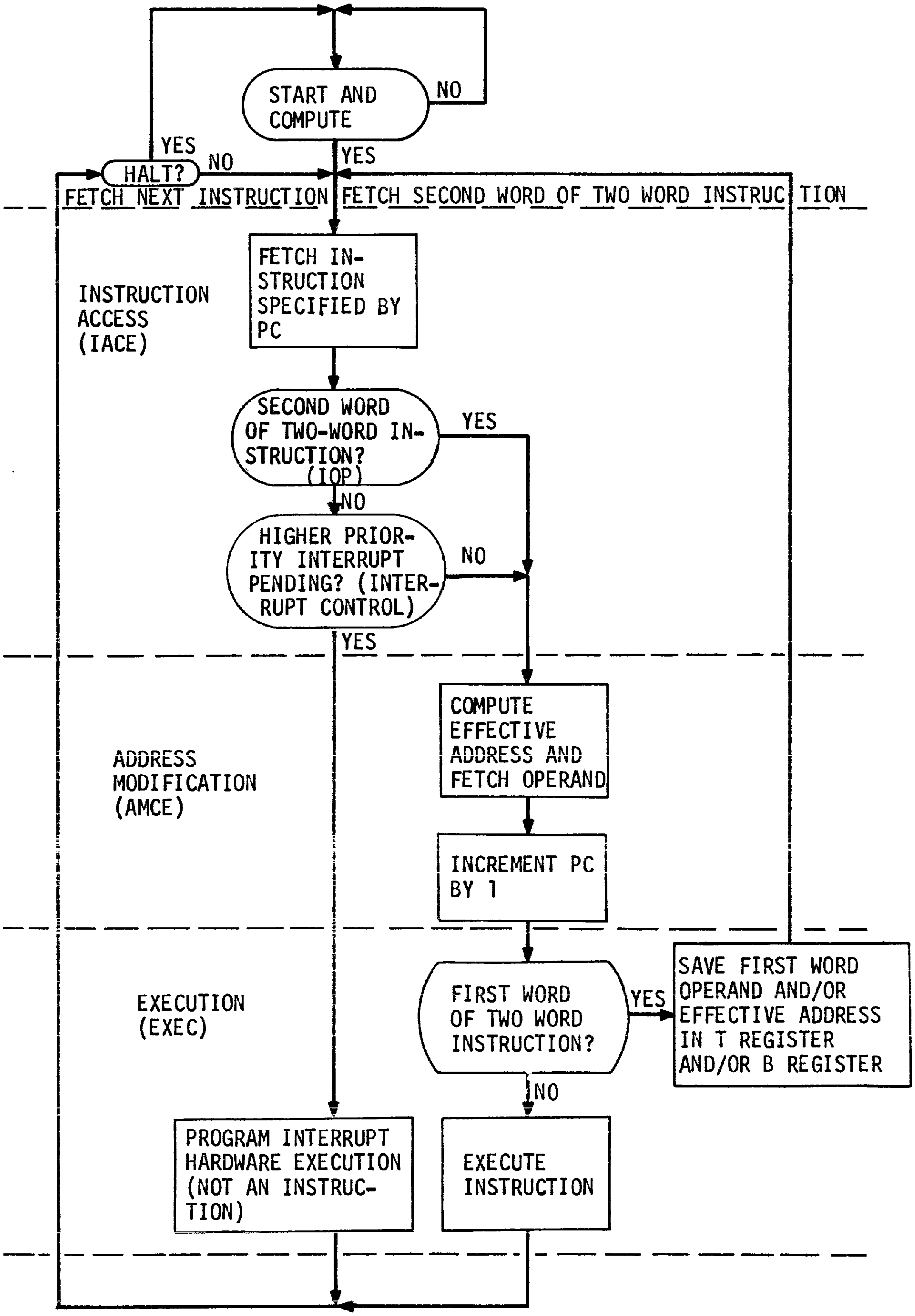

START AND COMPUTE
NO
YES
HALT?
YES
NO
FETCH NEXT INSTRUCTION
FETCH SECOND WORD OF TWO WORD INSTRUCTION
INSTRUCTION ACCESS (IACE)
FETCH INSTRUCTION SPECIFIED BY PC
SECOND WORD OF TWO-WORD INSTRUCTION? (IOP)
YES
NO
HIGHER PRIORITY INTERRUPT PENDING? (INTERRUPT CONTROL)
NO
YES
ADDRESS MODIFICATION (AMCE)
COMPUTE EFFECTIVE ADDRESS AND FETCH OPERAND
INCREMENT PC BY 1
EXECUTION (EXEC)
FIRST WORD OF TWO WORD INSTRUCTION?
YES
SAVE FIRST WORD OPERAND AND/OR EFFECTIVE ADDRESS IN T REGISTER AND/OR B REGISTER
NO
PROGRAM INTERRUPT HARDWARE EXECUTION (NOT AN INSTRUCTION)
EXECUTE INSTRUCTION

Figure 2-2-3. Major State Control

particular state requires memory data, counter advance is suspended until the memory access is completed. The decoded D register output gates the signals generated in each sequence state (allowing only those appropriate to the particular instruction), controls the sequence counter advance (or jump), and determines the state and conditions under which an END signal may be generated.

Major State Loop - The final execution step of any instruction generates an END signal and transfers the contents of the program counter (PC) to the effective address register (EA). The END signal (provided that the halt flip-flop is reset) sets instruction access enable (IACE) to fetch the next instruction from the memory location specified in EA. Thus the three major states form a closed loop.

Two-Word Instructions - Four instructions require a second instruction word from memory: Compare With Memory, Move Multiple, Bit, and Branch and Save Region. Branch and Save Region remains in the execution state while obtaining the second instruction word. The other three instructions remain in the execution state for one or two pulse trains to save the first word operand or effective address in the B or T registers, then revert to the instruction access state to read the second word. Since the operation code must be preserved in the D portion of the instruction register (IR), a signal called inhibit operation code (IOP) is developed for the second word only, causing the memory control logic to place the first six bits of the second word in the shift counter. The remainder of the second instruction word is read into the M and AR portions of the instruction register. The address modification state proceeds in exactly the same manner as for the first instruction word, ending by incrementing the program counter. When the processor enters the execution state the second time, it has all the necessary parameters to perform the instruction.

System Control

Program Interrupt - There are two conditions (shown in the previously referenced Figure 2-2-3) that can break the major state loop. The first condition, program interrupt, is tested at the end of the instruction access state (except during the access of the second word of a two-word instruction). If the interrupt request level in the mark register is higher than the interrupt status (IS) of the active program, the instruction access control logic generates the signal "set K logic 2" (SKL2) and sets the program interrupt flip-flop (PI). SKL2 bypasses the address modification state and selects the CG execution counter. The PI flip-flop input to the CG counter gates the signals to execute the hardware interrupt action in the same way that decode lines from the D register gate the execution of program instructions. The PI execution ends after the contents of the ten highest numbered addressable registers have been stored in memory and the first six of these registers provided with new values. (More information on program interrupt is presented in Sections 4 and 5.) Since the program counter (PC) is one of the initialized registers, when the END signal is generated, the first instruction of the interrupt handler program will be fetched in the ensuing instruction access state, instead of the next sequential instruction.

Programmed Halt - The second condition that can break the major state loop is the setting of the halt flip-flop. One way to set the halt flip-flop is to execute a Halt instruction in a program with an interrupt status of 00. With the halt flip-flop set, the END signal is prevented from setting instruction access enable.

Compute Operational State - As long as the central processor control panel MODE SELECT switch is in the ON-LINE or the OFF-LINE position and the computer is fetching instructions from memory and executing them, it is in the compute operational state. This is one of six possible states generated by a 3-stage operational state counter working in conjunction with the MODE SELECT switch. The names and octal designations of the operational state counter outputs are listed in Table 2-2-4.

Table 2-2-4. Operational States

| STATE | NAME | COMMENTS |
|---|---|---|
| 0 | Halt | No instructions executed. Control panel can display or store hardware register data. |
| 2 | Compute | System is running, either on-line or off-line. |
| 3 | WSR Loop | The processor executes an instruction coded in the switches of the word switch register. |
| 4<br>6<br>7 | Display Memory<br>Store Memory<br>Sequential Store Memory | Manual store or display of core data is provided by manual mode memory control logic (MMM1, MMM2, and MMM3). |

Halt Operational State - When a programmed Halt at level 00 is encountered, the operational state counter is cleared at the end of the execution state, producing the halt operational state. The same result may be obtained by pressing the HALT button on the CP control panel. Once the computer is in the halt state, operator intervention is required to attain the compute state again (except that following a power failure, an automatic restart signal initiates the compute state.)

WSR Loop State - The WSR LOOP state differs from the compute state in that a single instruction word is read from the word switch register (WSR) during each instruction access state, in place of the memory instruction word. The address modification and execution states are normal, except that at the end of execution, the signal to transfer PC to EA is blocked, and the address of the WSR is transferred to EA, causing the contents of the WSR to be read into the instruction register during instruction access.

Manual Memory Control Operational States - Three operational states provide for manual control of memory data: display, store, and sequential store. A 2-stage memory manual mode counter (analogous to an instruction execution counter) governs memory cycles for all of the memory control operational states.

(More information on the central processor control panel is presented in Section 6.)

## PROGRAMMED INPUT/OUTPUT

Two data paths are available for transmitting digital information between the central processor and IO devices; the programmed IO path (PIO) and the channel IO path (CIO). The PIO route is taken for single-word data transmission and command/control/status transmissions between the CP and a device. Block-data transmissions are initialized through the PIO, initiated by the device request, and executed through the CIO on a cycle-stealing basis by direct memory access.

PIO Component Hardware

The hardware in the processor for implementing programmed IO includes the following:

a. CC Execution Counter -- governs the PIO command sequence and the data transfer sequence.

b. B Register -- through the programmed IO bus links the processor with control, data, and status registers in all IO device synchronizers, as shown in Figure 2-2-4.

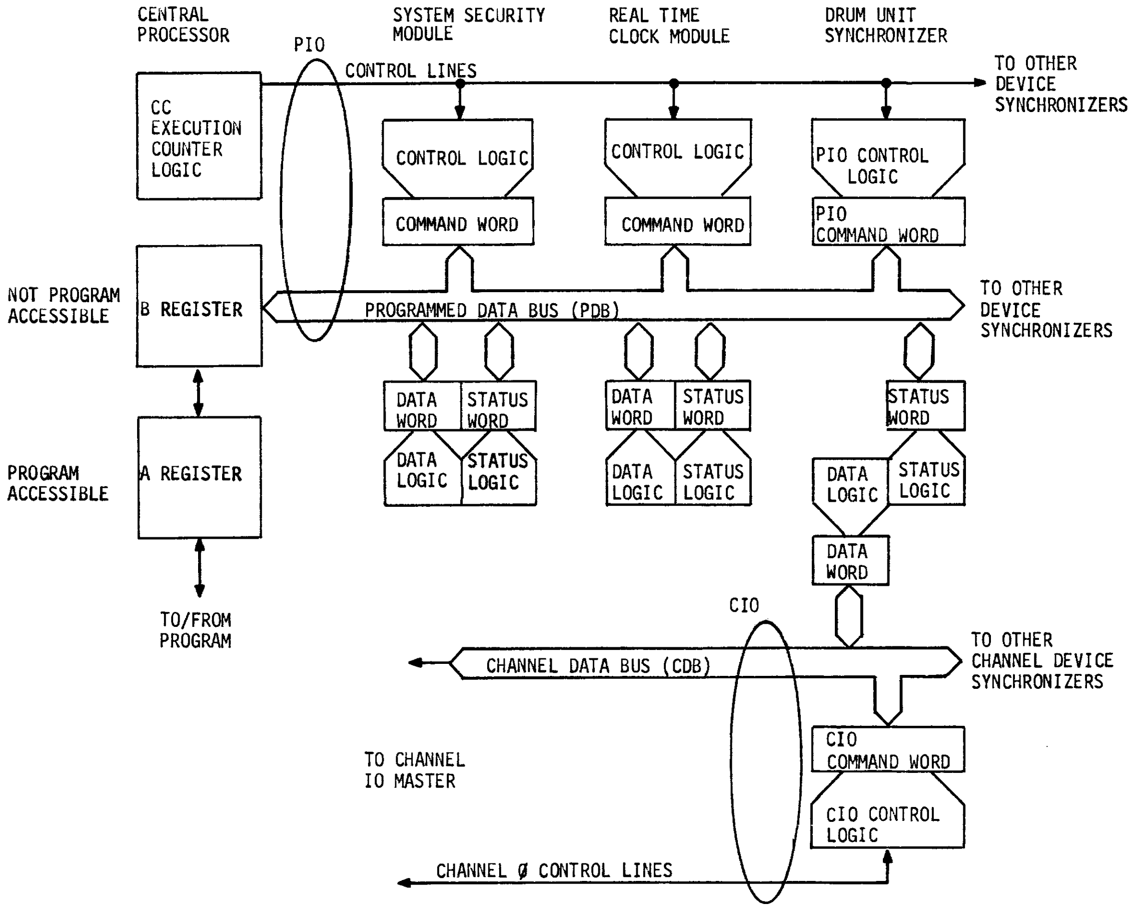


Figure 2-2-4. PIO Data and Control Paths

c. A Register -- as an addressable register provides the data interface between the program and the non-addressable B register.

d. Programmed IO Bus consisting of

(1) Programmed Data Bus (PDB) -- 24 bidirectional lines for the transmission of commands to all devices, status information to and from all devices, and data between the processor and PIO (non-channel IO) devices.

(2) PIO Control Lines -- processor initiated lines indicate the type of information on the PDB and the direction of transfer; synchronizer initiated lines indicate receipt of a command or data/status word, or that an input data/status word from the synchronizer is on the PDB. The PIO control lines are described individually in Table 2-2-5.

(3) Interrupt Bus -- 24 lines for the transfer of device interrupt requests to the CP interrupt matrix (one line per interrupt level).

General Description

Instruction access and address modification hardware fetches and decodes the PIO instruction word and obtains or develops the PIO operand or command word. If the literal addressing mode is used, the command word is formed by moving the 15 low order bits of the instruction word into the B register in the format shown in Figure 2-2-5. In any other mode of addressing, an operand is fetched from memory as in Figure 2-2-6.

Table 2-2-5. PIO Control Lines

| MNEMONIC | SIGNAL FUNCTION | SIGNAL SOURCE |
|---|---|---|
| MCL | Master Clear. Clears all device synchronizers at once. The signal on this line is generated from the CLEAR switch of the CP control panel or automatic restart logic. | CP Control Panel or CP |
| COMO | Command Out. Signals the Device synchronizer to prepare to receive a command. This signal is generated by execution of the PIO instruction. | CP |
| DTIR | Data Input Request. Signals the synchronizer to prepare the selected device to input information. This signal is generated from the PIO operand (bit 13 = 1 and bit 14 = 1). | CP |
| DTOA | Data Output Available. Signals the synchronizer that information in the A register is ready to be transmitted. This signal is generated from the PIO operand (bit 13 = 0 and bit 14 = 1). | CP |
| ACK | Acknowledge. This signal is a device response to COMO, DTIR, or DTOA. ACK -- not accompanied by REJ -- initiates a skip if the PIO operand has bit 15 set. PIO logic generates ACK automatically for RILS, since this command is never rejected. | Device Synchronizer |
| REJ | Reject. This signal is sent with the ACK signal to reject a COMO that cannot be executed immediately. REJ prevents skipping of the next instruction following PIO. | Device Synchronizer |
| BOOT | Read Bootstrap. This signal is sent to the memory drum to initiate the bootstrap loader function. BOOT is generated from the LOAD BOOTS switch on the CP control panel. | CP Control Panel |
| WRBT | Write Bootstrap. This signal is used by the drum to bypass the write protection on track 0. WRBT is generated from the WRITE BOOTS switches on the CP control panel. | CP Control Panel |
| OFF | Off-line. This signal, generated when the control panel MODE SELECT switch is in the OFF-LINE position, inhibits interrupts from process IO devices and consoles. | CP Control Panel |

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PIO OPERATION CODE | | | | | | MODE | | | COMMAND | | | | | | SKIP | DEVICE ADDRESS | | | | | | | |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | | | | | | | | | | | | | | | |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

INSTRUCTION WORD, LITERAL MODE

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | COMMAND (UP TO 6 BITS) | | | | | | SKIP | DEVICE ADDRESS (8 BITS) | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | | | | | | | | | |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

LITERAL OPERAND[1]

[1] Bit 09 is left-propogated in generating literal operand.

Figure 2-2-5. PIO Instruction and Operand Format, Literal Mode

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PIO OPERATION CODE | | | | | | MODE | | | ADDRESS OF MEMORY OPERAND | | | | | | | | | | | | | | |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | | | | | | | | | | | | | | | |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

INSTRUCTION WORD, NON-LITERAL MODE[1]

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| COMMAND[2] (UP TO 15 BITS) | | | | | | | | | | | | | | | SKIP | DEVICE ADDRESS (8 BITS) | | | | | | | |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

MEMORY OPERAND

[1] Any mode except 2. Mode 4 is shown in figure.

[2] Non-literal addressing must be used for devices that require a command field of more than six bits. For example, process IO synchronizers use part of the command field to provide an additional level of addressing to select individual multiplexers.

Figure 2-2-6. PIO Instruction and Operand Format, Non-Literal Mode

Command Sequence (Figure 2-2-7) - The execution of any PIO instruction starts with a command word (PIO operand) in the B register ready for transmission to all device synchronizers. The PIO command word includes one or two address fields to select one specific device, or in the case of a Read Interrupt Level instruction, all devices sharing the same interrupt level. The addressed synchronizer stores the command word and sends an acknowledge signal.

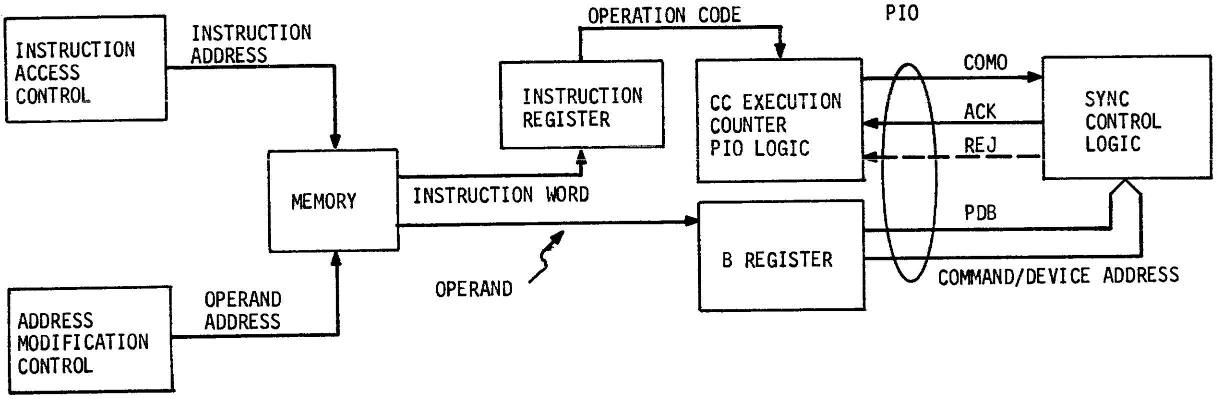


Figure 2-2-7. PIO Command Block Diagram

Data Transfer Sequence (Figure 2-2-8) - Data transfer is bidirectional. In a write sequence, the contents of the A register, placed there by a previous instruction, are transferred to the B register, from the B register to the PDB, and from the PDB to the synchronizer data or status register. In a read sequence, the order and direction is reversed: data are transferred from the PDB to the B register, and from the B register to the A register. In either case, an acknowledge signal synchronizes the exchange.

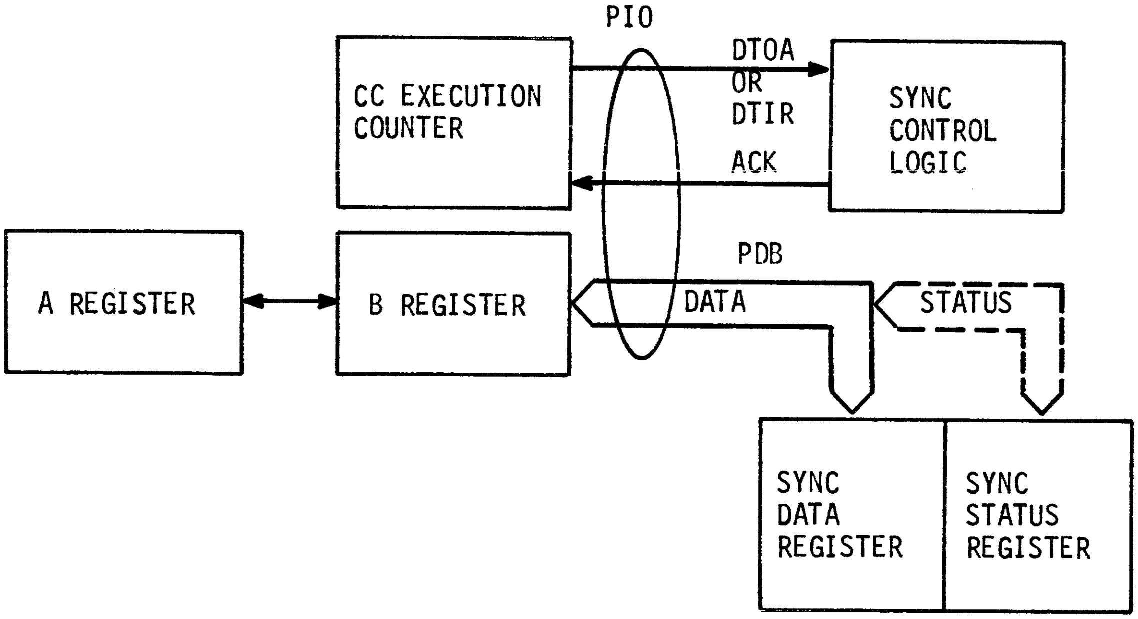


Figure 2-2-8. PIO Data Transfer Block Diagram

Note that only one word is transferred by a single PIO instruction and that the word remains in the A register. If multiple words are to be transferred, other instructions must transfer the IO data between the A register and core before another PIO instruction can be executed.

PIO Commands - There are five categories of defined PIO instructions, based on the type of information transferred, as follows:

a. Data Transfer Commands
b. Status Commands
c. Read Interrupt Level Command
d. Start Channel IO Command
e. No Operation Command

The first three commands transfer data or status information, as summarized in Figure 2-2-9; the fourth merely transfers a command. This signals the device synchronizer to contact the IO master (by a sequence instruction request, SIR) to obtain an IO command. The last command listed requires only an acknowledge signal from the synchronizer. A device responds to every command sent to it. Undefined commands are treated as no-operations. Every no-operation generates an acknowledge signal from the device; none is rejected.

IO Privilege - Every IO instruction, including the Read Interrupt Level instruction, requires one of the IO privileges provided by bits 03, 04, and 05 of the privilege and level register. The specific privilege required depends upon the address of the device being used. The devices requiring the lowest privilege, the non-critical IO privilege (NIOP), are those with addresses from 000 through 127 (000 through 177, octal). The next level, critical IO privilege (CIOP), which also encompasses the NIOP group, permits the use of any device except those in the address range of 160 through 167 (240 through 247, octal). The highest level, system IO privilege (SIOP), enables a program to use any IO device in the system. All IO device addresses presently assigned are listed in Table 2-2-6; unassigned addresses are listed in Table 2-2-7.

Detailed Sequence Description

When a PIO instruction is fetched the CP obtains the operand for the instruction and executes the following sequence of operations:

a. Check for IO usage violation.

   This check compares the IO privilege levels (SIOP, CIOP, NIOP; bits 03, 04, and 05 of the privilege and level register) to

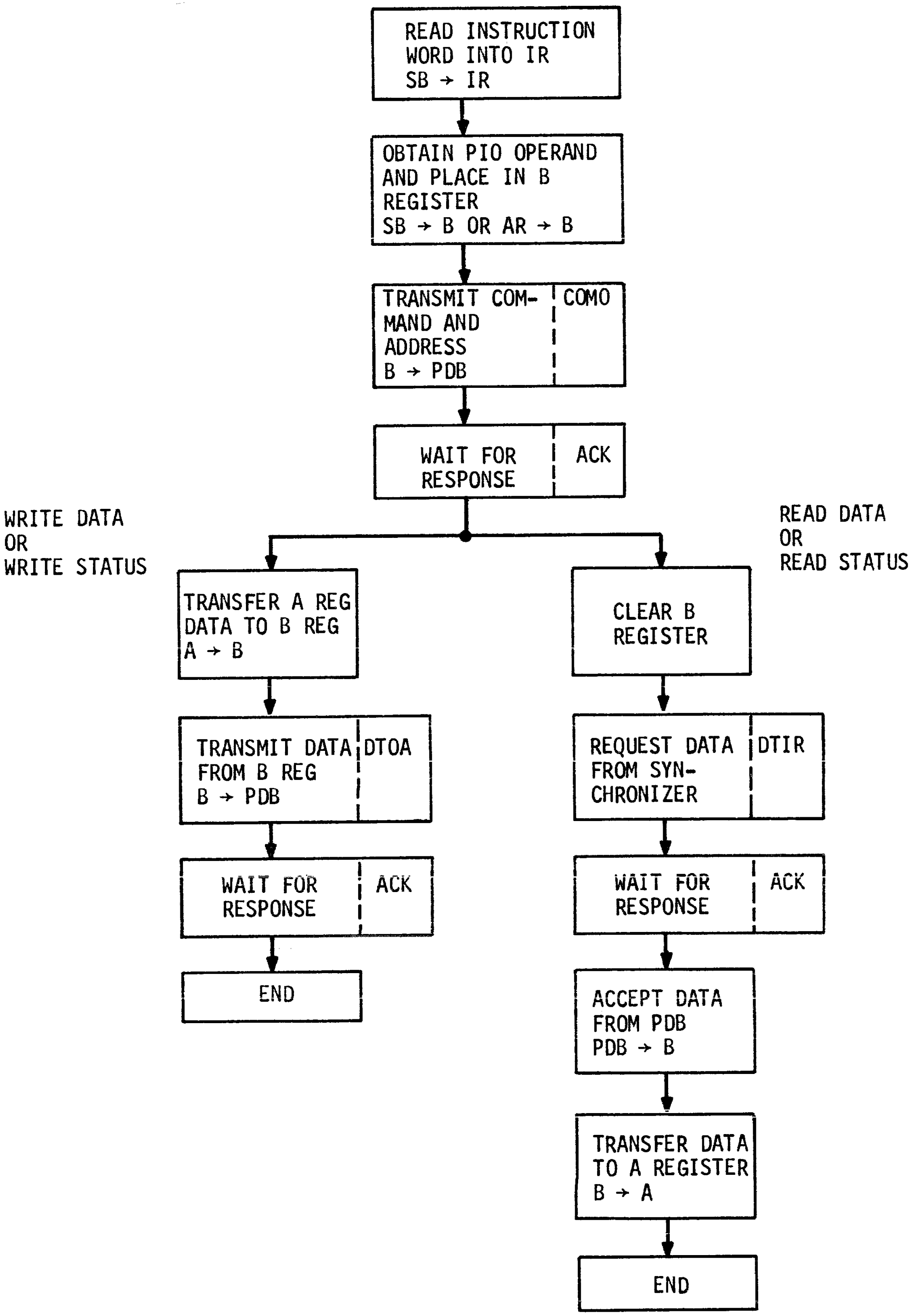


Figure 2-2-9. PIO Read/Write Sequence

Table 2-2-6. Standard IO Device Address Assignments

| DEVICE ADDRESS | | PRIVILEGE REQUIRED | ASSEMBLER MNEMONIC | DEVICE |
|---|---|---|---|---|
| DECIMAL | OCTAL | | | |
| 000 | 000 | NIOP | RILS 00 | Interrupt Level 00 |
| 001 | 001 | ▲ | RILS 01 | Interrupt Level 01 |
| • | • | | • | • |
| • | • | | • | • |
| • | • | | • | • |
| 027 | 023 | | RILS 23 | Interrupt Level 23 |
| 032 | 040 | | PTR | Paper Tape Reader |
| 033 | 041 | | PTP | Paper Tape Punch |
| 034 | 042 | | TTY | Teletype |
| 035 | 043 | | CARD | Card Reader/Punch |
| 036 | 044 | | PRINT | Line Printer |
| 064 | 100 | | CLK 2 | Real Time Clock 2 |
| 065 | 101 | | CLK 3 | Real Time Clock 3 |
| 096 | 140 | | TYPR 0 | Typer 0 |
| 097 | 141 | | TYPR 1 | Typer 1 |
| • | • | | • | • |
| • | • | | • | • |
| • | • | | • | • |
| 104 | 150 | ▼ | TYPR 8 | Typer 8 |
| 120 | 170 | NIOP | DION | Digital IO Unit, Noncritical |
| 128 | 200 | CIOP | CNS 0 | Console 0 |
| 129 | 201 | ▲ | CNS 1 | Console 1 |
| • | • | | • | • |
| • | • | | • | • |
| • | • | | • | • |
| 135 | 207 | ▼ | CNS 7 | Console 7 |
| 136 | 210 | CIOP | CNSYN | Console Synchronizer |
| 160 | 240 | SIOP | DRUM | Drum Memory |
| 162 | 242 | ▲ | SSEC | System Security Module |
| 164 | 244 | | DISK 0 | Disk Memory Unit 0 |
| 165 | 245 | | DISK 1 | Disk Memory Unit 1 |
| 166 | 246 | ▼ | DISK 2 | Disk Memory Unit 2 |
| 167 | 247 | SIOP | DISK 3 | Disk Memory Unit 3 |
| 192 | 300 | CIOP | CLK 0 | Real Time Clock 0 |
| 193 | 301 | ▲ | CLK 1 | Real Time Clock 1 |
| 208 | 320 | | CONTRL | Controller Communication Module (753) |
| 216 | 330 | | VALCON | Valve Control Output Module |
| 224 | 340 | ▼ | ANI | Analog Input Unit |
| 248 | 370 | CIOP | DIOC | Digital IO Unit, Critical |

Table 2-2-7. Unassigned Standard IO Device Addresses

| DEVICE ADDRESSES | | PRIVILEGE REQUIRED |
|---|---|---|
| DECIMAL | OCTAL | |
| 028 through 031 | 024 through 037 | NIOP |
| 037 through 063 | 045 through 077 | NIOP |
| 066 through 095 | 102 through 137 | NIOP |
| 105 through 119 | 151 through 167 | NIOP |
| 121 through 127 | 171 through 177 | NIOP |
| 137 through 159 | 211 through 237 | CIOP |
| 161 | 241 | SIOP |
| 163 | 243 | SIOP |
| 168 through 191 | 250 through 277 | CIOP |
| 194 through 207 | 302 through 317 | CIOP |
| 209 through 215 | 321 through 327 | CIOP |
| 217 through 223 | 331 through 337 | CIOP |
| 225 through 247 | 341 through 367 | CIOP |
| 249 through 255 | 371 through 377 | CIOP |

the device address field of the PIO operand. A violation sets the IO usage bit (IOU, bit 13 of the CPS register), which produces a level 01 interrupt request, and terminates the PIO instruction.

b. Transmit Command Word.

If no usage violation is indicated, the execution control logic places the command word (PIO operand) on the PDB, sends a COMO signal to the specified device, and starts the IO timer. The IO timer allows approximately 12 microseconds for the device to return an acknowledge (ACK) signal. Exceeding the time limit sets the IO timer bit (IOT, bit 12 of the CPS register), which produces a level 01 interrupt request, and terminates the PIO instruction. ACK accompanied by a reject (REF) signal (usually because the device or its synchronizer is busy) also terminates the PIO execution, but without setting any violation. The program resumes with the next instruction in sequence when a busy condition is found.

If the command is accepted (ACK and not REJ) and skip (bit 15 of the command word) is set, the contents of the PC register are incremented to omit the next instruction after the PIO. (With skip reset, PC is not altered.) The accumulator usage bit (ACU) indicates whether a data transfer is required. With ACU reset, the PIO operation is complete at this point, and the appropriate program instruction is fetched.

c. Transmit or Receive Data or Status Word.

If the ACU is set, the data transfer in (DTI, bit 13 of the PIO operand) is examined. With DTI set, the execution control logic transmits a data transfer in requested (DTIR) signal to the device. ACK from the device gates the information (data or status) on the PDB into the B register, and from the B to the A register.

With ACU set and DTI reset, the execution control logic transfers information (data or status) from the A register to the B register, gates the B register contents onto the PDB, and transmits a data output available (DTOA) signal to the device. The information is kept on the PDB until an ACK signal indicates the device has received it.

The IO timer allows approximately 12 microseconds for the device to return an ACK signal for the PDB data transfer in either direction. Exceeding the time limit ends the instruction and sets the IOT.

The sequence of PIO events is summarized in Table 2-2-8.

## CHANNEL INPUT/OUTPUT

The CIO path between the memory and an IO device is direct, except for the intervening input output master (IOM) electronics. The IOM multiplexes a single bidirectional memory path for use by (up to) eight IO devices and controls all transfers through this path. Access to memory by the eight devices is granted on a priority schedule assigned by hardware at the time the device is installed in the system. Priority assignments are numbered from 0 through 7, with 0 having the highest priority.

The eight channels are divided into two high-speed channels having priorities 0 and 1, and six low-speed channels having priorities 2 through 7. The high-speed channels, used by the drum and disk, contain

Table 2-2-8. Summary of PIO Operations

| COMMAND | SITUATION | DATA TRANSFERRED? | PIO SEQUENCE OF SIGNALS | COMMENTS |
|---|---|---|---|---|
| Read Data or Read Status (RDA/RST) | Normal | Yes | 1. COMO<br>2. ACK<br>3. DTIR<br>4. ACK | Skip initiated if command had skip bit set. |
| Write Data or Write Status (WDA/WST) | Normal | Yes | 1. COMO<br>2. ACK<br>3. DTOA<br>4. ACK | Skip initiated if command had skip bit set. |
| No Operation (NOP) | Normal | No | 1. COMO<br>2. ACK | Instruction complete in one sequence. |
| Start Channel IO (CIO) | Normal | No | 1. COMO<br>2. ACK | Flag set in device synchronizer to initiate a sequence instruction request (SIR) to the IO master. |
| Read or Write Data, Read Status and Clear, or Start CIO | Busy | No | 1. COMO<br>2. a. ACK<br>b. REJ | No skip initiated on command with skip bit set. |
| Any Command | No Response | No | COMO | IO timeout (IOT) violation set after 12 microseconds and command aborted. |

hardware registers to control the memory address and number of words in the transfer, and transfer one word for each CP cycle stolen. Low-speed channels, for devices such as the card punch and line printer, use core memory registers to control the memory address and size of the transfer, and transfer one word for each four CP cycles stolen. Dedicated memory in locations 00032 through 00059 (excluding 00040 and 00041) are reserved for the data address register (low-speed channels only; in locations 00032 through 00037), last address register (low-speed channels only; in locations 00042 through 00047), and next instruction address registers (all channels; locations 00050 through 00057). Refer to the previous Figure 2-2-1 for dedicated memory locations related to CIO operation.

CIO Component Hardware

The hardware for implementing channel IO includes the following:

a. IO master state and sequence counters
b. Common registers and flip-flops
c. Dedicated channel registers and flip-flops
d. CIO bus

The CIO counters, registers, and flip-flops are listed in Table 2-2-9; the CIO bus lines are listed in Table 2-2-10; and their interconnection with memory and the IO device synchronizer is shown (in simplified form) in Figure 2-2-10.

General Functional Description

Up to eight channels may be processing data concurrently, but memory, the IO master, and the channel data bus can service only one channel at a time. The IO master (IOM) monitors channel activity and establishes priority through a set of three request flip-flops, sequence instruction

Table 2-2-9. IO Master and Channel Logic Components

<table>
<tr><th>IO MASTER<br>HARDWARE REGISTERS</th><th>HIGH SPEED CHANNEL<br>HARDWARE REGISTERS<br>2 EACH TYPE</th><th>LOW SPEED CHANNEL<br>CORE REGISTERS<br>6 EACH TYPE</th></tr>
<tr><td>Channel Memory Address (CMA)<br>Channel Memory Output (CMO)<br>Auxiliary Register (AUX)</td><td>Data Address Register (DAR)<br>IO Word Counter (CNT)</td><td>Data Address Register (DAR)<br>Last Data Address Register (LDAR)</td></tr>
<tr><th>IO MASTER<br>CONTROL COUNTERS</th><th colspan="2">CORE REGISTERS COMMON TO ALL CHANNELS<br>8 EACH</th></tr>
<tr><td>Operation Counter (OP), 3-Stage<br>Sequence Counter (SQ), 5-Stage</td><td colspan="2">Next Instruction Address Register (NXAR)</td></tr>
<tr><th>CHANNEL MEMORY CYCLE<br>CONTROL FLIP-FLOPS</th><th>CHANNEL ACTIVITY REQUEST<br>CONTROL FLIP-FLOPS<br>8 EACH</th><th>CHANNEL DATA BUS<br>CONTROL FLIP-FLOPS</th></tr>
<tr><td>Memory Request (MRQ)<br>Memory Write Flip-Flop (MWRF)</td><td>Sequence Instruction Request (SIR)<br>Input Data Request (IDR)<br>Output Data Request (ODR)</td><td>Last Word Level Flip-Flop (LWLF)<br>Enable Bus Data Flip-Flip (ENBDF)<br>Strobe Output (STBO)</td></tr>
</table>

Table 2-2-10. Channel IO Bus (59 Lines)

| NAME | DEFINITION | FLOW* | LINES |
|---|---|---|---|
| LWL- | Last word level. Channel detected the end of a block transfer. LWL is transmitted along with the last data word and sampled by the device synchronizer at the trailing edge of ENB. | C → S | 1 |
| PER- | Parity error. During a core cycle stolen for SIR, IDR, or ODR, channel detected a real parity error or an attempted access to nonexistent memory. PER is sampled by the device synchronizer at the trailing edge of ENB. | C → S | 1 |
| CDBn-<br>00-23 | Channel data bus. In response to SIR; carries the first word of the COTW; in response to IDR or ODR, transmits data between the CIO word register and the device synchronizers. | C → S<br>S → C<br>C → S | 24 |
| SIRn-<br>Ø-7 | Sequence instruction request. Sets up the CIO for a subsequent input or output data transmission. | S → C | 8 |
| ODRn-<br>Ø-7 | Output data request. Signals the channel to transmit a word to the device synchronizer. The channel responds with an enable (ENB) signal, resetting ODR until the synchronizer is ready for the next word of a multiword output data transfer. | S → C | 8 |
| IDRn-<br>Ø-7 | Input data request. Signals the channel to receive a data word on the CDB into its word register. | S → C | 8 |
| ENBn-<br>Ø-7 | Enable. Allocates CDB usage to the device synchronizer sending SIR, ODR, or IDR. Generates STBO for an SIR or ODR, and provides a request for data in response to IDR. ENB permits the synchronizer to reset its request. | C → S | 8 |
| STBO- | Strobe output. Following an ODR or IDR, signals the device synchronizer that the data on the CDB is now valid; not used for IDR. | C → S | 1 |

*C = Channel
S = Device synchronizer

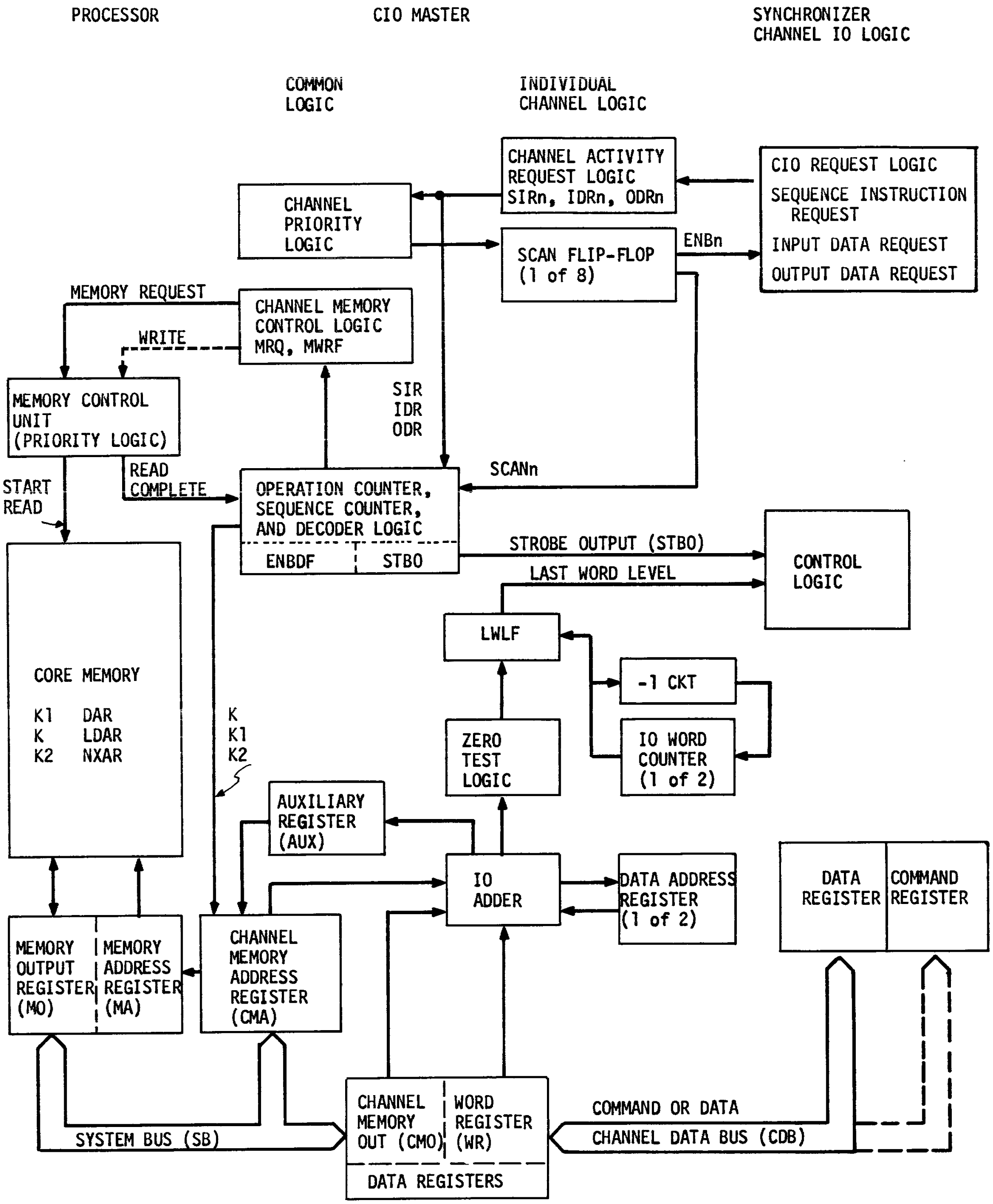


Figure 2-2-10. CIO Data and Control Paths

request (SIR), input data request (IDR), and output data request (ODR), and one scan flip-flop (SCN) provided for each of the eight channels. Any CIO device synchronizer can set one of the request flip-flops for its assigned channel at any time. The request sets the channel's scan flip-flop only when no lower numbered channel has a request pending. For example, channel 0 would be serviced before channel 1. The accepted channel (the one whose scan flip-flop is set) has exclusive use of the IOM until its sequence instruction request or data transfer request (one word of data) has been completed.

One set of address and data registers and control flip-flops in the IOM provides the interface between the core memory module and the device synchronizers for all eight channels. However, with the channels taking turns, each must have its own set of registers and flip-flops to store operating parameters during the time that other channels are being serviced. Every time that a data word for channel 0 is to be transferred, the address register for channel zero must be transferred to the IOM's address register. The IOM updates the register count of words transferred and detects when the last word has been sent. At the end of each single word transfer the updated control information is returned to the channel registers for storage until the next word is requested. The IO device sets the interval between data transfer cycles (subject to some delay if a higher priority device pre-empts the IOM).

The CP initiates all CIO operations by issuing a Start Channel IO command over the PIO bus to a device synchronizer. The synchronizer responds by requesting the IOM to read a 3-word instruction (channel order) from memory. The IOM obtains and sends the first of 3-words to the synchronizer and stores the remaining two words in the data address and word count (or last data address) registers reserved to that channel. A single channel order might, for example, control the transfer of 1,024 words from the drum to memory. For each word transferred, address and count information must be shuttled between the dedicated channel registers and the common registers in the IOM.

CIO Command Format - Commands that guide channel IO operation sequence are provided as a three-word or four-word packet called the channel order triple word (COTW). The fourth word is required only if chaining is specified. Collectively, the COTW is called a CIO sequence instruction. The basic formats of these words are:

Word 1

| OUT | CHR | EOI | NOP | DEVICE CONTROL | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Word 2

| IGNORED | | | | | | | | | | | | WORD COUNT | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Word 3

| IGNORED | | | | | | | | | FIRST WORD ADDRESS | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Word 4 (Optional)

| IGNORED | | | | | | | | | CHAINING LINK ADDRESS | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Word 1 is the channel order. The remaining three words provide sequencing information.

Field — Definition

OUT

Data Transfer Direction. This bit designates whether data are to be transmitted to the device (OUT = 1) or if data are to be transmitted by the device (OUT = 0).

CHR

Chaining Request. This bit designates whether data chaining is to be used (CHR = 1) or if activity should end with the operation specified by this COTW (CHR = 0).

EOI

End Order Interrupt. This bit specifies whether interrupt should occur when CIO ends for this order (EOI = 1). If EOI = 0, only error interrupt may occur.

NOP

No Operation. This bit designates that no data are to be transferred (NOP = 1). If NOP = 0, data are to be transferred.

Device Control

Information in this field depends upon the individual devices.

Word Count

This field designates the number of words in a data transfer.

First Word Address — This field specifies the address of the first memory word from which data are to be output or to which data are to be input.

Chaining Link — This address specifies the address of the next four-word-sequence instructions. If CHR = 0, indicating no chaining, this address need not be a part of the COTW packet.

The COTWs are made available to the IO master by use of the next instruction address registers (NXAR, locations 00050 through 00057 of dedicated memory). The address in NXAR is used by channel IO as an indirect address that points to a memory word containing the address of the instruction packet.

It is the responsibility of a channel IO requesting program to initialize NXAR prior to initiating a channel IO operation.

Because both the high-speed and low-speed channels use the same COTW format, any device that normally functions on a high-speed channel will work on a low-speed channel and vice versa.

Example:

Assume that output of two blocks of data to a channel IO device is required. The first block consists of $150_{10}$ words (the first word stored in location BLK1). The second block consists of $50_{10}$ words (the first word stored in location BLK2). The symbolic location TEMP1 contains the address of the first COTW to be used for the transmission.

2.180

| Location | Contents | Remarks |
|---|---|---|
| TEMP1 | xxxxxxxx COTW1<br>0 8 9 23 | Pointer to Instruction Packet |
| COTW1 | 1 1 0 0 DEVICE CONTROL<br>0 1 2 3 4 23 | Output, Chain, No Interrup, Transmit |
| COTW1+1 | xxxxxxxxxxxx 000010010110<br>0 11 12 23 | Word Count<br>$150_{10}$ Words |
| COTW1+2 | xxxxxxxx BLK1<br>0 8 9 23 | First Word Address |
| COTW1+3 | xxxxxxxx COTW2<br>0 8 9 23 | Chain Link |
| COTW2 | 1 0 1 0 DEVICE CONTROL<br>0 1 2 3 4 23 | Output, No Chain, Interrupt, Transmit |
| COTW2+1 | xxxxxxxxxxxx 000000110010<br>0 11 12 23 | Word Count<br>50 Words |
| COTW2+2 | xxxxxxxx BLK2<br>0 8 9 23 | First Word Address |

For this example, the appropriate NXAR must contain the address TEMP1 prior to initiating channel IO. When the sequence instruction request for the first block is completed, NXAR contains COTW1+3. Also, both blocks (200 words) will be output before interrupt.

Input Output Requests - When a device synchronizer requests a data transfer (IO request), the channel IO logic causes a single word to be transferred to or from memory. The memory location affected by this transfer is specified in the channel's data address register (dedicated memory for low-speed channels; channel hardware register for high-speed channels). Each time such a request is processed, the data word is counted, and the contents of the data address register are incremented by 1. The incremented address is used during the next word transfer.

When the number of words specified by the word count is satisfied, the device synchronizer is signalled and will then terminate operation or request a new sequence instruction, depending on the chaining request (CHR) bit of the previous sequence instruction process. Also, if the interrupt bit (EOI) of the previous sequence instruction process was set, interrupt occurs when the word count is satisfied.

Operation Notes - The following notes should be observed concerning CIO operation:

a. Channel IO addresses memory in an end-around method; i.e., the next address after 77777 is 00000.

b. A word count of 0000 is interpreted to mean 4,096 words on high-speed channels and 32,768 words on low-speed channels.

c. Channel IO addresses of 77760 through 77777 are interpreted as memory addresses rather than CP register addresses. Channel IO has no access to the CP addressable hardware registers.

d. The initial indirect address of the first CIO instruction word must be preloaded by programming technique into the proper NXAR register prior to initiation of channel IO device activity.

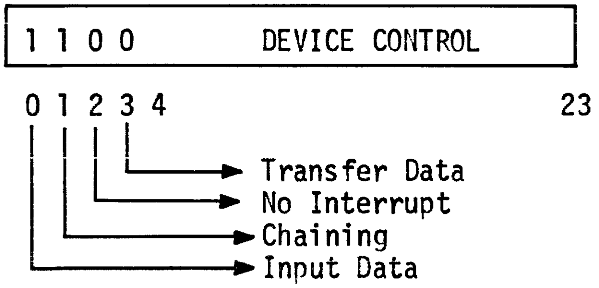
Channel IO Interrupt - Channel IO normally causes an interrupt only when the number of words specified by the word count in the channel order triple word has been transmitted, and if the instruction (word 1 of the COTW) specified EIO = 1.

In most cases, EOI = 0, except in the last COTW used in a channel transmission. This means that, normally, in a chained transmission, EOI = 1 for the transmission of the last block of data in the chain.

Example:

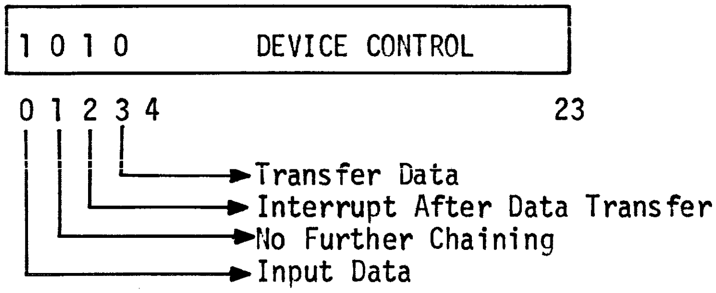
The first words of the COTWs for input of three blocks of data in three different memory areas might be as follows:

LOC
COTW1:



COTW2: Same as COTW1

COTW3:



If the EOI is set in each COTW, interrupt will occur after the transfer of each block of data; but if the chaining bit is set, chaining also occurs, and the next transfer of data begins after the interrupt is turned off.

Channel IO Timing - CIO timing for each type of request on each type of channel is shown in Table 2-2-11.

The range in timing is due to the wait for memory availability if the central processor is in the midst of a memory cycle.

## Detailed Functional Description

IO Master Control Logic - All CIO operations are governed by the combination of operation state counter output (OP) and sequence counter output (SQ). The 3-stage OP counter provides seven decoded outputs. The appropriate OP state is established at the beginning of a channel request cycle for the duration of that cycle. The 5-stage sequence counter provides 18 decoded output sequences. A CP clock pulse advances the SQ by one each pulse train, except when a memory cycle is requested. Then, the advance is blocked until the memory control logic sends a read complete (RDC) signal, indicating that the memory data transfer is complete. OP determines the number of sequence states used and the type of controls generated in any given operation.

Channel Selection - When the priority logic honors the highest pending channel activity request, the corresponding scan flip-flop (SCN) is set. The combination of SCN and the request flip-flop sets the appropriate operation state for the type of channel, high or low speed, and the type of request. Although all channels have individual control registers, low speed channel registers are dedicated core memory locations. SCN, together with OP/SQ decoding, generates the appropriate address for control registers in core or in the case of high speed channels (0 and 1), gates information for hardware control registers.

## Table 2-2-11. Channel IO Operations Summary

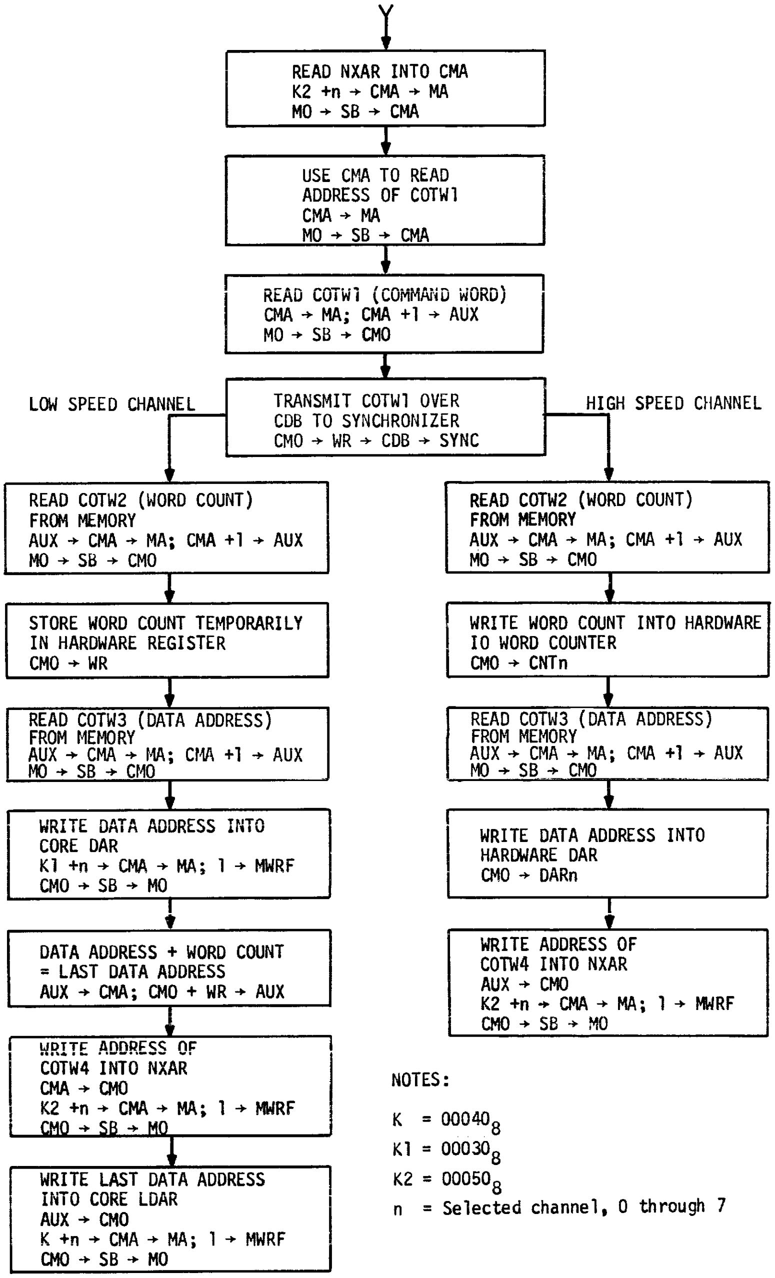
| OPERATION | CHANNEL TYPE | MEMORY CYCLES | SEQUENCE OF EVENTS | COMMENTS |
|---|---|---|---|---|
| Sequence Instruction Request (SIR) | High Speed | Read 1 | 1. Read vector from next instruction address register. | Second level indirect, points to vector for COTW. |
| 6.08 to 6.72 us | | 1 | 2. Read vector referenced by NXAR. | |
| | | 3 | 3. Read channel order triple word (COTW). | |
| | | | a. Command | Transmitted to device synchronizer. |
| | | | b. Word count | Stored in IOM hardware register. |
| | | | c. Data address | Stored in IOM hardware register. |
| | | Write: 1 | 4. Update NXAR. | Write indirect vector for next COTW into NXAR. |
| | | Total 6 | | |
| Sequence Instruction Request (SIR) | Low Speed | Read 1 | 1. Read vector from NXAR. | |
| 8.32 to 8.96 us | | 1 | 2. Read vector referenced by NXAR. | |
| | | 3 | 3. Read COTW. | |
| | | | a. Comand | Transmitted to device synchronizer. |
| | | | b. Word count | Stored by IOM temporarily in hardware register. |
| | | | c. Data address | Stored by IOM temporarily in hardware register. |
| | | Write: 1 | 4. Write data address in core data address register. | |
| | | 1 | 5. Update NXAR. | Write indirect vector for next COTW into NXAR. |
| | | | 6. Develop last data address: | |
| | | | a. Add word count to data address | |
| | | 1 | b. Write last data address (LDAR) into core. | |
| | | Total 8 | | |
| Read Data or Write Data | High Speed | | 1. Decrement hardware word counter. | Last word level (LWL) sent to synchronizer if count = zero. |
| RD: 2.24 to 2.56 us/word | | Read or Write: 1 | 2. Transmit or receive memory data. | |
| Wr: 2.56 to 2.88 us/word | | | 3. Increment hardware data address register (DAR). | |
| | | Total 1 | | |
| Read Data or Write Data | Low Speed | Read 1 | 1. Read core LDAR contents. | Last word level (LWL) sent to synchronizer if DAR = LDAR. |
| | | 1 | 2. Read core DAR contents. | |
| Rd: 4.16 to 4.8 us/word | | Read or Write: 1 | 3. Transmit or receive memory data. | |
| Wr: 4.48 to 5.12 us/word | | Write: 1 | 4. Update DAR: | |
| | | | a. Increment data address by one. | |
| | | | b. Write new address | |
| | | Total 4 | | |

Low Speed Channels - The increased access time for a word in core memory compared to one in a hardware register is the primary difference between high and low speed channels. Both types use the same channel order triple word (COTW) format, the same data format, and the same control lines, but the method of processing low speed requests is modified to make the most efficient use of the relatively time-consuming memory cycles.

CIO core registers are designated, according to the type of information stored, as last data address register (LDAR), data address register (DAR), and next instruction address register (NXAR). CIO operation state and sequence logic generates the address of a given core register from two components, block address, and word within the block. A constant, K, signifies the address of the block of LDAR's; K1 signifies the address of the block of DAR's; and K2 signifies the address of the block of NXAR's. K has a value of $00040_8$, K1 a value of $00030_8$, and K2 a value of $00050_8$. The particular scan flip-flop set, such as SCN2 for channel 2, provides the units digit of the address to identify the register within the block, as $00052_8$ for channel 2 NXAR.

Sequence Instruction Request, High Speed Channel (Figure 2-2-11) - Servicing a sequence instruction request (SIR) for a high speed channel requires transfers of information over the CIO bus between the synchronizer and the IO master, within the IO master between channel and common logic, and over the system bus between the IO master and memory as follows:

1. Obtain Command Word -- The first memory cycle reads the channel's next instruction address register (NXAR). The indirect address obtained is then used in the second memory cycle to read the address of the first word of the channel order triple word (COTW), the command word. The third memory cycle reads the command word.



NOTES:

K = $00040_8$

K1 = $00030_8$

K2 = $00050_8$

n = Selected channel, 0 through 7

| MNEMONIC | NAME | LOCATION |
|---|---|---|
| AUX | Auxiliary Register | IOM |
| CDB | Channel Data Bus | |
| CMA | Channel Memory Address Register | IOM |
| CMO | Channel Memory Output Register | IOM |
| DAR | Data Address Register | IOM/MEM |
| IOM | Input/Output Master | |
| MA | Memory Address Register | MEMORY |
| MO | Memory Output Register | MEMORY |
| MWRF | Memory Write Flip-Flop | IOM |
| NXAR | Next Instruction Address Register | MEMORY |
| SB | System Bus | |
| WR | Word Register | IOM |

Figure 2-2-11. Sequence Instruction Request, High/Low Speed Channel

2. Channel Command Sequence -- The command word is transferred to the channel data bus (CDB) and the synchronizer signalled to transfer the command word to its command register.

3. Word Count Sequence -- The fourth memory cycle reads the second word of the COTW, the word count for the order, which is then transferred to the channel's word counter (CNT).

4. Data Address Sequence -- The fifth memory cycle reads the third word of the COTW, the data address for the order, which is then transferred to the channel's data address register (DAR).

5. Update NXAR -- The sixth memory cycle, a write operation, updates the contents of the channel's NXAR. The new value in NXAR is the address of the word following the third word of the COTW. This word should contain the address of the first word of the next COTW to be executed by that channel.

<u>Sequence Instruction Request, Low Speed Channel (Figure 2-2-11)</u> - The following transfers are necessary to service a sequence instruction request (SIR) from a device synchronizer using a channel from 2 through 7:

1. Obtain Command Word -- The first memory cycle reads the contents of the channel's next instruction address register (NXAR). This indirect address is then used in the second memory cycle to read the <u>address</u> of the first word of the channel order triple word (COTW), the command word. The third memory cycle reads the command word.

2. Channel Command Sequence -- The command word is transferred to the channel data bus (CDB) and the synchronizer signalled to transfer the command word to its command register.

3. Word Count Sequence -- The fourth memory cycle reads the second word of the COTW, the word count for the channel order, which is stored in a hardware register temporarily until the data address is obtained.

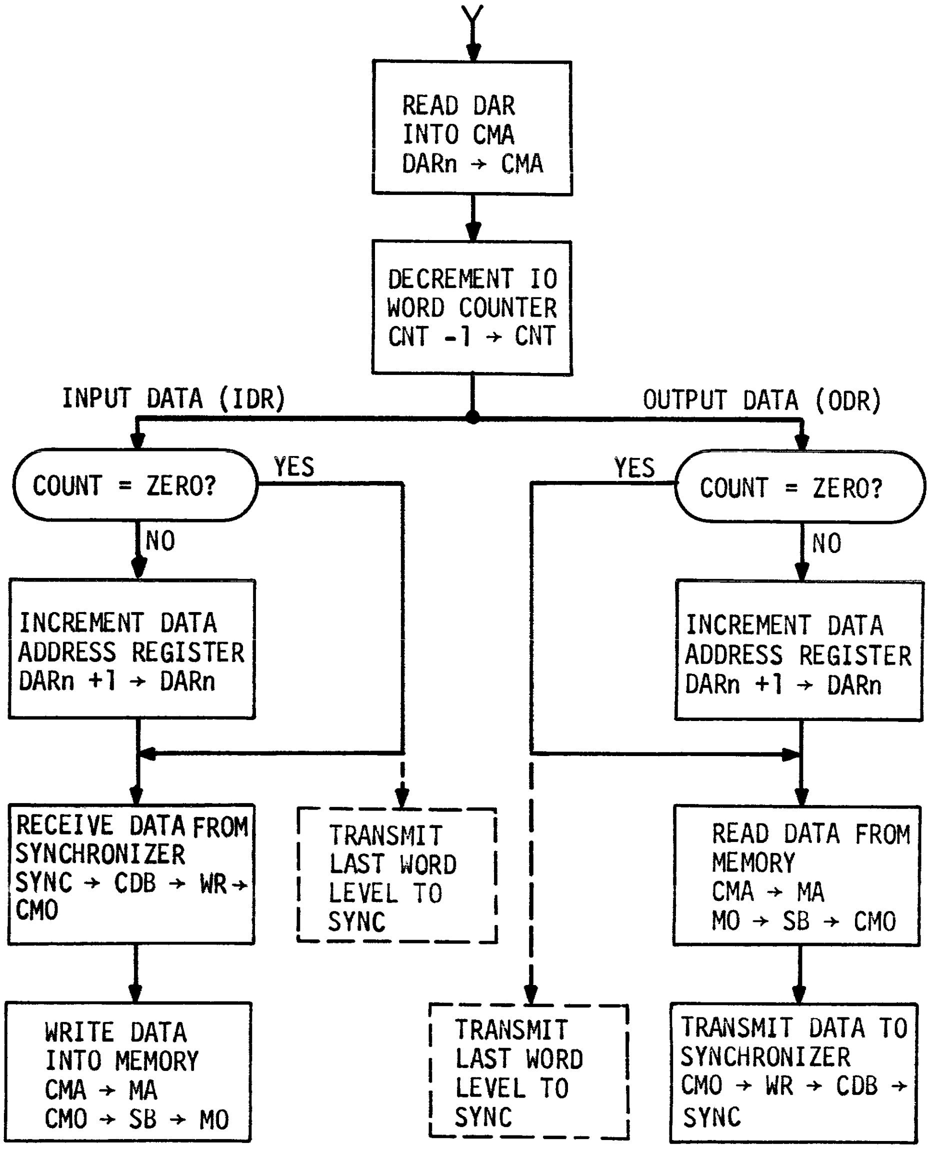
4. Data Address Sequence -- The fifth memory cycle reads the third word of the COTW, the data address for the order. A memory write cycle stores the data address in the channel's data address register (DAR).

5. Compute Last Data Address -- The data address and the word count are added to generate the last data address.

6. Update NXAR -- The seventh memory cycle, a write operation, updates the contents of the channel's NXAR. The new value in NXAR is the address of the word following the third word of the COTW.

7. Store Last Data Address -- The eight memory cycle stores the last data address in the channel's last data address register (LDAR).

Input Data Request, High Speed Channel (Figure 2-2-12) - Servicing an input data request (IDR) for a high speed channel requires transfers of information over the CIO bus between the synchronizer and the IO master, within the IO master between channel and common logic, and over the system bus between the IO master and memory as follows:



| MNEMONIC | NAME | LOCATION |
|---|---|---|
| CDB | Channel Data Bus | |
| CMA | Channel Memory Address Register | IOM |
| CMO | Channel Memory Output Register | IOM |
| CNT | IO Word Counter | IOM |
| DAR | Data Address Register | IOM |
| IOM | Input/Output Master | |
| MA | Memory Address Register | MEMORY |
| MO | Memory Output Register | MEMORY |
| n | Selected Channel | |
| SB | System Bus | |
| WR | Word Register | IOM |

Figure 2-2-12. Input/Output Data Sequence, High Speed Channel

1. Check for Last Data Word -- The IO word counter (CNT) for the channel is decremented by one at the start of each data transfer sequence. A word count of zero sets the IOM's last word level flip-flop (LWLF).

2. Channel Data Sequence -- The device data is accepted from the channel data bus (CDB). If the LWLF is set, the last word level control signal is sent to the synchronizer.

3. Memory Data Sequence -- A memory write cycle is requested to store the data word in the location designated by the data address register (DAR).

4. Data Address Update -- If the LWLF is reset, the contents of the channel's DAR are incremented by one.

Input Data Request, Low Speed Channel (Figure 2-2-12) - The following transfers are required to service an input data request (IDR) from channel 2 through 7:

1. Check for Last Data Word -- The first memory cycle reads the contents of the last data address register (LDAR). The second memory cycle reads the contents of the channel's data address register (DAR). The data address is subtracted from the last data address, and if the difference is equal to one, the IOM's last word level flip-flop (LWLF) is set.

2. Channel Data Sequence -- The device data is accepted from the channel data bus (CDB). If the LWLF is set, the last word level control signal is sent to the synchronizer.

3. Memory Data Sequence -- The third memory cycle, a write operation, stores the data word in the location designated by the channel's DAR.

4. Data Address Sequence -- If the LWLF is reset, the channel's data address is incremented. The fourth memory cycle, a write operation, stores the updated data address in the channel's DAR.

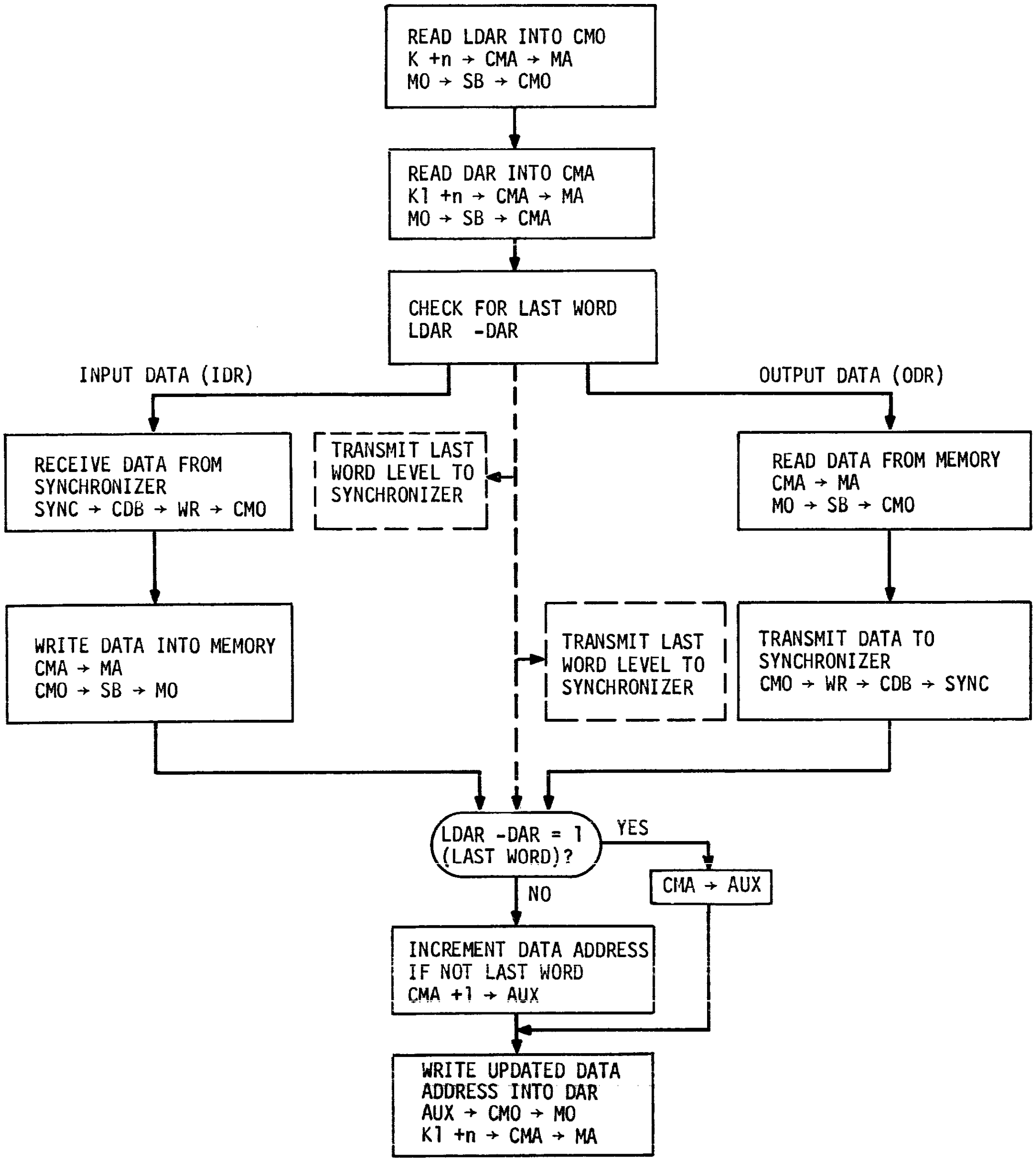
Output Data Request, High Speed Channel (Figure 2-2-13) - Servicing an output data request (ODR) for a high speed channel requires transfers of of information over the CIO bus between the synchronizer and the IO master, within the IO master between channel and common logic, and over the system bus between the IO master and memory as follows:

1. Check for Last Data Word -- The IO word counter (CNT) for the channel is decremented at the start of each data transfer sequence. A word count of zero sets the IOM's last word level flip-flop (LWLF).

2. Memory Data Sequence -- A memory cycle reads the data word at the address obtained from the channel's data address register (DAR).

3. Channel Data Sequence -- The data word is transferred to the channel data bus (CDB) and the synchronizer signalled to transfer data from the CDB to its data register.

4. Data Address Update -- If the LWLF is reset, the contents of the channel's DAR are incremented by one.



| MNEMONIC | NAME | LOCATION |
|---|---|---|
| AUX | Auxiliary Register | IOM |
| CDB | Channel Data Bus | |
| CMA | Channel Memory Address Register | IOM |
| CMO | Channel Memory Output Register | IOM |
| DAR | Data Address Register | MEMORY |
| IOM | Input/Output Master | |
| LDAR | Last Data Address Register | MEMORY |
| MA | Memory Address Register | MEMORY |
| MO | Memory Output Register | MEMORY |
| SB | System Bus | |
| WR | Word Register | IOM |

Figure 2-2-13. Input/Output Data Sequence, Low Speed Channel

Output Data Request, Low Speed Channel (Figure 2-2-13) - The following transfers are required to service an output data request (ODR) from channel 2 through 7:

1. Check for Last Data Word -- The first memory cycle reads the contents of the last data address register (LDAR). The second memory cycle reads the contents of the data address register (DAR). The data address is subtracted from the last data address, and if the difference is equal to one, the IOM's last word level flip-flop is set.

2. Memory Data Sequence -- The data address is used to read a data word in the third memory cycle.

3. Channel Data Sequence -- The data word is transferred to the channel data bus (CDB) and the synchronizer signalled to transfer data from the CDB to its data register.

4. Data Address Sequence -- If the last word level flip-flop is reset, the data address is incremented by one. In the fourth memory cycle, the new data address is written into the channel's data address register.

# 3/FORMATS AND ADDRESSING

## MEMORY WORDS

A memory word contains information that can be used by the central processor (CP) as either an instruction or as data. Words in each memory location are composed of 24 binary digits (bits) plus a parity bit. The parity bit is generated and checked within the CP to determine accuracy of information transfer into or out of memory and is not accessible for use by a program.

A word from one memory location is in a short or single-word format in which the 24 information bits are numbered from 00 through 23. Bit 00 is the most significant bit (MSB), bearing a binary weight of 8,388,608 ($2^{23}$ = 8388608), and bit 23 is the least significant bit (LSB), with a binary weight of 1 ($2^0$ = 1). A short-format word is usually displayed with bit 00 on the left, with bit weights decreasing (bit number increasing) to bit 23 on the right.

Instruction logic also defines long or double-word format information. This information normally occupies two contiguous memory locations. The 48 bits of the long format are numbered 00 through 47. The word with the smaller address contains bits 00 through 23 (24 high-order bits), and the word with the larger address contains bits 24 through 47 (24 low-order bits). The memory reference address of long-format information is the address of the first of the two memory words.

Long or double words have the format:

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Word 1: 24 High-Order Bits | | | | | | | | | | | | | | | | | | | | | | | |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| Word 2: 24 Low-Order Bits | | | | | | | | | | | | | | | | | | | | | | | |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |

The exact nature of the information that may be held in a memory word or double word depends entirely upon information usage. The various information formats that are imposed upon memory words are included with the descriptions of the central processor and the memory-referencing instructions.

For ease in representing the contents of memory words, a short word can be divided into eight 3-bit fields and a long word into sixteen 3-bit fields. Each 3-bit field can then be represented by one octal digit.

## INSTRUCTION WORDS

A memory word is designated as an instruction word when its address appears in the PC register, and it becomes an instruction when the word is transferred from memory into the instruction register. In the instruction register, the word is decoded to define the precise operation to be performed. Several parameters are required to specify each operation. In general, these parameters specify an operation code and a memory address called the "effective address". The operation code (OP) in the six MSBs defines an action that, in most cases, is performed upon one or more operands. The effective address (EA) parameters in the 18 LSBs defines one of the following, depending upon the operation code and addressing parameters:

a. The location of an operand that is to be fetched from memory.

b. The location into which an operand is to be stored.

c. The location of an operand that is to be altered.

d. An operand itself; in this case, EA is used as an immediate operand, eliminating the need for memory access.

e. The location of the next instruction (this is a branch condition).

The EA parameters provide a versatile set of techniques for determining an effective address for an instruction and enhance flexibility in program control and memory access. EA parameters define three specific instruction formats: absolute, literal, and relative. Distinction between these formats is made by the mode field (M) consisting of bits 06, 07, and 08 of the instruction word.

| Value of M | Format Specification |
|---|---|
| M = 0 or 1 | Absolute Format |
| M = 2 | Literal Format |
| M = 3, 4, 5, 6, or 7 | Relative Format |

Absolute Instruction Format

Absolute instructions are composed of three fields in the following format:

| OP | | | | | | M | | | Y | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

where:

OP - The operation code field. The six bits designate the particular operation the CP is to perform. These operations are defined in Section 4, Instruction Repertoire.

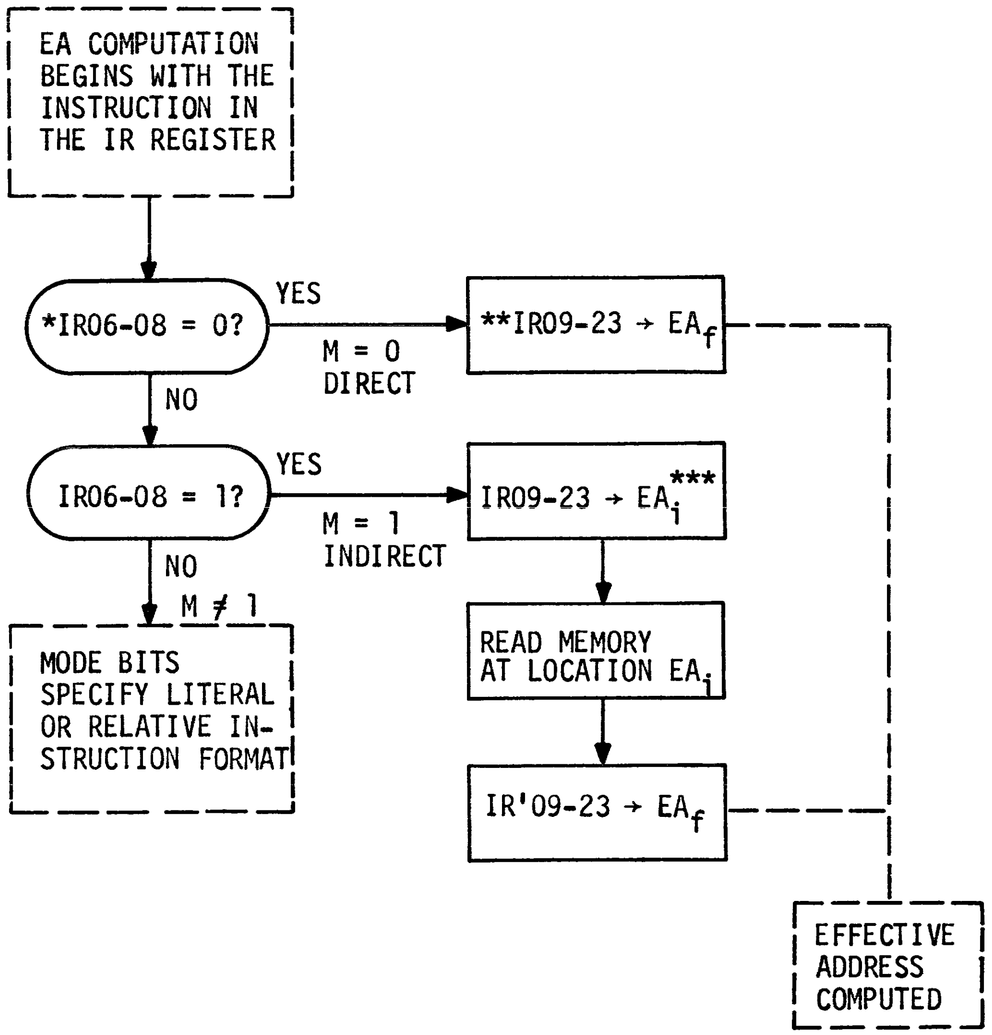
M - The mode field. This field defines the type of effective address computation to be used for the instruction. In absolute addressing format, the M field contains the value 0 or 1.

Y - The address field. This field can specify the effective address or the address of a memory word that contains the effective address.

Absolute Addressing (Figure 2-3-1)

The value of M (mode field) determines the utilization of Y (address field) in obtaining EA. The possible values of M have the following interpretations:

| Value of M | Interpretation |
|---|---|
| M = 0 | Absolute direct addressing mode. |
| M = 1 | Absolute indirect addressing mode. The memory location specified by Y supplies the effective address. |


EA COMPUTATION BEGINS WITH THE INSTRUCTION IN THE IR REGISTER
*IR06-08 = 0?
YES
M = 0
DIRECT
**IR09-23 → $EA_f$
NO
IR06-08 = 1?
YES
M = 1
INDIRECT
IR09-23 → $EA_i$***
NO
M ≠ 1
MODE BITS SPECIFY LITERAL OR RELATIVE IN-STRUCTION FORMAT
READ MEMORY AT LOCATION $EA_i$
IR'09-23 → $EA_f$
EFFECTIVE ADDRESS COMPUTED

*Mode field of Instruction Register.
**Y field of Instruction Register.
***Intermediate Effective Address.

Figure 2-3-1. Computation of EA for Absolute Instructions

Absolute Direct Mode - In absolute addressing (mode 0), the Y field (bits 09 through 23) of the instruction word is used directly as the effective address for the instruction. This allows an instruction to specify directly any storage location address, up to 77777, as the effective address.

Examples:

a. To load the A register with the contents of memory location 01076, the instruction could be:

| OP | M | Y |
|---|---|---|
| 53 | 0 | 01076 |

where:

53 is the load A operation, 0 specifies direct mode, and EA = 01076.

b. To double the contents of the A register and leave the result in A, the instruction could be:

| OP | M | Y |
|---|---|---|
| 10 | 0 | 77775 |

10 is the ADD operation, 0 specifies direct mode, and EA = 77775 (A is one of the addressable registers, address 77775).

Absolute Indirect Mode - The absolute indirect addressing mode (M = 1) interprets the Y field as containing the address of the memory location, which, in turn, contains the effective address. In this case, only bits

09 through 23 of the specified memory word are used as the 15-bit effective address. Bits 00 through 08 are ignored.

Example:

If memory location 01076 contains the octal value 53302044, execution of the instruction:

| OP | M | Y |
|---|---|---|
| 53 | 1 | 01076 |

causes the contents of location 02044 to be loaded into the A register. Fifty-three is the load A operation code, 1 specifies indirect mode, 01076 is the address of the memory word containing the effective address, and EA = 02044 (bits 09 through 23 of the word in location 01076). Bits 00 through 08 of the word in location 01076 are ignored.

Literal Instruction Format

The literal instruction format is identical to the absolute format except in the value and intepretation of the mode control bits. If the value of M is 2, literal format is specified. The exact utilization of Y in mode 2 operations depends upon the operation code (OP). The operations (defined in Section 4) are grouped into three literal format classes: class 1, class 2, and class special.

Literal Class 1 - This class includes all operations that normally require an operand to be fetched from memory. In these cases, the contents of the Y field, with bit 09 extended left into bits 00 through 08 to form a full 24- or 48-bit word, is the operand. Use of immediate operands creates savings of both time and storage space. An immediate operand specified by a long instruction (i.e., an instruction that uses a double-word operand) has the Y field right-justified in the 48-bit double-word and bit 09 of Y is extended left into bits 00 through 32. One or two full words of memory are saved by not having the operand in storage, and the time required to execute an instruction using an immediate operand is reduced by 0.32 microsecond. (Refer to instructions in Section 4 for timing.)

Examples:

a. Execution of this instruction causes the immediate operand specified in Y to be loaded into the A register.

| OP | M | Y |
|---|---|---|
| 53 | 2 | 00005 |

Bit 09, the high-order bit of Y, is extended left into bits 00 through 08 to form a full 24-bit word. Regardless of the contents of the A register prior to execution of the instruction, after execution, the A register contains the 24 bits:

$$\underbrace{000\ 000\ 000}_{\text{Extended Bits}}\ \underbrace{000\ 000\ 000\ 000\ 101}_{\text{Original Y}}$$

b. If the instructions were:

| OP | M | Y |
|---|---|---|
| 53 | 2 | 60000 |

the result in the A register would be:

$$\underbrace{111\ 111\ 111}_{\text{Extended Bits}}\ \underbrace{110\ 000\ 000\ 000\ 000}_{\text{Original Y}}$$

Literal Class 2 - This class includes all operations that store information into memory or that cause a branch. In this case, the mode of the instruction is equivalent to M = 0 in absolute addressing; i.e., the value of Y is the effective address of the instruction.

For example:

| OP | M | Y |
|---|---|---|
| 44 | 2 | 00450 |

In this case, since 44 is the store accumulator operation, M = 2 is taken to be the equivalent of M = 0 (absolute direct mode); i.e., the address 00450 is used as the effective address, and the instruction causes the contents of the A register to be stored in memory location 00450.

Literal Class Special - A special interpretation of the literal mode is applied to instructions involved in subroutine linkages (BSP, BSR, and BRU operations and long format floating point instructions (FAL, FSL, FML, and FDL). These instructions require special handling and are described, with examples, in Section 4 of this document.

Relative Instruction Format

The relative instruction format provides a variety of options in computing effective addresses. These options are provided by bits 06 through 11 of the instruction word, and specify:

a. Base relative addressing.

b. Indexing.

c. Indirect addressing by means of an indirect vector that provides additional second- and third-level options.

Instruction words using relative addressing have the following format:

| OP | | | | | | M | | | I | XB | XA | DISP | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

where:

OP - The operation code field. The six bits designate the particular operation the CP is to perform.

M - The mode control field. This field defines the type of effective address computation to be used for the instruction (M > 2 specifies relative addressing) and defines which CP register is to be used as a displacement base in computing EA.

I - Indirect field. This bit specifies direct (I = 0) or indirect (I = 1) addressing. If I = 0, EA is computed from information in the instruction. If I = 1, information in the instruction points to a word in memory that either provides the effective address or points to another word that specifies EA.

XB - Index B field. This bit specifies the use of the index B (XB) register in computing EA. If XB = 0, index B is not used. If XB = 1, index B is used.

XA - Index A field. This bit specifies the use of the index A (XA) register in computing EA. If XA = 0, index A is not used. If XA = 1, index A is used.

DISP - Displacement field. This 12-bit field contains a displacement relative to the base specified by M. The value range of DISP is:

$$0000 \leq \text{DISP} \leq 7777$$

or, effectively:

$$-2048_{10} \leq \text{DISP} \leq +2047_{10}$$

The value of M (mode field) specifies a particular relative base to be used in computing EA.

| Value of M | Base Used |
|---|---|
| M = 3 | Zero relative |
| M = 4 | Program counter (PC) relative |
| M = 5 | Index common (XC) relative |
| M = 6 | Top-of-stack (XT) relative |
| M = 7 | Index executive (XE) relative |

## Base Relative Addressing (Figure 2-3-2)

In relative addressing, an effective address is calculated through use of a base address, plus a displacement relative to that base. The instruction mode field specifies one of five available bases (as listed in the preceding paragraph), and the displacement field specifies a divergence from the base. The 12-bit displacement field is extended left (three places) to form a 15-bit address value. This value is in 2's complement notation, allowing specification of an address above or below the base address, as well as the address identified directly by the base. Displacement range is from 2048 locations preceding the base, to 2047 locations following the base address. Ring or end-around addressing occurs, in that the next address after the maximum address of 77777 is 00000.

An effective address is computed as the algebraic sum of the base address (usually the contents of a hardware register) and the extended displacement. For this addition, bit 12 of the DISP field is extended left (three bits) to form a 15-bit address value. For example, if the base address is 05010 and the displacement is 0005, the relative address is computed as:

```
     000  101  000  001  000   =  05010 (base)
    +000  000  000  000  101   =  00005 (DISP extended)
     000  101  000  001  101   =  05015 (relative address)
```

If the displacement is 4000 (1 in bit 12) or greater, this addition has a negative effect. For example, if the base address is 05010 and DISP is 77773 (-5), the addition is:

```
     000  101  000  001  000   =  05010 (base)
    +111  111  111  111  011   =  77773 (DISP extended)
   1 000  101  000  000  011   =  05003 (relative address)
   └→This carry bit is not part of the resulting relative
     address.
```

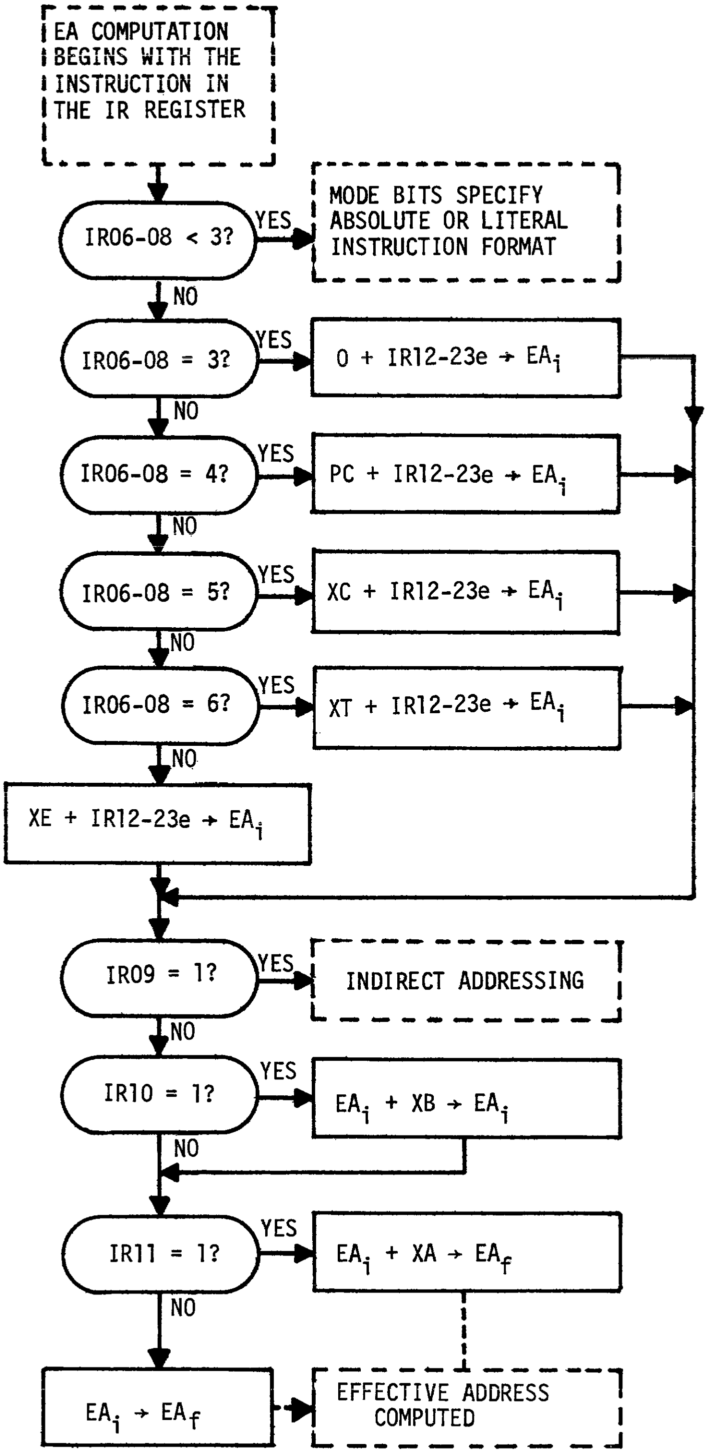


Figure 2-3-2. Relative Direct EA Computation

The FOX 1 CP provides six relative addressing bases. The base addresses are held in five 15-bit addressable registers and one pseudo-register. Figure 2-3-3 shows a typical memory map for a program unit and indicates areas associated with the base addresses.

Zero Base (M = 3) - This is a pseudo-register that provides a constant base address of 00000. The memory area defined by this base is composed of locations 74000 through 03777. The zero base is used in relative addressing to access absolute locations.

Program Counter (PC) Base (M = 4) - The PC base allows addressing relative to the location of the instruction being executed. Therefore, PC base defines a memory area that is constantly changing, because PC is constantly changing. The advantage of PC relative addressing is that references can be made to words within a program without being dependent upon the storage location of the program. This is particularly helpful in program relocation. For example, if an instruction requires access to information in a memory location seven locations after the storage location of the instruction, the instruction will be:

| OP | M | DISP |
|---|---|---|
| XX | 4 | 0007 |

If two instructions in succession require access to this same relative location, the instruction will be:

| OP | M | DISP |
|---|---|---|
| XX | 4 | 0007 |
| XX | 4 | 0006 |

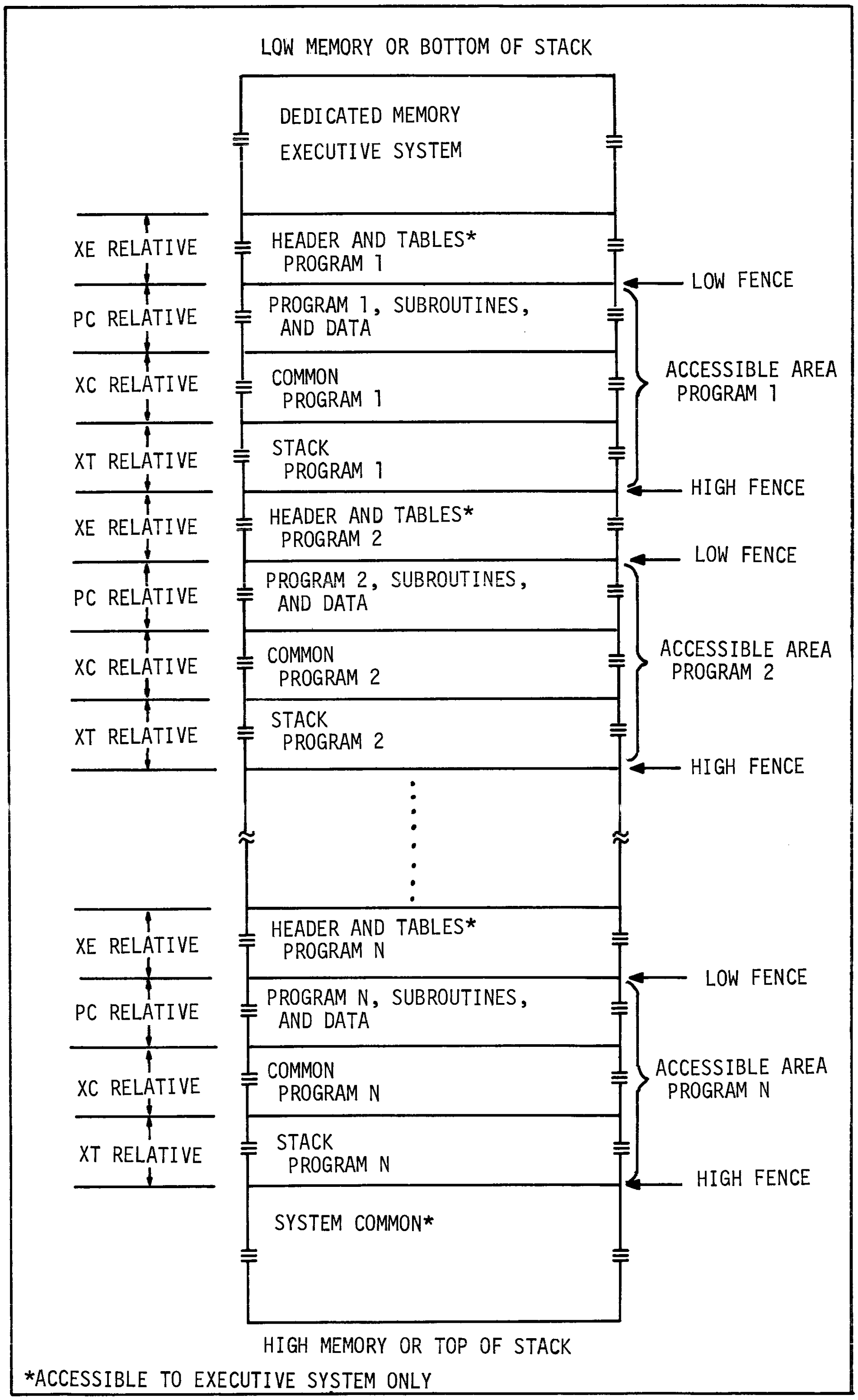


Figure 2-3-3. Typical Memory Map

To access a location seven locations prior to the location of an instruction, the instruction is:

| OP | M | DISP |
|---|---|---|
| XX | 4 | 7771 |

The relative address generated by this instruction is equivalent to PC minus 7.

Index Common (XC) Base (M = 5) - The XC base facilitates defining a memory area for use as common storage by a program unit. An address is maintained in XC, which points to a specific location in the common area. Instructions may refer to a word in the common area by using the contents of XC as an address base and providing a displacement from that base to point to a particular word whose location is relative to the common base.

A program unit (load module) is composed of a main program, subroutines, and data. Generally, the XC base is determined relative to the program unit origin at object program generation time (assembly or compilation). The common area follows the program unit in storage. The absolute value of XC is set by the real time executive system (RTX) when a program is loaded into memory.

For example, if a program unit is $1500_8$ words in size and requires $100_8$ words of common storage, program generation might establish an XC base of 01540 based on a program origin of 00000. If the RTX loads the program into memory beginning at location 07000, the program origin will then be 07000 and the XC base will be 10540. Instructions using XC relative addressing will refer to the memory area relative to 10540.

The XC base may be established at any point within the common area. It should be noted, however, that if XC points to the first word of common, then negative displacements from XC accesses locations in the program unit and the common area is limited to 2048 words. A common area of 4096 words may be used by specifying the XC base as the midpoint location. Then a full range of displacements will reference common.

The real time executive system and the standard language processors (assembler and compiler) allow the user to specify XC relative to program origin so that the XC base location may be anywhere within the common area. If this is not done, XC will point to the first word of the common area.

Index Top-Of-Stack (XT) Base (M = 6) - The XT base identifies the top (location with the largest address) of the push-down stack and provides base-relative access to words in the stack. The stack is used for subroutine linkages and temporary storage (refer to BSP and BSR instructions, Section 4). The stack is expanded when subroutines are called. Exit from a subroutine contracts the stack. This provides dynamic storage, which economizes storage requirements and allows recursive and reentrant subroutines in the stack. Under interrupt operations, a single subroutine may be called any number of times before any exit is achieved. A new top-of-stack is established for each subroutine call. Thus, information relating to each subroutine call is not destroyed. Exits associated with each subroutine call return XT to its last previous value.

Instructions that address information in the stack use XT relative addressing to access information relative to the current top-of-stack, independent of the XT base address. An address is maintained in the XT register, which points to the top of the stack. Instructions may refer to a word in the stack by using the contents of XT as an address base and providing a displacement from that base to point to a particular word whose location is relative to the top of the stack.

Generally, each program unit is assigned its own stack space, with the XT base determined relative to program-unit origin at object program generation time (assembly or compilation). The stack follows the program unit and common area in storage. The initial actual XT for program execution is set by the real time executive system when the program unit is loaded into memory.

For example, if a program unit is $1500_8$ words in size and requires $100_8$ words of common storage, program generation will establish an XT base of 01600 based on a program-unit origin of 00000. If the RTX loads the program beginning at location 07000, the program origin will be 07000, XC base will be 10500, and XT base will be 10600.

This method of establishing an initial XT is a convention of the real time executive system. However, incrementing and decrementing XT to expand and contract the stack is a function of the BSP, BSR, and BRU instructions.

Index Executive (XE) Base (M = 7) - The XE base is used primarily to point to the first word of header information of prime user programs. By convention, the XE base is reserved for exclusive use by the real time executive system. XE identifies an area preceding program storage that contains information about the program that is used for loading the program and establishing its operating conditions (such as privileges, memory fence, XC and initial XT bases, and so forth). The memory area defined by XE is not generally accessible to a user's program.

Operation not using the RTX system can assign a different function to XE.

## Indexing Relative Addresses

Two indexes are provided for use by relative instructions. These indexes occupy two 15-bit addressable registers: index A register (address 77774) and index B register (address 77773).

Bits 10 and 11 of relative addressing instructions specify use of these indexes in computing the effective address of the instruction. The function of these registers in this computation is described as follows.

Index A (XA) - If bit 11 of the instruction word is set (XA = 1), index A serves as a post-index for indirect addressing or as a general index for direct addressing. In either case, adding of the contents of the index A register is the final step in computing an effective address.

Index B (XB) - If bit 10 of the instruction word is set (XB = 1), index B serves as a pre-index for indirect addressing if index A is also set (XA = 1, XB = 1), or as a post-index if index A is reset (XA = 0, XB = 1). For direct addressing, index B serves as a general index. When used as a pre-index, index B is added to a relative address to determine the location of the indirect vector for indirect addressing. Otherwise, index B is used similarly to index A.

## Relative Indirect Addressing

Relative indirect addressing is specified if bit 9 of the instruction word is set (i.e., I = 1). If I = 0, relative direct addressing is specified and the effective address for the instruction is directly determined by the EA parameters.

If I = 1, the address determined by the EA parameters (including base relative and pre-indexing) identifies the memory location of an indirect vector which provides a new set of parameters for determining the effective address.

When the address of the indirect vector has been determined, the vector is transferred from memory into a supplementary instruction register (IR') and decoded. The vector has the following format:

<table>
<tr><td colspan="6">IGNORED</td><td colspan="3">M'</td><td colspan="15">Y'</td></tr>
<tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td></tr>
</table>

where bits 0 through 5 are ignored, and:

M' - The mode control field of the vector.

Y' - A displacement value similar to DISP in the relative instruction format. Note that Y' contains 15 bits and, thus, requires no left extension to achieve standard address size as does DISP.

The mode bits of the vector (M') are interpreted as follows with the indicated results:

| Value Of M' | Interpretation/Result |
|---|---|
| M' = 0 | Equivalent to M' = 3. Specifies zero-base relative addressing; i.e., EA = 00000 + Y' + post-index. |
| M' = 1 | Specifies a second level of indirect addressing; i.e., the contents of the location specified by Y' plus a post-index (if any) that provides the effective address. EA = (Y'09-23) + post index. |

| Value of M' | Interpretation/Result |
|---|---|
| M' = 2 | Specifies an escape condition. The FNV (fence violation) bit of the central processor status register (CPS) is set, the instruction is aborted and a level 01 interrupt occurs. The real time executive system uses this condition to activate program overlay through interrupt. |
| M' = 3 | Specifies zero-base relative addressing; i.e., EA = 00000 + Y' + post-index. |
| M' = 4 | Specifies PC-base relative addressing; i.e., EA = PC base + Y' + post-index. |
| M' = 5 | Specifies XC-base relative addressing; i.e., EA = XC base + Y' + post-index. |
| M' = 6 | Specifies XT-base relative addressing; i.e., EA = XT base + Y' + post-index. |
| M' = 7 | Specifies XE-base relative addressing; i.e., EA = XE base + Y' + post-index. |

## Computing EA for Relative Instructions (Figure 2-3-4)

Following are the step-by-step procedures employed in computing the effective address for relative instructions. Included are procedures for relative direct and relative indirect addressing. Symbolic notations used in the Preface.

Relative Direct EA Computation (I = 0) - The following steps are executed:

a. Add the specified base to DISP (extended) dependent on mode (bits 06 through 08):

| Mode | Computation (Intermediate EA) |
|---|---|
| 3 | $00000 + DISP_e \rightarrow EA_i$ |
| 4 | $(PC) + DISP_e \rightarrow EA_i$ |
| 5 | $(XC) + DISP_e \rightarrow EA_i$ |
| 6 | $(XT) + DISP_e \rightarrow EA_i$ |
| 7 | $(XE) + DISP_e \rightarrow EA_i$ |

b. If bit 10 = 0, continue with step c.

If bit 10 (XB indicator) = 1, add index B to $EA_i$:

$$EA_i + (XB) \rightarrow EA_i$$

c. If bit 11 = 0, $EA_i \rightarrow EA_F$

If bit 11 (XA indicator) = 1, add index A to $EA_i$:

$$EA_i + (XA) \rightarrow EA_F$$

Examples of Direct EA Computation:

| | Instructions | | | | | | Computation Of EA (With Assumed Base And Index Address) |
|---|---|---|---|---|---|---|---|
| | OP | M | I (Bits) | XB (Bits) | XA (Bits) | DISP | |
| (1) | XX | 4 | 0 | 0 | 0 | 0030 | $(PC) = 04053$<br>$DISP_e = \underline{00030}$<br>$EA_F = 04103$ |
| (2) | XX | 6 | 0 | 1 | 0 | 7770 | $(XT) = 04300$<br>$DISP_e = \underline{77770}$ |

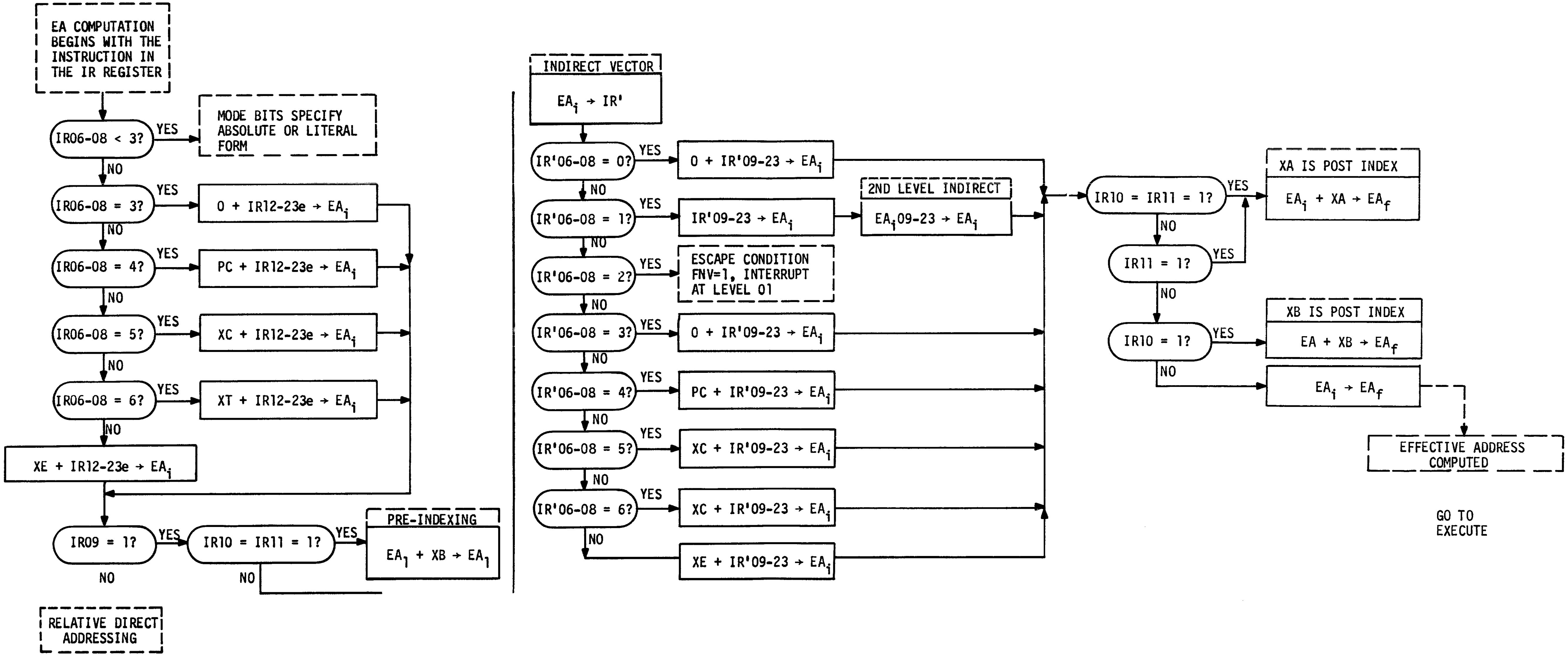


Figure 2-3-4. Computation of EA for Relative Indirect Instruction

| | Instructions | | | | | | Computation Of EA (With Assumed Base And Index Address) |
|---|---|---|---|---|---|---|---|
| | OP | M | I | XB | XA | DISP | |
| | | | (Bits) | | | | |
| (2 cont) | | | | | | | $EA_i = 04270$ |
| | | | | | | | $(XB) = \underline{00005}$ |
| | | | | | | | $EA_F = 04275$ |
| (3) | XX | 5 | 0 | 1 | 1 | 0003 | $(XC) = 04252$ |
| | | | | | | | $DISP_e = \underline{00003}$ |
| | | | | | | | $EA_i = 04255$ |
| | | | | | | | $(XB) = \underline{00005}$ |
| | | | | | | | $EA_i = 04262$ |
| | | | | | | | $(XA) = \underline{77776}$ |
| | | | | | | | $EA_F = 04260$ |

Relative Indirect EA Computation (I = 1) - The following steps are executed:

a. Add specified base to DISP (extended) dependent on mode:

| Mode | Computation (Intermediate EA) |
|---|---|
| 3 | $00000 + DISP_e \rightarrow EA_i$ |
| 4 | $(PC) + DISP_e \rightarrow EA_i$ |
| 5 | $(XC) + DISP_e \rightarrow EA_i$ |
| 6 | $(XT) + DISP_e \rightarrow EA_i$ |
| 7 | $(XE) + DISP_e \rightarrow EA_i$ |

b. Check for pre-indexing:

If bit 10 or bit 11 equals 0, no pre-index is added (continue with step c). If bit 10 and bit 11 (index indicators) both equal 1, add index B to $EA_i$:

$$EA_i + (XB) \rightarrow EA_i$$

c. Fetch indirect vector and check mode bits:

$$(EA_i) \rightarrow IR'$$

(1) If M' = 0, add 00000 to Y' to form new $EA_i$.

(2) If M' = 1, bits 09 through 23 of the memory word specified by Y' become the new $EA_i$:

$$(Y'\ 09\text{-}23) \rightarrow EA_i$$

This is the second-level indirect condition.

(3) If M' = 2, a fence violation occurs, the instruction is aborted, and a level-1 interrupt is invoked.

(4) If M' > 2, add specified base to Y' to form new $EA_i$:

| M' | Computation (Intermediate EA) |
|---|---|
| 3 | $00000 + Y' \rightarrow EA_i$ |
| 4 | $(PC) + Y' \rightarrow EA_i$ |
| 5 | $(XC) + Y' \rightarrow EA_i$ |
| 6 | $(XT) + Y' \rightarrow EA_i$ |
| 7 | $(XE) + Y' \rightarrow EA_i$ |

d. Check for post-indexing:

If bit 11 (index A indicator) = 1, add index A to $EA_i$:

$$EA_i + (XA) \rightarrow EA_F$$

If bit 11 = 0, check bit 10 (index B indicator).

If bit 10 = 1, add index B to $EA_i$:

$$EA_i + (XB) \rightarrow EA_F$$

If bit 10 = 0, $EA_i \rightarrow EA_F$

Examples of Indirect EA Computation:

| | Instructions | | | | | Computation of EA (With Assumed Base And Index Value) | |
|---|---|---|---|---|---|---|---|
| OP | M | I | XB | XA | DISP | | |
| | | (Bits) | | | | | |
| (1) | | | | | | | |
| XX | 4 | 1 | 1 | 0 | 0043 | (PC) | = 05531 |
| | | | | | | $DISP_e$ | = 00043 |
| | Indirect vector: | | | | | $EA_i$ | = 05574 (location of indirect vector) |
| | M' | | Y' | | | | |
| | 0 | | 00045 | | | Y' | = 00045 |
| | | | | | | (Zero Base) | = 00000 |
| | Use mode = 3 | | | | | $EA_i$ | = 00045 |
| | | | | | | (XB) | = 00010 |
| | | | | | | $EA_F$ | = 00055 |
| (2) | | | | | | | |
| XX | 6 | 1 | 1 | 1 | 7775 | (XT) | = 05750 |
| | | | | | | $DISP_e$ | = 77775 |

| Instructions<br>OP M I XB XA DISP<br>(Bits) | Computation of EA<br>(With Assumed Base<br>And Index Value) |
|---|---|
| (2 cont) | |
| Indirect vector: | $EA_i$ = 05745 |
| M' Y' | (XB) = 00010 |
| XX 1 77776 | $EA_i$ = 05755 (location of indirect vector) |
| 2nd level vector: | Y' = 77776 (location of 2nd level vector) |
| | $EA_i$ = 00100 (2nd level indirect) |
| XX X 00100 | (XA) = 00005 |
| | $EA_F$ = 00105 |
| (3) | |
| XX 5 1 0 1 0031 | (XC) = 05700 |
| | $DISP_e$ = 00031 |
| Indirect vector: | $EA_i$ = 05731 (location of indirect vector) |
| M' Y' | |
| 2 00000 | Instruction Aborted |
| | Fence Violation |
| | Level 1 interrupt |
| (4) | |
| XX 5 1 0 1 0031 | (XC) = 05700 |
| | $DISP_e$ = 00031 |
| Indirect vector: | $EA_i$ = 05731 (location of indirect vector) |
| M' Y' | |
| 4 77767 | Y' = 77767 |
| | (PC) = 05543 |
| | $EA_i$ = 05532 |
| | (XA) = 77771 |
| | $EA_F$ = 05523 |

### Two-Word Instructions

The FOX 1 CP provides several instructions that require a two-word format to fully define their functions. Each word of these instructions is handled separately and falls into one of the formats previously defined, except where specifically stated otherwise in Section 4 of this manual.

## DATA WORDS

Data words are used as instruction operands. Their memory addresses are specified by an instruction's effective addresses and their formats are defined by the operation that is to be executed using the data as an operand. Any memory word may be used as a data word.

Several types of data words are defined by the instruction execution logic. Each type has one or more formats. Three types of data are described in the following paragraphs. These are: bit data, character data, and numeric data.

### Bit Data

Bit data words represent 24 individual pieces of information. Each piece of information can be considered to represent I/O, on/off, yes/no, set/reset, present/absent, and so forth. Bit data is used when the choice of one condition out of a set of two possible conditions is significant in the context of a problem solution.

Operations on such data are logical rather than algebraic, and actions executed on one bit of a word do not carry over into adjoining bits.

Operations are provided by the instruction repertoire that can manipulate or sense individual bits of data words (BIT instruction), manipulate groups of bits (BYT instruction), perform logical arithmetic on full words (AND, OR, XOR instructions), or store selected portions of the words in memory (MST instruction).

Character (Alphanumeric) Data

Character data words represent alphanumeric data (i.e., numerals, letters of the alphabet, punctuation marks, and special symbols). In these words, individual characters are represented by a series of bits that forms words. A series of words is used to form complete alphanumeric texts.

For internal (central processor) character representation, the binary codes of characters and the word format are flexible according to need. Conversion of character codes between formats can be programmed to fit any application. Specific formats are primarily dependent on the character requirements of peripheral equipment used as the input source or output recipient of data.

Three of the more conventional character code formats are as follows:

a. Six-bit character codes/four characters per word

| Char. 1 | | | | | | Char. 2 | | | | | | Char. 3 | | | | | | Char. 4 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

In this format, a set of 64 different characters may be represented, with binary codes ranging from 000000 through 111111.

b. Eight-bit character codes/three characters per word

| Char. 1 | | | | | | | | Char. 2 | | | | | | | | Char. 3 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

This character data format is used in all standard software FOX 1 systems, including the assembler, compiler and real time executive system. This eight-bit code uses the USASCII-8 character convention. (Refer to Appendix A for the complete USASCII-8 code set.)

c. Twelve-bit character codes/two characters per word

| Char. 1 | | | | | | | | | | | | Char. 2 | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Twelve-bit characters are usually in the Hollerith code, which is the standard alphanumeric punched card code. These codes, when used with the standard FOX 1 software, are converted to eight-bit USASCII-8 code for system compatibility.

Numeric Data

Numeric data words represent arithmetic quantities used as operands by algebraic instructions. These words are in binary, 2's complement, notation to facilitate arithmetic operations in the central processor. Refer to Appendix D for additional information on 2's complement notation.

Four formats of numeric data are defined: single-precision and double-precision fixed point numbers; and single-precision and double-precision floating point numbers.

Fixed Point Numbers

Fixed point numbers represent integer values. Instructions are provided to perform algebraic operations using either short (single-precision) or long (double-precision) operands.

When a word (or double-word) is used as fixed point numeric data, the most significant or left-most bit (bit 00) of the data word specifies the algebraic sign of the number. When this sign bit is clear (bit 00 = 0), the entire (24- or 48-bit) number is positive. When the sign bit is set (bit 00 = 1), the entire number is negative. The remaining bits (23 bits for single-precision and 47 bits for double-precision) contain the magnitude for positive values or the 2's complement of the magnitude for negative values.

<u>Single-Precision (Short) Fixed Point Format</u> - The format of a single-precision (short) fixed point number is:

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SIGN | 23 bits of Magnitude (or 2's Complement) | | | | | | | | | | | | | | | | | | | | | | |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

The range of values that can be represented as single-precision fixed point numbers is:

$$(-8{,}388{,}608) \leq \text{number} \leq (+8{,}388{,}607)$$

or

$$-2^{23} \leq \text{number} \leq 2^{23} -1$$

Octally, this range is:

$$40000000 \leq \text{number} < 37777777$$

The formal definition of a single-precision fixed point number (held in the A register with the bits and their positions indicated as A00 through A23) is:

$$(-2^{23}) \times A00 + \sum_{B=1}^{23} (2^{23}-B) \times A_B$$

Two single-precision fixed point numbers are illustrated on the following page. Considered as integers, these numbers represent the values +5 and -5.

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

+ The Integer +5

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

- The Integer -5

Double-Precision (Long) Fixed Point Format - In some calculations, the single-precision value range may be insufficient. Instructions are included in the CP to perform algebraic operations on double-precision numbers. Such numbers are formed from two consecutive memory words (long or double-word).

The format of double-precision fixed point numbers is:

<table>
<tr><td>SIGN</td><td colspan="23">Most Significant 23 Bits of Magnitude (or 2's Complement)</td></tr>
<tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td></tr>
<tr><td colspan="24">Least Significant 24 Bits of Magnitude (or 2's Complement)</td></tr>
<tr><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td><td>32</td><td>33</td><td>34</td><td>35</td><td>36</td><td>37</td><td>38</td><td>39</td><td>40</td><td>41</td><td>42</td><td>43</td><td>44</td><td>45</td><td>46</td><td>47</td></tr>
</table>

The range of values that can be represented as double-precision fixed point numbers is:

$$(-140,737,488,355,328) \leq \text{number} \leq (+140,737,488,355,327)$$

or

$$-2^{47} \leq \text{number} \leq 2^{47}-1$$

Octally, this range is:

$$4000000000000000 \leq \text{number} \leq 3777777777777777$$

The formal definition of a double-precision fixed point number (held in the A and E registers with the bits and their positions indicated as A00 through A23 and E00 through E23) is:

$$(-2^{47}) \times A_0 + \sum_{B=1}^{23} (2^{47-B}) \times A_B + \sum_{B=0}^{23} (2^{23-B}) \times E_B$$

The two integers +5 and -5 are shown below in this format:

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |

Floating Point Numbers

There are two formats for floating point numbers: single-precision (short) and double-precision (long). Instructions are provided for algebraic operations using both formats.

A floating point number consists of two parts: a signed fraction (mantissa) and an exponent (characteristic). The value represented by a floating point number can be stated as:

$$N = F \times 2^{E}$$

where:

N = the floating point number.
F = the fraction portion.
E = an exponent that is:

Characteristic -32 (short form)

or

Characteristic -2048 (long form)

Features of floating point numbers are discussed in Appendix D.

Single-Precision (Short) Floating Point Format - A single-precision floating point number occupies one memory word and has the following format:

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S<br>F | S<br>C | C | | | | | F | | | | | | | | | | | | | | | | |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

where:

SF = the algebraic sign of the fraction

F = the magnitude of the fraction (or the 2's complement of the magnitude for a negative number)

SC = the algebraic sign of the characteristic

C = the characteristic of the floating point number (excess 32)

The range of the characteristic (C) is:

$$0 \leq C \leq 63 \text{ (in decimal notation)}$$

or

$$000000 \leq C \leq 111111 \text{ (in binary notation)}$$

This characteristic range means that the range of the exponent in the floating point notation:

$$N = F \times 2^{E} \ (= F \times 2^{C-32})$$

is:

$$-32 \leq E \leq +31$$

Thus, the characteristic is maintained equal to the exponent plus 32 (excess 32).

The range of the fraction (S,F) is:

$$-(1-2^{-17}) \leq (SF,F) \leq + (1-2^{-17})$$

$$(-0.99999237060546875) \leq (SF,F) \leq (+0.99999237060546875)$$

The range of single-precision floating point numbers is, therefore:

$$-(1-2^{-17}) \times 2^{31} \leq N \leq +(1-2^{-17}) \times 2^{31}$$

This range may also be represented in the following manner:

$$\left.\begin{array}{c} +(1-2^{-17}) \times 2^{31} \\ \downarrow \\ +1/2 \times 2^{-32} \end{array}\right\} \text{All possible positive values}$$

$$\begin{array}{cl} \downarrow & \qquad\qquad \text{The value zero} \end{array}$$

$$\left.\begin{array}{c} -1/2 \times 2^{-32} \\ \downarrow \\ -(1-2^{-17}) \times 2^{31} \end{array}\right\} \text{All possible negative values}$$

The binary formats of these numbers are shown in Table 2-3-1.

Table 2-3-1. Floating Point Short Notation

| DECIMAL NUMBER | FLOATING-PT. (SHORT) FORMAT |
|---|---|
| | S F   S C   C   F |
| $+(1-2^{-17}) \times 2^{31}$ | 0 111111 11111111111111111 |
| $+1/2 \times 2^{-32}$ | 0 000000 10000000000000000 |
| 0 | 0 000000 00000000000000000 |
| $-1/2 \times 2^{-32}$ | 1 111111 10000000000000000 |
| $-(1-2^{-17}) \times 2^{31}$ | 1 000000 00000000000000001 |

It should be noted that all positive, normalized values within the floating point range that are exact powers of 2 have the same fraction configuration:

0 CCCCCC 10000000000000000

The value of the characteristic differentiates between the represented fractional values. An increase of 1 in the characteristic doubles the value of the number; a decrease of 1 halves the values.

Similarly, the negative representation of floating point values whose absolute values are exact powers of 2 have the configuration:

1 CCCCCC 10000000000000000

Here, again, the characteristic identifies the exact value represented.

Additional floating point numbers are shown in Appendix D.

The formal definition of a single-precision fixed point value (held in the A register with the bits and their positions indicated as A00 through A23) is:

$$\left[(-2^{0}) \times A_0 + \sum_{B=7}^{23} (2^{6-B}) \times A_B\right] \times 2^{\left[\sum_{B=1}^{6}\left((2^{6-B}) \times (A_B \oplus A_0)\right) -32\right]}$$

$\oplus$ - indicates logical exclusive OR

<u>Double-Precision (Long) Floating Point Format</u> - A double-precision floating point number occupies two memory words and has the following format:

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S<br>F | S<br>C | C | | | | | | | | | | | MF | | | | | | | | | | |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| LF | | | | | | | | | | | | | | | | | | | | | | | |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |

where:

S = The algebraic sign of the fraction

MF = the most significant 11 bits of the magnitude of fraction (or 2's complement of the magnitude for negative numbers)

LF = the least significant 24 bits of the magnitude of the fraction (or 2's complement of the magnitude for negative numbers)

C = the characteristic of the floating point number

The range of the characteristic (C) is:

$$0 \leq C \leq 4095$$

or

$$000000000000 \leq C \leq 111111111111$$

This characteristic range means that the range of the exponent in the floating point notation:

$$N = F \times 2^{E} \; (=F \times 2^{C} - 2048)$$

is:

$$-2048 \leq E \leq 2047$$

The range of the fraction (SF, MF, LF) is:

$$-(1-2^{-35}) \leq (SF, MF, LF) \leq (+1-2^{-35})$$

Thus, the range of double-precision floating point number is:

$$-(1-2^{-35}) \times 2^{2047} \leq N \leq +(1-2^{-35}) \times 2^{2047}$$

This range may also be represented in the following manner:

$$\left.\begin{array}{c} +(1-2^{-35}) \times 2^{2047} \\ \downarrow \\ +1/2 \times 2^{-2048} \end{array}\right\} \quad \text{All possible positive values}$$

$$\downarrow \qquad \text{The value zero}$$

$$\left.\begin{array}{c} -1/2 \times 2^{-2048} \\ \downarrow \\ -(1-2^{-35}) \times 2^{2047} \end{array}\right\} \quad \text{All possible negative values}$$

The binary formats of these values are shown in Table 2-3-2.

Table 2-3-2. Floating Point Long Notation

| DECIMAL NUMBER | FLOATING POINT (LONG) FORMAT | | | | |
|---|---|---|---|---|---|
| | S<br>F | S<br>C | C | MF | LF |
| $+(1-2^{-35}) \times 2^{2047}$ | 0 | | 111111111111 | 11111111111 | 111111111111111111111111 |
| $+1/2 \times 2^{2048}$ | 0 | | 000000000000 | 10000000000 | 000000000000000000000000 |
| 0 | 0 | | 000000000000 | 00000000000 | 000000000000000000000000 |
| $-1/2 \times 2^{2048}$ | 1 | | 111111111111 | 10000000000 | 000000000000000000000000 |
| $-(1-2^{-35}) \times 2^{2047}$ | 1 | | 000000000000 | 00000000000 | 000000000000000000000001 |

As with single-precision floating point numbers, all positive values within this range that are exact powers of 2 have the same fractional configuration:

0 CCCCCCCCCCCC 10000000000000000000000000000000000

The value of the characteristic differentiates between the represented fractional values. An increase of 1 in the characteristic doubles the value of the number; a decrease of 1 halves the value.

Similarly, the negative representation of values whose absolute values are exact powers of 2 have the configuration:

1 CCCCCCCCCCCC 10000000000000000000000000000000000

The characteristic signifies the exact value represented.

Additional floating point numbers are shown in Appendix D.

The formal definition of a double-precision floating point value (held in the A and E registers with the bits and their positions indicated as A00 through A23 and E00 through E23) is:

$$\left[(-2^{0}) \times A_{0} + \sum_{B=13}^{23} (2^{12-B}) \times A_{B} + \sum_{B=0}^{23} (2^{-12-B})\ E_{B}\right] \times 2^{N}$$

$$N = \left[\sum_{B=1}^{12} \left((2^{12-B})\ (A_{B} \oplus A_{0})\right) -2048\right]$$

$\oplus$ - indicates logical exclusive OR. Thus, the characteristic is maintained equal to the exponent

# 4/INSTRUCTION REPERTOIRE

Stored instructions executed by the FOX 1 Central Processor are separated into the following functional groups for description in this section of the reference manual: load; store; logical; rotate, shift, and normalize; fixed point arithmetic; floating point arithmetic; index control; branch; linkage; compare with memory; bit manipulation; byte manipulation; special; and programmed input/output.

Each instruction and most microprogrammed instructions (extended mnemonics) are discussed in the following terms:

a. Name: The instruction name concisely describes the operation performed by the instruction.

b. Mnemonic: The mnemonic code accepted by the FOX 1 Assembler Program to generate the operation code. Extended mnemonics are given where appropriate and accepted by the assembler program.

c. Op Code: The operation code (bits 00 through 05 of the instruction word) represented by two octal digits.

d. Timing: An estimate of the time required to execute the instruction is given, based on a 0.96-microsecond memory cycle, inactive channel IO, no tracing (ASE = 0), no violations (such as FNV), and an absolute direct-addressing mode. To estimate execution timing for other addressing modes, the following correction factors must be applied to the listed time.

| Addressing Mode | Correction |
|---|---|
| Absolute direct (M = 0) | 0.00 μsec (Ref.) |
| Absolute indirect (M = 1) | +0.96 μsec |
| Literal (M = 2) | -0.32 μsec |
| Relative (M = 3,4,5,6,7) | |
| direct, no indexing | +0.00 μsec |
| direct, with indexing | +0.64 μsec |
| indirect, original base (M' = 0), no indexing | +1.60 μsec |
| indirect, original base (M' = 0), with indexing | +1.92 μsec |
| double indirect (M' = 1) | +2.24 μsec |
| indirect, new base (M' = 3,4,5,6,7), no indexing | +1.28 μsec |
| indirect, new base (M' = 3,4,5,6,7), with indexing | +1.60 μsec |

Two-word instructions (BIT, CWM, and MOV, but not BSR) have two addresses and may require two addressing corrections to estimate execution time.

e. Literal Class: Instructions respond to literal mode addressing in one of three classes specified here.

(1) Literal class 1 - used only by instructions that normally require an operand from memory. In this class, bits 09 through 23 of the instruction word are right-justified in either single-word or double-word format, with more significant bits set to correspond with the contents of bit 09 to form an immediate operand.

(2) Literal class 2 - used by instructions that store data or effect a branch out of program sequence. This class is equivalent to the direct-address mode (M = 0).

(3) Literal class special - used only by the BRU, BSR, and BSP instructions. Special handling of the literal mode by these instructions is included with the instruction descriptions.

f. Execution Counter: The mnemonic of the counting register that controls the sequence of operations in executing the instruction.

g. Registers Affected: Registers, whose contents are involved in (not necessarily changed by) execution of the instruction, are listed.

h. Errors Possible: Hardware faults or program violations that can be detected by the CP logic circuits during normal instruction execution are listed here.

i. Symbolic Notation: A pseudo-Boolean algebra notation is used to summarize the logical operations executed by instructions. Refer to Symbols and Notation in Section 1 for an explanation of this notation.

j. Description: The operations performed by the FOX 1 CP in executing an instruction are described in detail, and include any special circumstances.

All instructions are assumed to be contained in one word of core memory, unless otherwise stated.

Two-word instructions require the use of two full words to define their functions. The two-word instructions are as follows:

| Instruction Name | Mnemonic |
|---|---|
| Branch and Save Region | BSR |
| Compare With Memory | CWM |
| Multiple Move | MOV |

| Instruction Name | Mnemonic |
|---|---|
| Byte Manipulation** | BYT |
| Bit Manipulation | BIT |
| Masked Store** | MST |

NOTE

**BYT and MST actually use two words to specifiy their functions; however, one of the words is in the E register. MST is described with the store instructions; BYT is described as a group by itself because of the large amount of decoding and microprogramming it provides.

The BSR, CWM, MOV, and BIT instructions occupy two contiguous storage locations. In decoding these instructions, the central processor transfers the first word from memory, decodes and determines its effective address, and then increments the PC register for access to the second word. After the second word is decoded and the effective address has been determined, the PC register is incremented again.

Two-word instructions cannot be skipped by a programmed skip instruction.

Example: If the comparison (CXA) is satisfied in the sequence:

```
CXA          ]1 word
MOV          ]2 words
```

the second word of the MOV instruction would be executed (instead of the first word). Care should be taken to avoid this condition.

## LOAD INSTRUCTIONS

Load instructions transfer single-word or double-word information from memory into one of the CP addressable registers. Some load instructions, because of their special functions, fit into two different instruction groupings. The descriptions of these instructions are included in the alternate group, denoted below by an asterisk (*). Only those affecting the A and E registers are described here. The load instructions are as follows:

| Instruction Name | Mnemonic |
|---|---|
| Load A Register | LDA |
| Load E Register | LDE |
| Load Long | LDL |
| Load Logical Complement | LLC |
| *Exchange A Register and Memory | EAM |
| *Load Index A | LXA |
| *Load Index B | LXB |

Name: Load A register

Mnemonic: LDA

Op Code: 53

Timing: 1.92 μsec

Literal Class: 1

Execution Counter: CD

Registers Affected: A, CPI00, CPI01, (EA)

Errors Possible: None

Symbolic Notation: (EA) → A
0 → FPU
0 → FPL

Description: The contents of the location specified by the effective address are loaded into (replace the previous contents of) the A register, and both the FPU (CPI00) and FPL (CPI01) are cleared.

Name: Load E Register

Mnemonic: LDE

Op Code: 52

Timing: 1.92 μsec

Literal Class: 1

Execution Counter: CD

Registers Affected: E, (EA)

Errors Possible: None

Symbolic Notation: (EA) → E

Description: The contents of the location specified by the effective address are loaded into (replace the previous contents of) the E register.

Name: Load Long

Mnemonic: LDL

Op Code: 51

Timing: 2.88 μsec

Literal Class: 1

Execution Counter: CC

Registers Affected: A, E, CPI00, CPI01

Errors Possible: None

Symbolic Notation: (EA) → A
(EA + 1) → E
0 → FPU
1 → FPL

Description: The contents of the location specified by the effective address are transferred into (replace the previous contents of) the A register, and the contents of the next location above that loaded into the A register (EA + 1) are transferred into the E register. Central processor indicator register bit 00 is cleared (0 → FPU) and bit 01 is set (1 → FPL).

If the literal mode is specified, the immediate operand in bits 09 through 23 of the instruction word is loaded into corresponding bits of the E register, and the content of bit 09 is extended left into bits 00 through 08 of the E register and into all bits of the A register.

Examples:

| OP | M | Y | Results |
|---|---|---|---|
| 51 | 2 | 00005 | A = 00000000; E = 00000005 |
| 51 | 2 | 77775 | A = 77777777; E = 77777775 |
| 51 | 0 | 77774; XA = 12345 | A = 00012345; E = 00012345 |

Name: Load Logical Complement

Mnemonic: LLC

Op Code: 04

Timing: 1.92 μsec

Literal Class: 1

Execution Counter: CD

Registers Affected: A, (EA)

Errors Possible: None

Symbolic Notation: $(\overline{EA}) \rightarrow A$ or $(EA)- \rightarrow A$

Description: The 1's complement (inverse) of the contents of the location specified by the effective address are transferred into (replace the previous contents of) the A register.

Examples:

a. If LLC is executed on a word containing 41772356, the result in the A register is 36005421.

b. If the instruction (literal mode):

| OP | M | Y |
|---|---|---|
| 04 | 2 | 00034 |

is executed, the result in the A register is 77777743.

## STORE INSTRUCTIONS

Store instructions affect the portion of memory storage specified by their effective address and operation code. Both single- and double-format words might be affected. Some store instructions also fit into other instruction groupings. The store instructions are identified in the following list. An asterisk (*) denotes those store instructions that are described with their alternate group. All store instructions can cause fence violations. Such a violation sets the FNV bit of the central processor status register and initiates a level 01 interrupt.

| Instruction Name | Mnemonic |
|---|---|
| Store A Register | STA |
| Store E Register | STE |
| Store Long | STL |
| Masked Store | MST |
| Exchange A Register and Memory | EAM |
| *Store Normalized and Rounded | SNR |
| *Store Index A | SXA |
| *Store Index B | SXB |

Name: Store A Register

Mnemonic: STA

Op Code: 44

Timing: 2.24 μsec

Literal Class: 2

Execution Counter: CD

Registers Affected: A, (EA)

Errors Possible: None

Symbolic Notation: A → (EA)

Description: The contents of the A register are stored in (replace the previous contents of) the location specified by the effective address.

Name: Store E Register

Mnemonic: STE

Op Code: 46

Timing: 2.24 μsec

Literal Class: 2

Execution Counter: CD

Registers Affected: E, (EA)

Errors Possible: None

Symbolic Notation: E → (EA)

Description: The contents of the E register are stored in (replace the previous contents of) the location specified by the effective address.

Name: Store Long

Mnemonic: STL

Op Code: 45

Timing: 3.20 μsec

Literal Class: 2

Execution Counter: CC

Registers Affected: A, E, (EA), (EA + 1)

Errors Possible: None

Symbolic Notation: A → (EA)
E → (EA + 1)

Description: The double word contained in the A and E registers is stored in (replaces the previous contents of) two consecutive locations, the first of which is specified by the effective address.

Name: Masked Store

Mnemonic: MST

Op Code: 55

Timing: 2.88 μsec

Literal Class: 2

Execution Counter: CD

Registers Affected: A, E, (EA)

Errors Possible: None

Symbolic Notation: $[A \,\&\, E] + [(EA) \,\&\, \overline{E}] \rightarrow (EA)$

on a bit-by-bit basis: If $E_j = 1$, then $A_j \rightarrow (EA_j)$
If $E_j = 0$, then $(EA_j) \rightarrow (EA_j)$

Description: The contents of the E register serve as a mask for storing the contents of the A register in the location specified by the effective address. Bits of the E register containing ones enable the storing of corresponding bits of A register data in corresponding bits of the storage location. Bits of the E register containing zeros inhibit change of corresponding bit data in the storage location.

Example:

| Before Execution | After Execution |
|---|---|
| A = 01234567 | A = 01234567 |
| E = 07070707 | E = 07070707 |
| (EA) = XXXXXXXX | (EA) = X1X3X5X7 |

Intermediate Results

A & E = 01030507 (determine masked bits of A)
(EA) & $\overline{E}$ = X0X0X0X0 (determine unmasked bits of A)
01030507 + X0X0X0X0 = X1X3X5X7 (final result)

Name: Exchange A Register and Memory

Mnemonic: EAM

Op Code: 54

Timing: 2.88 μsec

Literal Class: 2

Execution Counter: CD

Registers Affected: A, (EA)

Errors Possible: None

Symbolic Notation: A → (EA) and (EA) → A

Description: The contents of the A register and the contents of the location specified by the effective address are exchanged.

LOGICAL INSTRUCTIONS

All logical instructions are performed between corresponding bits of the operand in the A register and the operand specified by the instruction (either a literal value or the contents of the effective address). The results of any logical instruction replace the contents of the A register.

The operands used by logical instructions are in the bit data format.

The logical instructions are:

| Instruction Name | Mnemonic |
|---|---|
| Logical AND | AND |
| Inclusive OR | IOR |
| Exclusive OR | XOR |

Name: Logical AND

Mnemonic: AND

Op Code: 02

Timing: 1.92 μsec

Literal Class: 1

Execution Counter: CD

Registers Affected: A, (EA)

Errors Possible: None

Symbolic Notation: A & (EA) → A

Description: The logical product (or Boolean AND function) of the contents of the location specified by the effective address and the contents of the A register are formed and replace the previous contents of the A register. The result in the A register is a 1 whenever the corresponding bit positions of the two words both contain ones; otherwise, the result is 0. Corresponding bit settings and results are:

| Initial Bit Value | | Result In |
|---|---|---|
| A | (EA) | A |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Example:

| Before Execution | After Execution |
|---|---|
| A = 55550202 | A = 00000202 |
| (EA) = 00007777 | (EA) = 00007777 |

Name: Inclusive OR

Mnemonic: IOR

Op Code: 03

Timing: 1.92 μsec

Literal Class: 1

Execution Counter: CD

Registers Affected: A, (EA)

Errors Possible: None

Symbolic Notation: A + (EA) → A

Description: The logical sum (or Boolean inclusive OR function) of the contents of the location specified by the effective address and the contents of the A register are formed and replace the previous contents of the A register. The result is a 1 in the A register wherever the corresponding bit positions of either word contain a 1. Corresponding bit settings and results are:

| Initial Bit Value | | Result In |
|---|---|---|
| A | (EA) | A |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Example:

| Before Execution | After Execution |
|---|---|
| A = 01020304 | A = 13365774 |
| (EA) = 12345670 | (EA) = 12345670 |

Name: Exclusive OR

Mnemonic: XOR

Op Code: 05

Timing: 1.92 μsec

Literal Class: 1

Execution Counter: CD

Registers Affected: A, (EA)

Errors Possible: None

Symbolic Notation: A $\oplus$ (EA) → A

Description: The logical half-add (or Boolean exclusive OR function) of the contents of the location specified by the effective address and the contents of the A register are formed and replace the previous contents of the A register. The result is a 1 in the A register wherever the corresponding bit of the two words are not equal. Corresponding bit settings and results are:

| Initial Bit Value | | Result In |
|---|---|---|
| A | (EA) | A |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Example:

| Before Execution | After Execution |
|---|---|
| A = 53262531 | A = 24512531 |
| (EA) = 77770000 | (EA) = 77770000 |

## ROTATE, SHIFT, AND NORMALIZE INSTRUCTIONS

Instructions in this class are:

| Instruction Name | Mnemonic |
|---|---|
| Rotate Left E Register | RLE |
| Shift and Rotate | SHF |
| Normalize Short | NMS |
| Normalize Long | NML |

These instructions use normal addressing. The Shift and Rotate instruction is microprogrammed to produce five rotate instructions and nine shift instructions, each with an individual extended mnemonic. These extended mnemonics are assumed to be in the literal mode when assembled by the FOX 1 Assembler Program.

### Rotate Operations

Rotate operations (bit 15 = 0; bit 14 of the SHF instruction is ignored) consider the register or registers affected by the operation to be circular; i.e., for rotate-short operations, bit 00 and bit 23 of the A register are considered adjacent bits. For rotate-long operations, bit 00 of the A register and bit 23 (bit 47, double word) of the E register are adjacent, as are bit 23 of the A register and bit 00 (bit 24) of the E register.

No information is lost in a rotate operation, nor can overflow occur.

## Shift Operations

Shift operations (bit 15 = 1) accomplish left or right shifts on the contents of the A register or the A and E registers taken as a double-precision word. A right shift causes information to be lost off the right end of the word. A left shift causes information to be lost off the left end of the word. Both logical and arithmetic shifts are provided. Logical shifts consider bit 00 of the shifted word or double word in the same way as other bits. Vacated bits in a logical shift are filled with zeros, whether on the left or right end of the register.

Arithmetic shifts are affected by the sign bit of the shifted word or double word. On right arithmetic shifts, the sign bit is recopied into bit 00 of the shifted word or double word, as well as being propagated right into vacated bits. On left arithmetic shifts, zero bits fill into bit position 23; bits shifted out of bit position 01 are lost. Initiation of an arithmetic left shift resets the overflow indicator (CPS18 = 0). The first occurrence of the bit shifted out of bit position 01, being different from the sign bit, causes the overflow indicator to be set (CPS18 = 1). This signals the loss of significant bits. The shift continues through the specified number of bit positions. The setting of the overflow indicator causes a level 01 interrupt. Therefore, to avoid unnecessary interrupts, logical shifts rather than arithmetic shifts should be used when loss of information does not matter.

## Normalize Operations

Normalize instructions aid in converting fixed point data to floating point data. Two normalize instructions are provided:

a. Normalize Short (NMS), which acts on single-precision data
b. Normalize Long (NML), which acts on double-precision data

Normalize instructions do not act on negative data. If bit 00 of the A register initially contains a 1, normalize instructions cause the overflow indicator to be set (CPS18 = 1), and the instruction is aborted and a level 01 interrupt occurs.

Normalize instructions shift the contents of the affected register(s) (A register for NMS: A and E registers for NML) left until bit 00 and bit 01 in the A register are different (i.e., bit 00 = 0, bit 01 = 1). Thus, the most significant bit is in bit position 01. A shift count is maintained, specifying the number of bits through which shifting was required. This shift count then replaces bits 18 through 23 of the contents of the effective address. The result of this action provides the essential information for converting the value in the register(s) to a floating point value.

If the value to be normalized is zero, maximum shifting occurs. A shift count of $27_8$ (or $23_{10}$) for single-precision operations, or $57_8$ (or $47_{10}$) for double-precision operations replaces the contents of the effective address bit 18 through 23. Bits 00 through 17 of the contents of the effective address are not affected by normalize instructions.

Name: Rotate Left E Register
Mnemonic: RLE
Op Code: 40
Timing: 2.24 μsec plus 0.32 μsec for each shift
Literal Class: 1
Execution Counter: CA
Registers Affected: E
Errors Possible: None
Symbolic Notation: For each shift $E00 \rightarrow E23$
$En + 1 \rightarrow En$; where $0 \leq n \leq 22$

Description: The 24 bits of the E register are rotated left the number of bit positions specified by bits 18 through 23 of the instruction operand. Bits rotated out of position 00 enter bit position 23. No information is lost.

Example: RLE 12 bits

| Before Execution | After Execution |
|---|---|
| E = 01234567 | E = 45670123 |

Name: Shift and Rotate

Mnemonic: SHF

Op Code: 43

Timing: 2.24 μsec plus 0.32 μsec for each shift

Literal Class: 1

Execution Counter: CA

Registers Affected: A, E

Errors Possible: None

Symbolic Notation: Not applicable

Description: All rotate and shift instructions, except Rotate Left E, are microprogrammed SHF commands. The SHF operand (either a literal operand or the location specified by the effective address) specifies both the type of shift or rotate to be executed and the number of bit positions to be shifted. An SHF operand has the following format:

| NOT USED | | | | | | | | | | | | | | TYPE | | | | N | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

where:

Bit 14: specifies a logical (bit 14 = 1) or arithmetic (bit 14 = 0) operation.

Bit 15: specifies shift (bit 15 = 1) or rotate (bit 15 = 0). If bit 15 = 0, bit 14 is ignored.

Bit 16: specifies the use of long (A and E registers combined) format (bit 16 = 1) or short (A register alone) format (bit 16 = 0).

Bit 17: specifies left (bit 17 = 1) or right (bit 17 = 0) movement.

Bits 18 through 23: specify the number of bit positions to be shifted or rotated.

Extended mnemonics of SHF are: RRS, RRL, RLS, RLL, ARS, LRS, ALS, LLS, ARL, LRL, ALL, and LLL. The first four of these are rotate commands and the last eight are shift commands.

Name: Rotate Right Short
Mnemonic: RRS
Op Code: 43, Type: X0
Timing: 2.24 μsec plus 0.32 μsec for each shift
Literal Class: 1
Execution Counter: CA
Registers Affected: A
Errors Possible: None
Symbolic Notation: For each shift A23 → A00
$An \rightarrow An + 1$; where $0 \leq n \leq 22$

Description: The 24-bit contents of the A register are rotated right (from more significant bit towards less significant bit) the number of bit positions specified in bits 18 through 23 of the instruction operand.

Example: RRS 3 bits

| Before Execution | After Execution |
|---|---|
| A = 01234567 | A = 70123456 |

Name: Rotate Right Long

Mnemonic: RRL

Op Code: 43, Type: X2

Timing: 2.24 μsec plus 0.32 μsec for each shift

Literal Class: 1

Execution Counter: CA

Registers Affected: A, E

Errors Possible: None

Symbolic Notation: For each bit rotation $A, E47 \rightarrow A, E00$
$A, En \rightarrow A, En + 1;$
where $0 \leq n \leq 46$

Description: The 48-bit double word in the A and E registers is rotated right the number of bit positions specified in bits 18 through 23 of the instruction operand.

Example: RRL 6 bits

| Before Execution | After Execution |
|---|---|
| A,E = 0123456776543210 | A,E = 1001234567765432 |

Name: Rotate Left Short

Mnemonic: RLS

Op Code, 43, Type: X1

Timing: 2.24 μsec plus 0.32 μsec for each shift

Literal Class: 1

Execution Counter: CA

Registers Affected: A

Errors Possible: None

Symbolic Notation: For each shift A00 → A23

$An + 1 \rightarrow An$; where $0 \leq n \leq 22$

Description: The 24-bit contents of the A register are rotated left (from less signnificant bit towards more significant bit) the number of bit positions specified in bits 18 through 23 of the instruction operand.

Example: RLS 6 bits

| Before Execution | After Execution |
|---|---|
| A = 01234567 | A = 12345670 |

Name: Rotate Left Long

Mnemonic: RLL

Op Code: 43, Type: X3

Timing: 2.24 μsec plus 0.32 μsec for each shift

Literal Class: 1

Execution Counter: CA

Registers Affected: A, E

Errors Possible: None

Symbolic Notation: For each shift A, E00 → A, E47

A, $En + 1 \rightarrow$ A, $En$; where $0 \leq n \leq 46$

Description: The 48-bit double word in the A and E registers is rotated left the numbers of bit positions specified in bit 18 through 23 of the instruction operand.

Example: RLL 6 bits

| Before Execution | After Execution |
|---|---|
| A,E = 0123456776543210 | A,E = 2345677654321001 |

Name: Shift Right Short - Arithmetic

Mnemonic: ARS

Op Code: 43, Type: 04

Timing: 2.24 μsec plus 0.32 μsec for each shift

Literal Class: 1

Execution Counter: CA

Registers Affected: A

Errors Possible: None

Symbolic Notation: For each shift A00 → A00
A00 → A01
An → An + 1, where $0 \leq n \leq 22$
A23 is lost

Description: The contents of the A register shifts right the number of bit positions specified by the instruction operand. The sign (A00) is maintained and fills bit position 01 for each shift. Bits shifted out of position 23 are lost.

Example: ASR 5 bits

| Before Execution | After Execution |
|---|---|
| A = 77773451 | A = 77777671 |
| A = 00345665 | A = 00007135 |

Name: Shift Right Short-Logical

Mnemonic: LRS

Op Code: 43, Type: 14

Timing: 2.24 μsec plus 0.32 μsec for each shift

Literal Class: 1

Execution Counter: CA

Registers Affected: A

Errors Possible: None

Symbolic Notation: For each shift 0 → A00
An → An + 1; where $0 \le n \le 22$
A23 is lost

Description: The contents of the A register shift right the number of bit positions specified by the instruction operand. For each shift, a zero enters A00 (the sign is not preserved) and the data in A23 are lost.

Example: LRS 3 bits

| Before Execution | After Execution |
|---|---|
| A = 73562105 | A = 07356210 |

Name: Shift Left Short - Arithmetic
Mnemonic: ALS
Op Code: 43, Type: 05
Timing: 2.24 μsec plus 0.32 μsec for each shift
Literal Class: 1
Execution Counter: CA
Registers Affected: A, CPS18
Errors Possible: Accumulator overflow (OV)
Symbolic Notation: For each shift 0 → A23
A00 → A00
If A00 ≠ A01, then 1 → CPS18
A01 is lost
An + 1 → An; where $1 \le n \le 22$

Description: The contents of the A register (bits 01 through 23) shift left the number of bit positions specified by the instruction operand. The sign bit (A00) is not shifted. For each shift, a zero enters A23 and data shifted out of A01 are lost. If the content of bit 01 is different from the content of the sign bit at any time, the accumulator overflow bit of the central processor status register is set (1 → CPS18). The instruction runs to completion even after the overflow indicator is set.

Example: ALS 6 bits

| Before Execution | After Execution |
|---|---|
| A = 00001234 | A = 00123400 |
| A = 66123456 | A = 52345600 and CPS18 = 1 |

Name: Shift Left Short - Logical

Mnemonic: LLS

Op Code: 43, Type: 15

Timing: 2.24 μsec plus 0.32 μsec for each shift

Literal Class: 1

Execution Counter: CA

Registers Affected: A

Errors Possible: None

Symbolic Notation: For each shift 0 → A23
An + 1 → An; where $0 \leq n \leq 22$
A00 is lost

Description: The contents of the A register shift left the number of bit positions specified by the instruction operand. For each shift, a 0 enters A23 and data shifted out of A00 are lost (the sign bit is not preserved).

Example: LLS 6 bits

| Before Execution | After Execution |
|---|---|
| A = 77776666 | A = 77666600 |

Name: Shift Right Long - Arithmetic

Mnemonic: ARL

Op Code: 43, Type: 06

Timing: 2.24 μsec plus 0.32 μsec for each shift

Literal Class: 1

Execution Counter: CA

Registers Affected: A, E

Errors Possible: None

Symbolic Notation: For each shift A,E00 → A,E01
A,En → A,En+1; where 0 ≤ n ≤ 46
A,E47 is lost

Description: The 48-bit double word in the A and E registers shifts right the number of bit positions specified by the instruction operand. The sign (A00) is maintained and fills bit position 01 (A01) for each shift. Bits shifted out of position 47 (E23) are lost.

Example: ARL 3 bits

| Before Execution | After Execution |
|---|---|
| A,E = 0005673112345670 | A,E = 0000567311234567 |
| A,E = 7777777676543210 | A,E = 7777777767654321 |

Name: Shift Right Long - Logical

Mnemonic: LRL

Op Code: 43, Type: 16

Timing: 2.24 μsec plus 0.32 μsec for each shift

Literal Class: 1

Execution Counter: CA

Registers Affected: A, E

Errors Possible: None

Symbolic Notation: For each shift 0 → A,E00
A,En → A,En + 1; where 0 ≤ n ≤ 46
A,E47 is lost

Description: The 48-bit double word in the A and E registers shifts right the number of bit positions specified by the instruction operand. For each shift, a zero enters the sign position (A00) and data shifted out of position 47 (E23) are lost.

Example: LRL 6 bits

| Before Execution | After Execution |
|---|---|
| A,E = 7654321001234567 | A,E = 0076543210012345 |

Name: Shift Left Long-Arithmetic
Mnemonic: ALL
Op Code: 43, Type: 07
Timing: 2.24 μsec plus 0.32 μsec for each shift
Literal Class: 1
Execution Counter: CA
Registers Affected: A, E, CPS18
Errors Possible: Accumulator overflow (OV)
Symbolic Notation: For each shift A,E00 → A,E00
0 → A,E47
If A,E00 ≠ A,E01, then 1 → CPS18
A,E01 is lost
A,En + 1 → A,En; where $1 \leq n \leq 46$

Description: The double-precision fixed point number in the A and E registers (bits 01 through 47) shifts left the number of bit positions specified by the instruction operand. The sign bit (A00) is preserved. For each shift, a 0 enters bit position 47 (E23) and data shifted out of position 01 (A01) are lost. If the content of bit 01 is different from the content of the sign bit at any time, the accumulator overflow bit of the central processor status register is set (1 → CPS18). The instruction runs to completion even after the overflow indicator is set.

Example: ALL 6 bits

| Before Execution | After Execution |
|---|---|
| A,E = 0012345676543210 | A,E = 1234567654321000 |
| A,E = 0123456765432100 | A,E = 2345676543210000 and CPS18 = 1 |

Name: Shift Left Long-Logical

Mnemonic: LLL

Op Code: 43, Type: 17

Timing: 2.24 μsec or 0.32 μsec for each shift

Literal Class: 1

Execution Counter: CA

Registers Affected: A, E

Errors Possible: None

Symbolic Notation: For each shift 0 → A,E47
A,En + 1 → A,En; where $0 \leq n \leq 46$
A,E00 is lost

Description: The 48-bit double word in the A and E registers shifts left the number of bit positions specified by the instruction operand. For each shift, a zero enters bit position 47 (E23) and data shifted out of position 00 are lost (the sign bit is not preserved).

Example: LLL 18 bits

| Before Execution | After Execution |
|---|---|
| A,E = 4123456007654321 | A,E = 6007654321000000 |

Name: Normalize Short

Mnemonic: NMS

Op Code: 41

Timing: 3.20 μsec plus 0.32 μsec for each shift

Literal Class: 2

Execution Counter: CA

Registers Affected: A, (EA18-23), CPS18

Errors Possible: Accumulator overflow (OV)

Symbolic Notation: For each shift An + 1 → An; where $0 \le n \le 23$

A00 → A00

0 → A23

A01 is lost

SHC + 1 → SHC

Description: The contents of the A register are normalized and the shift count replaces the contents of bits 18 through 23 of the location specified by the effective address. If the sign bit is set (A00 = 1) initially, the accumulator overflow indicator of the central processor status register is set (1 → CPS18) and no shifting occurs. In this case execution time is 1.92 μsec. If the word to be normalized initially contains all zeros, the maximum shifting occurs and a shift count of $27_8$ is stored. In this case execution time is 10.56 μsec. Normally, left-shifting occurs until A01 contains a 1. The functions performed by execution of the NMS instruction are shown in Figure 2-4-1.

Example:

| Before Execution | After Execution |
|---|---|
| A = 00356700 | A = 35670000 |
| (EA) = XXXXXXXX | (EA) = XXXXXX06 |

The following series of instructions normalizes a positive integer contained in the A register and creates a floating-point number in the A register for the result. These instructions do not take into consideration the possible loss of significant bits. Symbolic notations, including storage reference addresses, are used:

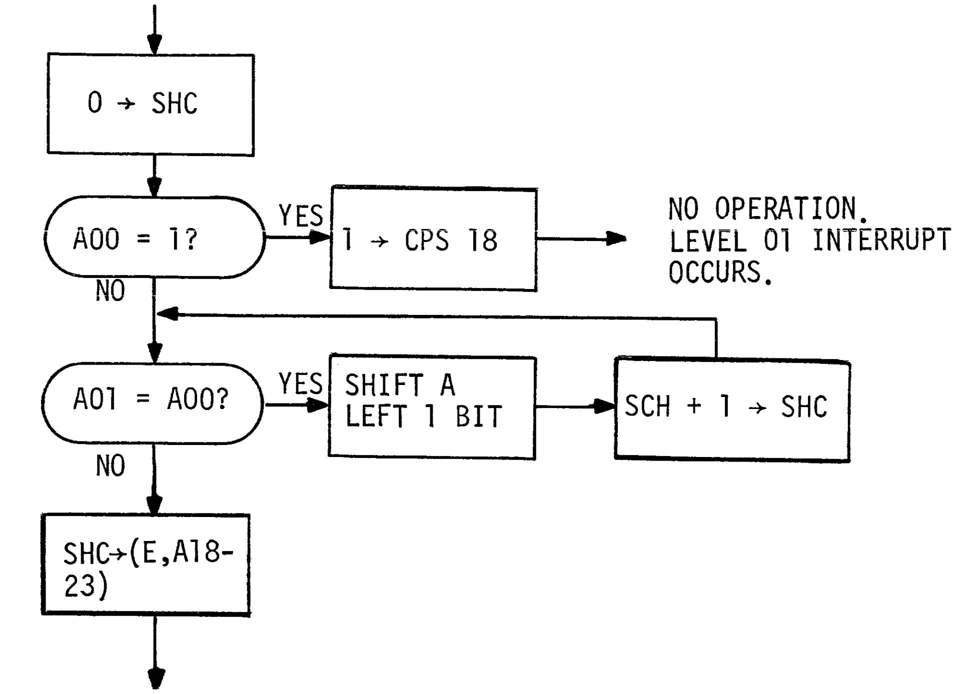


Figure 2-4-1. Normalize Short Flow Diagram

| Op Code | M | Y | Interpretation |
|---|---|---|---|
| LDE | 2 | 00000 | Zero the E register |
| NMS | 0 | 77776 (E) | Normalize and store shift count in E18-23 |
| LRS | | 00110 | Logical right shift 6 bits; i.e., right-justify most significant 17 bits of the fraction |
| STA | | FRAC | Save fraction |
| LDA | 2 | 00027 | $23_{10}$ → A18-23 |
| SUB | 0 | 77776 (E) | 23-Count = Exp → A18-23 |
| XOR | 2 | 00040 | Insert excess 32 |
| LLS | | 10001 | Position characteristic → A01-06 |
| IOR | | FRAC | Insert fraction |
| • | | | The floating point number is |
| • | | | now in the A register |

Using this example, if the A register contains 00432167, the results are:

| Instruction | M | Y | Results |
|---|---|---|---|
| LDE | 2 | 00000 | E = 00000000 |
| NMS | 0 | 77776 | A = 21507340; E = 00000005 |
| LRS | | 00110 | A = 00215073 |
| STA | | FRAC | FRAC = 00215073 |
| LDA | 2 | 00027 | A = 00000027 |
| SUB | 0 | 77776 | A = 00000022 = Exponent |
| XOR | 2 | 00040 | A = 00000062 = Characteristic |
| LLS | | 10001 | A = 31000000 |
| IOR | | FRAC | A = 31215073 = Floating point number |
| • | | | |
| • | | | |

The result of these instructions, in binary, is:

```
0|11  001  0|10  001  101  000  111  011
-----------------------------------------
S| CHARAC- |          FRACTION
I| TERIS-  |
G| TIC     |
N|         |
```

Name: Normalize Long

Mnemonic: NML

Op Code: 42

Timing: 3.20 μsec plus 0.32 μsec for each shift

Literal Class: 2

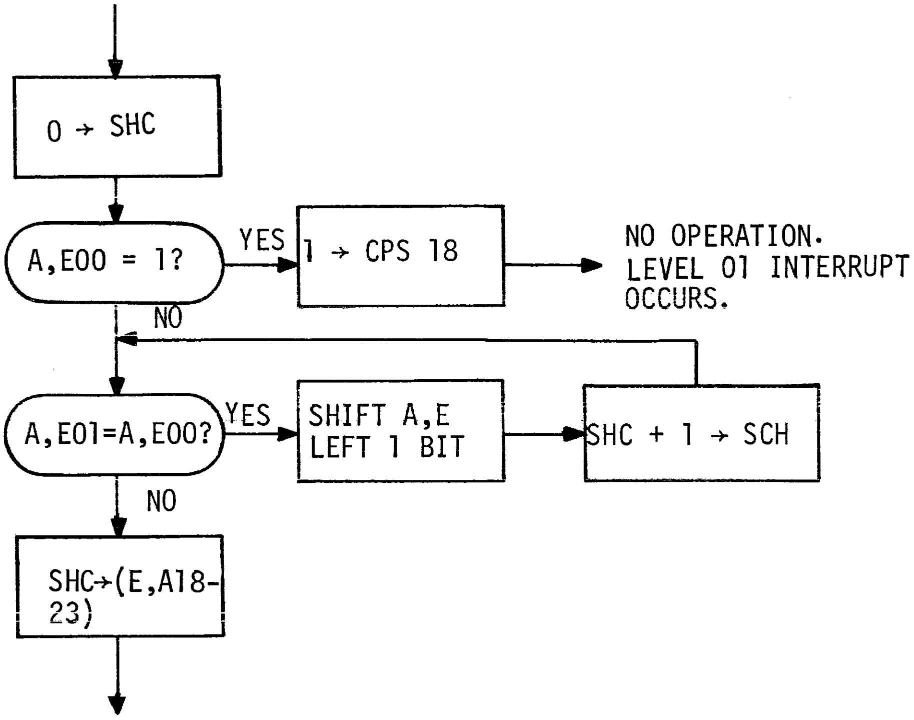
Execution Counter: CA

Registers Affected: A, E, (EA18-23), CPS18

Errors Possible: Accumulator overflow (OV)

Symbolic Notation: For each shift A,E00 → A,E00
A,En + 1 → A,En; where 0 ≤ n ≤ 47
0 → A,E47
A,E01 is lost
SHC + 1 → SHC

Description: The 48-bit double word in the A and E registers is normalized and the shift count replaces the contents of bits 18 through 23 of the location specified by the effective address. If the sign bit is set (A,E00 = 1, or A00 = 1) initially, the accumulator overflow indicator of the central processor status register is set (1 → CPS18) and no shifting occurs. In this case execution time is 1.92 μsec. If the double word to be normalized initially contains all zeroes, the maximum shifting occurs and a shift count of $57_8$ is stored. In this case execution time is 18.24 μsec. Normally, left-shifting occurs until A,E01 contains a 1. The functions performed by executing the NML instruction are shown in Figure 2-4-2.



PROCEED TO NEXT INSTRUCTION

Figure 2-4-2. Normalize Long Flow Diagram

Example:

| Before Execution | After Execution |
|---|---|
| A,E = 0032345676543210 | A,E = 3234567654321000 |
| (EA) = XXXXXXXX | (EA) = XXXXXX06 |

The following series of instructions normalizes the positive integer contained in the A and E registers and then creates a floating-point number in A and E from the result. These instructions do not take into consideration the possible loss of significant bits. Symbolic notations, including storage location addresses, are used.

| Op Code | M | Y | Interpretation |
|---|---|---|---|
| STA | | TEMP | Store most significant half of value to be normalized |
| LDA | 2 | 00000 | 0 → A |
| EAM | | TEMP | Restore most significant half of value to A; 0 → TEMP |
| NML | | TEMP | Normalize two-word value in A,E; store shift count into TEMP |
| LRL | | 01100 | Logical right shift $12_{10}$ bits; i.e., right-justify 35-bit fraction in A,E |
| STA | | MFRAC | Store most significant 11 bits of fraction |
| LDA | 2 | 00057 | $47_{10}$ → A |
| SUB | | TEMP | 47-Shift Count → A12-23 |
| XOR | 2 | 04000 | Insert excess 2048 |
| LLS | | 01011 | Position Characteristic → A01-12 |
| IOR | | MFRAC | Insert most significant 11 bits of fraction |
| • | | | |
| • | | | The double-precision floating-point value is now in the A and E registers |
| • | | | |

Using this example, if the A and E registers contain the integer 0000213456701234, the results are:

| Instruction | M | Y | Interpretation |
|---|---|---|---|
| STA | | TEMP | TEMP = 00002134 |
| LDA | 2 | 00000 | A = 00000000 |
| EAM | | TEMP | TEMP = 00000000; A = 00002134 |
| NML | | TEMP | A,E = 2134567012340000<br>TEMP = 00000014 ($12_{10}$) |
| LRL | | 01100 | A,E = 0000213456701234 |
| STA | | MFRAC | MFRAC = 00002134 |
| LDA | 2 | 00057 | A = 00000057 ($47_{10}$) |
| SUB | | TEMP | A = 00000043 ($35_{10}$) = Exponent |
| XOR | 2 | 04000 | A = 00004043 = Characteristic |
| LLS | | 01011 | A = 202140000 |
| IOR | | MFRAC | A,E = 2021613456701234 |

The result of these instructions, in memory, is:

```
 0|10 000 010 001 1|10 001 011 100 101 110 111 000 001 010 011 100
------------------------------------------------------------------
 S| CHARACTERISTIC |                FRACTION
 I|                |
 G|                |
 N|                |
```

## FIXED POINT ARITHMETIC INSTRUCTIONS

The fixed point arithmetic instructions are:

| Instruction Name | Mnemonic |
|---|---|
| Add Short | ADD |
| Add Long | ADL |
| Subtract Short | SUB |
| Subtract Long | SBL |
| Divide | DIV |
| Multiply | MPY |

Fixed point arithmetic instructions perform binary addition, subtraction, multiplication, and division, using integer operands. One operand may be either in the instruction itself (literal mode) or in memory (identified by the effective address of the instruction); the other operand is in the A register (short) or in the A and E registers (long). The results of fixed-point operations are in the A register or A and E registers.

Any fixed point arithmetic instruction, with the exception of MPY, can cause an overflow, which sets bit 18 of the CPS register. This is an arithmetic fault condition that causes a level 01 interrupt.

Name: Add Short
Mnemonic: ADD
Op Code: 10
Timing: 1.92 μsec
Literal Class: 1
Execution Counter: CB
Registers Affected: A, (EA), CPS18, CPI01
Errors Possible: Accumulator overflow (OV)
Symbolic Notation: A + (EA) → A
0 → FPL

Description: The contents (addend) of the location specified by the effective address are algebraically added to the contents (augend) of the A register. The resulting sum replaces the previous contents of the A register. The floating point long bit of the central processor indicator register is cleared (0 → CPI01) to denote that data are in single-word format. If the two operands have the same sign, the sum can be outside the range:

$$-2^{23} \leq \text{sum} \leq 2^{23}-1$$

When the sum is outside this range, the accumulator overflow bit of the central processor status register is set (1 → CPS18), causing a level 01 interrupt. When overflow occurs, the resultant sum in the A register is incorrect and is in one of the following forms:

If sum < $-2^{23}$, then A = $2^{24}$ + sum
If sum $\leq 2^{23}$, then A = $-2^{24}$ + sum

The relationship between operand signs and resultant sign after overflow is:

| Operand Signs |  | Resultant Sign After Overflow |
|---|---|---|
| A00 | (EA00) | A00 |
| 0 (+) | 0 (+) | 1 (-) |
| 1 (-) | 1 (-) | 0 (+) |

Name: Add Long

Mnemonic: ADL

Op Code: 11

Timing: 3.84 μsec

Literal Class: 1

Execution Counter: CB

Registers Affected: A, E, (EA), (EA + 1), CPS18, CPI01

Errors Possible: Accumulator overflow (OV)

Symbolic Notation: A,E + (EA),(EA + 1) → A,E
1 → FPL

Description: The double-precision fixed point value (addend) contained in two consecutive locations, the first of which is specified by the effective address, is algebraically added to the double word (augend) in the A and E registers; the A register contains the sign and the most significant half of the sum, and the E register contains the least significant half of the sum. The floating point long bit of the central processor indicator register is set (1 → CPI01) to denote that data are in double-word format. If the two operands have the same sign, a sum is possible outside the range:

$$-2^{47} \leq \text{sum} \leq 2^{47}-1$$

When the sum is outside this range, the accumulator overflow bit of the central processor status register is set (1 → CPS18), causing a level 01 interrupt. When overflow occurs, the resultant sum in the A and E registers is incorrect and is in one of the following forms:

If sum $< -2^{47}$, then A,E $= 2^{48}$ + sum<br>
If sum $\geq 2^{47}$, then A,E $= -2^{48}$ + sum

The relationship between operand signs and the resultant sign after overflow is:

| Operand Signs | | Resultant Sign After Overflow |
|---|---|---|
| A,E00 | (EA, (EA + 1) | A,E00 |
| 0 (+) | 0 (+) | 1 (-) |
| 1 (-) | 1 (-) | 0 (+) |

NOTE

ADL cannot perform the function A,E + A,E.

Name: Subtract Short

Mnemonic: SUB

Op Code: 12

Timing: 1.92 µsec

Literal Class: 1

Execution Counter: CB

Registers Affected: A, (EA), CPS18, CPI01

Errors Possible: Accumulator overflow (OV)

Symbolic Notation: A - (EA) → A
0 → FPL

Description: The contents (subtrahend) of the location specified by the effective address are algebraically subtracted from the contents (minuend) of the A register. The resulting difference replaces the previous contents of the A register. The floating point long bit of the central processor indicator register is cleared (0 → CPI01) to denote that data are in single-word format. If the two operands have opposite signs, a difference is possible outside the range:

$$-2^{23} \leq \text{difference} \leq 2^{23}-1$$

When the difference is outside this range, the accumulator overflow bit of the central processor status register is set (1 → CPS18), causing a level 01 interrupt. When overflow occurs, the resultant difference in the A register is incorrect and is in one of the following forms:

If difference $< -2^{23}$, then $A = 2^{24}$ + difference
If difference $\geq 2^{23}$, then $A = -2^{24}$ + difference

The relationship between operand signs and the resultant sign after overflow is:

| Operand Signs<br>A00 | <br>(EA00) | Resultant Sign<br>After Overflow<br>A00 |
|---|---|---|
| 0 (+) | 1 (-) | 1 (-) |
| 1 (-) | 0 (+) | 0 (+) |

Name: Subtract Long
Mnemonic: SBL
Op Code: 13
Timing: 3.84 μsec
Literal Class: 1

Execution Counter: CB

Registers Affected: A, E, (EA), (EA + 1), CPS18, CPI01

Errors Possible: Accumulator overflow

Symbolic Notation: A,E - (EA), (EA + 1) → A,E
1 → FPL

Description: The double-precision, fixed point value (subtrahend) contained in two consecutive locations, the first of which is specified by the effective address, is algebraically subtracted from the double-word value (minuend) in the A and E registers. The resulting difference replaces the previous contents of the A and E registers; the A register contains the sign and most significant 23 bits of the difference and the E register contains the least significant 24 bits of the difference. The floating point long bit of the central processor indicator register is set (1 → CPI01) to denote that data are in double-word format. If the two operands have opposite signs, a difference is possible outside the range:

$$-2^{47} \leq \text{difference} \leq 2^{47}-1$$

When the difference is outside this range, the accumulator overflow bit of the central processor status register is set (1 → CPS18), causing a level 01 interrupt. When overflow occurs, the resultant difference in the A and E registers is incorrect and in one of the following forms:

If difference $< -2^{47}$, then A,E = $2^{48}$ + difference
If difference $\geq 2^{47}$, then A,E = $-2^{48}$ + difference

The relationship between operand signs and the resultant sign after overflow is:

| Operand Signs | | Resultant Sign After Overflow |
|---|---|---|
| A,E00 | (EA, (EA + 1) | A,E00 |
| 0 (+) | 1 (-) | 1 (-) |
| 1 (-) | 0 (+) | 0 (+) |

Name: Divide

Mnemonic: DIV

Op Code: 15

Timing: 10.88 to 18.24 μsec for positive dividends; 12.16 to 19.52 μsec for negative dividends; 15.20 μsec overage (execution time is invarient with respect to divisor sign).

Literal Class: 1

Execution Counter: CB

Registers Affected: A, E, (EA), CPS18

Errors Possible: Accumulator overflow

Symbolic Notation: A,E ÷ (EA) → A
Remainder → E

Description: The double-precision value (dividend) in the A and E registers is algebraically divided by the value (divisor) in the location specified by the effective address. The resulting signed integer (quotient) replaces the previous contents of the A register and the remainder replaces the previous contents of the E register.

Execution of the DIV instruction begins by comparing the two high-order bits of the dividend (A00 and A01). If they are not the same, the accumulator overflow bit of the central processor status register is set (if A00 ≠ A01, then 1 → CPS18) and the instruction is terminated. In this case execution time is 1.92 or 3.20 μsec. (Setting of the overflow bit causes a level 01 interrupt.) If these bits are the same, the contents of the A and E registers are shifted left one bit position to convert the dividend to the form of a sign bit and 46 significant data bits; then the division is initiated. The divide operation is accomplished using sign/magnitude notation. Therefore, the signs of the divisor and dividend are compared and recorded for later examination to determine the correct signs of the quotient and remainder. The first cycle of the actual division begins by checking the sign of the divisor. If it is positive, the divisor is subtracted from that portion of the dividend that is in the A register. If the divisor is negative, it is added to the contents of the A register. This process results in a positive divisor/dividend relationship. If a carry is generated from the most significant bit add/subtract operation during this first cycle, the divisor is equal to or less than the dividend; therefore, the division is halted, the dividend is restored to its original position and sign, and the accumulator overflow bit is set. If no carry occurs, the dividend is shifted left one bit position and a 0 is placed in the first bit of the quotient. This entire process is then repeated 23 times; however, a carry now produces a 1 in the next bit of the quotient and cannot set the overflow bit. This division produces an algebraically correct result; i.e.,

$$\text{dividend} = \text{divisor} \times \text{quotient} + \text{remainder}$$

Thus, the range of the result is:

$$-2^{23} < \text{quotient and/or remainder} < 2^{23}$$

Attempts to divide by zero result in immediate overflow and subsequent level 01 interrupt.

Examples:

|  | Before Execution | Results |
|---|---|---|
| a. | A,E = 0000000000000031<br>(EA) = 00000006 | A = 00000004<br>E = 00000001<br>CPS18 = 0 |
| b. | A,E = 3767732100732601<br>(EA) = 07777776 | A = 37677321<br>E = 00732601<br>CPS18 = 1 |

Name: Multiply

Mnemonic: MPY

Op Code: 14

Timing: 10.56 μsec, except for the anomalous case of multiplication by -8388608 which takes 11.20 μsec.

Literal Class: 1

Execution Counter: CB

Registers Affected: A, E, (EA)

Errors Possible: None

Symbolic Notation: A × (EA) → A,E

Description: The value (multiplicand) in the A register is algebraically multiplied by the value (multiplier) in the location specified by the effective address. The resulting signed integer product replaces the previous contents of the A and E registers in double-precision, fixed point format. Any previous contents at the E register are lost. Overflow is not possible.

Example:

| Before Execution | After Execution |
|---|---|
| A = 00000005 | A,E = 7777777777777716 |
| (EA) = 77777766 | |

FLOATING POINT ARITHMETIC INSTRUCTIONS

The formats and descriptions of floating point number representation are given in Section 3. The floating point instructions are:

| Instruction Name | Mnemonic |
|---|---|
| Floating Add Short | FAS |
| Floating Add Long | FAL |
| Floating Subtract Short | FSS |
| Floating Subtract Long | FSL |
| Floating Multiply Short | FMS |

| Instruction Name | Mnemonic |
|---|---|
| Floating Multiply Long | FML |
| Floating Divide Short | FDS |
| Floating Divide Long | FDL |
| Store Normalized and Rounded | SNR |

The following rules and conventions apply to floating point operations:

a. The operands used in floating point operations are considered to be in normalized floating point format.

b. If a floating point operation uses operands that are not normalized floating point numbers, the results are unpredictable. Such usage should be avoided.

c. If FPL is reset (CPI01 = 0), all floating point arithmetic instructions convert the single-precision floating point number held in the A register to double-precision format and set FPL prior to instruction execution.

   All floating-point arithmetic operations are performed in the double-precision format. Floating point short instructions (FAS, FSS, FMS, and FDS) convert the floating point operand fetched from memory to double-precision (long) format prior to instruction execution.

d. The results of floating point arithmetic operations are in normalized double-precision floating point format in the A and E registers.

e. Use of abnormal zero may cause loss of accuracy on floating point add and subtract operations.

f. Use of abnormal zero may erroneously cause floating point overflow on floating point multiply or divide operations.

g. Any floating point instruction can cause a characteristic overflow condition. This occurs whenever the characteristic becomes too large to be held in the characteristic field (6 bits for single-precision and 12 bits for double-precision). Characteristic overflow sets the floating-point overflow alarm (FPO, CPS19). This also terminates the instruction and makes the result zero in the A and E registers. With FPO set, a level 01 interrupt occurs. Further floating point instructions will be treated as no-ops until FPO is reset. The Real Time Executive System is designed to handle this interrupt condition and to reset FPO.

h. Any floating point instruction can cause a characteristic underflow. This occurs if the operation causes the characteristic to be reduced below zero. Characteristic underflow sets the floating point underflow alarm (FPU, CPI00) and has the following effects:

(1) If FPU is set either before or during execution of a floating point multiply or divide instruction, the instruction becomes a no-operation and the A and E registers are cleared (result is floating point zero). FPU remains set and will be reset only by subsequent successful execution of a floating point add or subtract or of a Load A (LDA) or Load Long (LDL) instruction.

(2) If FPU is set either before or during execution of floating point add or subtract, AE and FPU are cleared and the instruction becomes an add or subtract from zero instruction, using the effective address specified by the floating point instruction. However, if the effective address contains a fraction (negative characteristic), FPU will be set again during the final normalization cycle.

4.113

Execution time of floating point instructions varies according to many factors. Details on some of these factors are as follows:

a. Execution time for FAL and FAS, for large exponent differences, is as follows. dX = $X_{AE} - X_{BT}$ is the signed exponent difference between the augend (or minuend) and the addend (or subtrahend).

| Mnemonic | AE<br>Sign | BT<br>Sign | dX | Timing<br>(μsec) |
|---|---|---|---|---|
| FAL | Pos. | - | > +35 | 9.92 |
| FSL | Pos. | - | > +35 | 9.92 |
| FAL | Neg. | - | > +35 | 11.20 |
| FSL | Neg. | - | > +35 | 11.84 |
| FAL | - | Pos. | < -35 | 8.96 |
| FAL | - | Neg. | < -35 | 10.24 |
| FSL | - | Pos. | < -35 | 10.24 |
| FSL | - | Neg. | < -35 | 8.96 |

For small exponent differences (-36 < dX < 36), execution times are as follows for operand-pairs invoking minimum post-normalization times.

| Mnemonic | AE<br>Sign | BT<br>Sign | dX | Timing<br>(μsec) |
|---|---|---|---|---|
| FAL | Pos. | Pos. | > 0 | 8.96 + 0.32 × dX |
| FSL | Pos. | Neg. | > 0 | 8.96 + 0.32 × dX |
| FAL | Pos. | Neg. | > 0 | 10.56 + 0.32 × dX |
| FSL | Pos. | Pos. | > 0 | 10.56 + 0.32 × dX |
| FAL | Pos. | Pos. | < 1 | 9.92 - 0.32 × dX |
| FSL | Pos. | Neg. | < 1 | 9.92 - 0.32 × dX |
| FAL | Pos. | Neg. | < 1 | 10.24 - 0.32 × dX |
| FSL | Pos. | Pos. | < 1 | 10.24 - 0.32 × dX |
| FAL | Neg. | Neg. | > 0 | 10.88 + 0.32 × dX |
| FSL | Neg. | Pos. | > 0 | 10.88 + 0.32 × dX |
| FAL | Neg. | Pos. | > 0 | 9.28 + 0.32 × dx |
| FSL | Neg. | Neg. | > 0 | 9.28 + 0.32 × dX |
| FAL | Neg. | Neg. | < 1 | 11.84 - 0.32 × dX |
| FSL | Neg. | Pos. | < 1 | 11.84 - 0.32 × dX |
| FAL | Neg. | Pos. | < 1 | 11.52 - 0.32 × dX |
| FSL | Neg. | Neg. | < 1 | 11.52 - 0.32 × dX |

Execution time for FAL and FSL may be increased by as much as 10.88 μsec due to post normalization. This only occurs on FAL if the operand signs differ and on FSL if the operand signs agree.

b. Execution time for (nontrivial) FML is 29.12 μsec; plus 0.32 μsec times the number of nonzero bits in the positive representation of the multiplicand (A, E), positions A14 through E11 inclusive; plus 2.88 μsec if the multiplier is negative, or 1.28 μsec if the multiplier is positive but the multiplicand is negative. The range is, then:

| Multiplier | Multiplicand | Minimum | Maximum |
|---|---|---|---|
| Pos. | Pos. | 29.12 μsec | 36.16 μsec |
| Pos. | Neg. | 32.00 μsec | 39.04 μsec |
| Neg. | Pos. | 30.40 μsec | 37.44 μsec |
| Neg. | Neg. | 32.00 μsec | 39.04 μsec |

c. Execution time for (nontrivial) FLD is 36.80 μsec; plus 0.32 μsec times the number of nonzero bits in the positive representation of the quotient (A, E), positions A14 through E23 inclusive; plus 2.88 μsec if the divisor is negative, or 1.28 μsec if the divisor is positive but the dividend is negative. The range is, then:

| Dividend | Divisor | Minumum | Maximum |
|---|---|---|---|
| Pos. | Pos. | 36.80 μsec | 47.68 μsec |
| Pos. | Neg. | 39.68 μsec | 50.56 μsec |
| Neg. | Pos. | 38.08 μsec | 48.96 μsec |
| Neg. | Neg. | 39.68 μsec | 50.56 μsec |

d. If, after execution of the SNR, FPO and FPU are both reset, then execution time for absolute addressing is:

Positive operand: 7.36 μsec
Negative operand: 8.64 μsec

If FPO and FPU are set prior to execution, or become set during execution of the SNR, then execution time for absolute addressing is:

Positive operand: 5.12 μsec
Negative operand: 6.40 μsec

e. Average execution times for floating point operations are calculated for two data bases: Process and Random. The relevant characteristics of these data bases are as follows:

(1) In both cases it is assumed that the frequency of occurrence of FPO and FPU is statistically insignificant.

(2) In the Process data base it is assumed that 80% of all operands are positive, and in the Random data base that 50% of all operands are positive.

(3) In the Process data base it is assumed that 80% of all operands are short; the probability that FPLN = 1 is 0.2. In the Random data base, nonmixed arithmetic is assumed: FPLN = 1 for FAL, FPLN = 0 for FAS, etc.

(4) In both cases, it is assumed that the probability distribution of exponent differences is an even function:

$$P(dX = n) = P(dX = -n)$$

(5) It is assumed that the probability distribution of exponent differences is a monotonically decreasing function of the difference, as follows:

$$k_p = [P(|dX| = n + 1)]/[P(|dX| = n)]$$

The average preshift count, $dX_{AVG}$, may be derived from $k_p$. Assumed values for $k_p$ are 0.8 (Process) and 0.9 (Random), leading to $dX_{AVG}$ values of 3.2 (Process) and 8.1 (Random).

It is noted that a completely random data base ($k_p = 1.0$) produces unrealistically fast execution times for add and subtract.

(6) The assumed probability of a (fraction) bit being a one is 0.5, except for the restriction that all operands are normalized.

f. The average base time for FAL, accounting for distribution of operand signs with Process data, is:

$$(0.8)(0.8)(0.5)(8.96 + 9.92) + (0.2)(0.2)(0.5)(10.88 + 11.84)$$
$$+ (0.8)(0.2)(0.5)(10.56 + 10.24 + 9.28 + 11.52) = 9.904 \ \mu\text{sec}$$

Analogously, FAL (Random) = 10.40 μsec
FSL (Process) = 10.404 μsec
FSL (Random) = 10.40 μsec

The average time is the base time, corrected for short operands (Process) and the average preshift. For Process data, this is:

| | FAL: | FSL: |
|---|---|---|
| Base time: | 9.904 μsec | 10.404 μsec |
| FPLN: (0.8)1.92 | 1.54 μsec | 1.54 μsec |
| $\|dX\|$: (0.32)3.2 | 1.024 μsec | 1.024 μsec |
| Average time: | 12.468 μsec | 12.968 μsec |

For Random data:

| | FAL: | FSL: |
|---|---|---|
| Base time: | 10.40 μsec | 10.40 μsec |
| $\|dX\|$: (0.32)8.1 | 2.59 μsec | 2.59 μsec |
| Average time: | 12.99 μsec | 12.99 μsec |

g. The average FML time is the base time, corrected for nonzero multiplicand bits, negative operands, and (for Process) short operands. For Process data, this is:

| | |
|---|---|
| Base time: | 29.12 μsec |
| Bits: 0.32[0.5(16) + 0.5(0.2)(6)] | 2.75 μsec |
| Neg: 2.88(0.2) + 1.28(0.2)(0.8) | 0.78 μsec |
| Short: 1.92(0.8) | 1.54 μsec |
| Average | 34.19 μsec |

For Random data:

| | |
|---|---|
| Base time: | 29.12 μsec |
| Bits: 0.32(0.5(22) | 3.52 μsec |
| Neg: 2.88(0.5) + 1.28(0.25) | 1.76 μsec |
| Average | 34.40 μsec |

h. The average FDL time is the base time, corrected for nonzero quotient bits, negative operands, and (for Process) short operands.

| | | Process | Random |
|---|---|---|---|
| Base time: | | 36.80 μsec | 36.80 μsec |
| Bits: | 0.32(0.5)33 | 5.28 μsec | 5.28 μsec |
| Neg. | | 0.78 μsec | 1.76 μsec |
| Short: | | 1.54 μsec | 0 μsec |
| | Average | 44.40 μsec | 43.84 μsec |

i. Execution times for short floating arithmetic are generally 1.92 μsec longer than their long counterparts: This is reflected in the figures for Process data. For Random data, due to assumption e(3), the difference is 3.84 μsec.

j. Average execution times for SNR follow from Assumption e(2):

Process: (0.8)7.36 + (0.2)8.64 = 7.616 μsec
Random: (0.5)7.36 + (0.5)8.64 = 8.00 μsec

Name: Floating Point Add Short
Mnemonic: FAS
Op Code: 30
Timing: 14.39 μsec for Process data, 16.83 μsec for Random data.
Literal Class: 1
Execution Counter: CF
Registers Affected: A, E, CPS19, CPI00, CPI01, (EAdp)
Errors Possible: Floating point overflow (FPO),
floating point underflow (FPU)
Symbolic Notation: A,E + (EAdp) → A,E
1 → FPL

Description: The floating point number (addend) in the location specified by the effective address is algebraically added to the contents (augend) of the A register. The addend and, if the floating point long (FPL) bit of the central processor indicator register is clear (if CPI01 = 0), the augend too, is converted to double-precision floating point format before the addition is performed. The resulting normalized floating point sum replaces the previous contents of the A and E registers. The FPL bit is set to indicate the result in the double-word, floating point format. Floating point overflow (1 → CPS19) or floating point underflow (1 → CPI00) is possible.

Name: Floating Point Add Long
Mnemonic: FAL
Op Code: 34
Timing: 12.47 μsec for Process data, 12.99 μsec for Random data
Literal Class: Special
Execution Counter: CF
Registers Affected: A, E, CPS19, CPI00, CPI01, (EA), (EA + 1)
Errors Possible: Floating point overflow (FPO),
floating point underflow (FPU)
Symbolic Notation: A,E + (EA), (EA + 1) → A,E
1 → FPL

Description: The double-precision, floating point value (addend) contained in two consecutive locations, the first of which is specified by the effective address, is algebraically added to the double word (augend) in the A and E registers. The resulting normalized double-precision, floating point sum replaces the previous contents of the A and E registers. If the floating point long (FPL) bit of the central processor indicator register is clear (if CPI01 = 0) when the instruction is issued, the single-precision, floating point augend in the A register is converted to long format before the addition is performed. The FPL bit is set to indicate that the result is in double-word, floating point format. Floating point overflow (1 → CPS19) or floating point underflow (1 → CPI00) is possible.

Name: Floating Point Subtract Short
Mnemonic: FSS
Op Code: 31
Timing: 14.89 μsec for Process data, 16.83 μsec for Random data
Literal Class: 1
Execution Counter: CF
Registers Affected: A, E, CPS19, CPI00, CPI01, (EAdp)
Errors Possible: Floating point overflow (FPO),
floating point underflow (FPU)
Symbolic Notation: A,E - (EAdp) → A,E
1 → FPL

Description: The floating point number (subtrahend) in the location specified by the effective address is algebraically subtracted from the number (minuend) in the A register. Before the subtraction is performed, the subtrahend, and also the minuend if the floating point long (FPL) bit of the central processor indicator register is clear (if CPI01 = 0), is converted to double-precision, floating point format. The resulting normalized floating point difference replaces the previous contents of the A and E registers. The FPL bit is set to indicate that the result is in double-word, floating point format. Floating point overflow (1 → CPS19) or floating point undeflow (1 → CPI00) is possible.

Name: Floating Point Subtract Long
Mnemonic: FSL
Op Code: 35
Timing: 12.97 μsec for Process data, 12.99 μsec for Random data
Literal Class: Special
Execution Counter: CF

Registers Affected: A, E, CPS19, CPI00, CPI01

Errors Possible: Floating point overflow (FPO),
floating point underflow (FPU)

Symbolic Notation: A, E - (EA), (EA + 1) → A,E
1 → FPL

Description: The double-precision, floating point value (subtrahend) contained in two consecutive locations, the first of which is specified by the effective address, is algebraically subtracted from the double-precision, floating point value (minuend) in the A and E registers. If the floating point long (FPL) bit of the central processor indicator register is clear (if CPI01 = 0) when the instruction is issued, the single-precision, floating point minuend in the A register is converted to double-precision floating point format before the subtraction is performed. The resulting normalized double-precision, floating point difference replaces the previous contents of the A and E registers. The FPL bit is set to indicate that the result is in double-word, floating point format. Floating point overflow (1 → CPS19) or floating point underflow (1 → CPI00) is possible.

Name: Floating Point Multiply Short

Mnemonic: FMS

Op Code: 32

Timing: 36.11 μsec for Process data, 38.24 μsec for Random data

Literal Class: 1

Execution Counter: CF

Registers Affected: A, E, CPS19, CPI00, CPI01, (EAdp)

Errors Possible: Floating point overflow (FPO),
floating point underflow (FPU)

Symbolic Notation: A,E × (EAdp) → A,E
1 → FPL

Description: The floating point number (multiplier) in the location specified by the effective address and the value (multiplicand) in the A register are algebraically multiplied. Before the multiplication is performed, the multiplier, and also the multiplicand if the floating point long (FPL) bit of the central processor indicator register is clear (if CPI01 = 0), are converted to double-precision, floating point format. The resulting double-precision, floating point product replaces the previous contents of the A and E registers. The FPL bit is set to indicate that the result is in double-word, floating point format. Floating point overflow (1 → CPS19) or floating point underflow (1 → CPI00) is possible.

Name: Floating Point Multiply Long<br>
Mnemonic: FML<br>
Op Code: 32<br>
Timing: 34.19 μsec for Process data, 34.40 μsec for Random data<br>
Literal Class: Special<br>
Execution Counter: CF<br>
Registers Affected: A, E, CPS19, CPI00, CPI01, (EA), (EA + 1)<br>
Errors Possible: Floating point overflow (FPO),
floating point underflow (FPU)<br>
Symbolic Notation: A,E × (EA), (EA + 1) → A,E
1 → FPL

Description: The double-precision, floating point value (multiplicand) in the A and E registers is algebraically multiplied by the double-precision, floating point number (multiplier) in two consecutive locations, the first of which is specified by the effective address. If the floating point long (FPL) bit of the central processor indicator register is clear (if CPI01 = 0) at the beginning of the instruction, the single-precision floating point multiplicand in the A register is converted to long format before the multiplication operation is performed. The resulting normalized double-precision, floating point product replaces the previous contents of the A and E registers. The FPL bit is set to indicate that the result is in double-word format. Floating point overflow (1 → CPS19) or floating point underflow (1 → CPI00) is possible.

Name: Floating Point Divide Short
Mnemonic: FDS
Op Code: 33
Timing: 46.32 μsec for Process data, 47.68 μsec for Random data
Literal Class: 1
Execution Counter: CF
Registers Affected: A, E, CPS19, CPI00, CPI01, (EAdp)
Errors Possible: Floating point overflow (FPO),
floating point underflow (FPU)
Symbolic Notation: A,E ÷ (EAdp) → A,E
1 → FPL

Description: The floating point number (dividend) in the A and E registers is algebraically divided by the value (divisor) in the location specified by the effective address. Before the division is performed, the divisor, and also the dividend if the floating point long (FPL) bit of the central processor indicator register is clear (if CPI01 = 0), are converted to double-precision floating point format. The resulting quotient replaces the previous contents of the A and E registers. The FPL bit is set to indicate that the result is in double-word, floating point format. Division by zero causes floating point overflow (1 → CPS19) unless the dividend is also zero, in which case, floating point underflow occurs (1 → CPI00) instead of overflow.

Name: Floating Point Divide Long
Mnemonic: FDL
Op Code: 37
Timing: 44.40 μsec for Process data, 43.84 μsec for Random data
Literal Class: Special

Execution Counter: CF

Registers Affected: A, E, CPS19, CPI00, CPI01, (EA), (EA+1)

Errors Possible: Floating point overflow (FPO),
floating point underflow (FPU)

Symbolic Notation: A,E ÷ (EA), (EA + 1) → A,E
1 → FPL

Description: The double-precision, floating point number (dividend) in the A and E registers is algebraically divided by the double-precision, floating point value (divisor) in two consecutive locations, the first of which is specified by the effective address. Before the division is performed, the single-precision, floating point dividend in the A register is converted to long format if the floating point long (FPL) bit of the central processor indicator register is clear (if CPI01 = 0). The resulting normalized double-precision, floating point quotient replaces the previous contents of the A and E registers. The FPL bit is set to indicate that the result is in double-word, floating point format. Division by zero causes floating point overflow (1 → CPS19) unless the dividend is also zero, in which case floating point underflow (1 → CPI00) occurs instead of overflow.

Name: Store Normalized and Rounded

Mnemonic: SNR

Op Code: 47

Timing: 7.62 μsec for Process data, 8.00 μsec for Random data

Literal Class: 1

Execution Counter: CF

Registers Affected: A, E, CPS19, CPI00, CPI01, (EA)

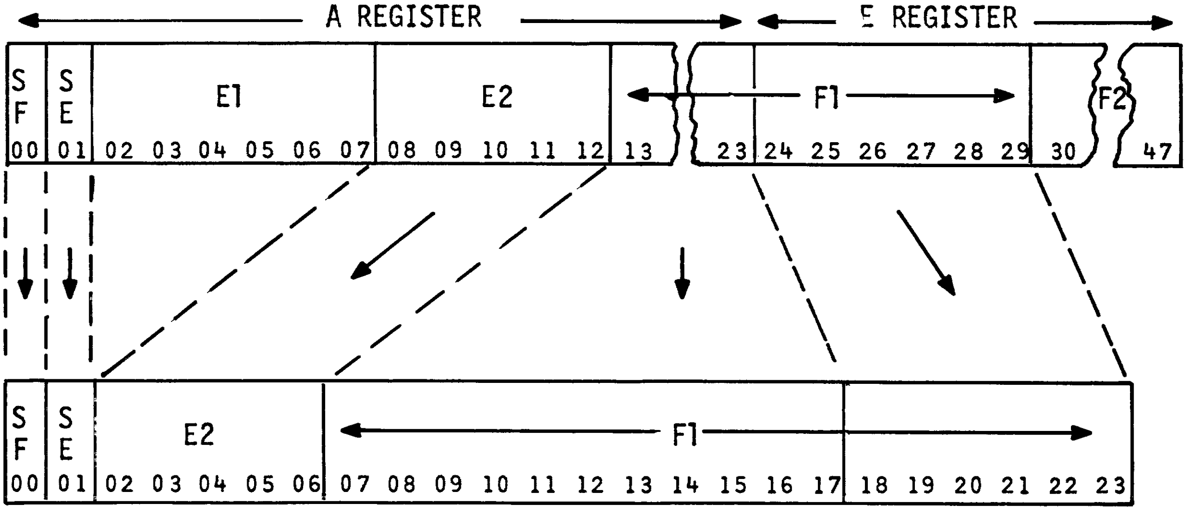
Errors Possible: Floating point overflow (FPO),
floating point underflow (FPU)

Symbolic Notation: None

Description: The double-precision, floating point number in the A and E registers is rounded and converted to single-precision, floating point format. This result replaces the previous contents of the A register and is stored in the location specified by the effective address. The E register is cleared and the floating point long (FPL) bit of the central processor indicator register is cleared (0 → CPI01) to denote that the result is in single-precision, floating point format. Both floating point overflow (1 → CPS19) and floating point underflow (1 → CPI00) are possible.

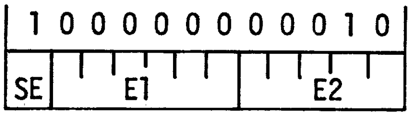
The following diagram shows how the original double-precision number is converted to the resulting single-precision number. Shaded areas (bits 02 through 07 and 30 through 47) indicate discarded portions of the original number.

DOUBLE-PRECISION FORMAT



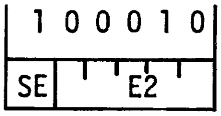
RESULT IN A REGISTER

The characteristic (exponent) of the resultant single-precision floating point number is made up of the high-order bit (SE) and the five low-order bits (E2) of the double-precision characteristic. The high-order bit retains the proper excess in the new characteristic. For example, if the characteristic of the double-precision number is:



or 2 excess 2048

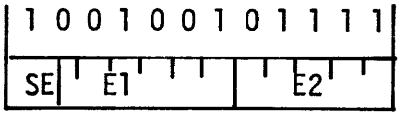
Then the characteristic of the single-precision number will be:



or 2 excess 32.

Note that the bits represented in the diagram by E1 must all be different from SE in order to maintain accuracy in the characteristic of the single-precision number. If this is not the case, overflow or underflow occurs.

A characteristic of:



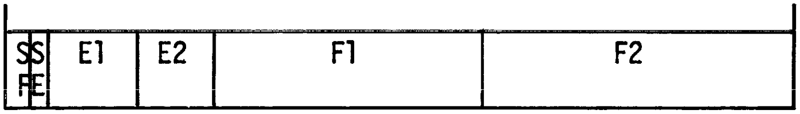
is unacceptable because all significant bits cannot be retained.

The fraction (mantissa) portion of the new number is made up of bits 13 through 29 of the double-precision word. These bits are rounded by adding a 1 bit to the original bit 30. Bit 30 of the original number represents the highest order bit of the discarded portion of the fraction. If bit 30 equals 1, the retained fraction is rounded by a carry. If bit 30 equals 0, no rounding takes place. Rounding can affect the characteristic only if bits 13 through 29 of the original value are all 1 bits and the number is positive, or if bits 13 through 29 are all zero bits and the number is negative. When this occurs, the characteristic is adjusted.

Examples:

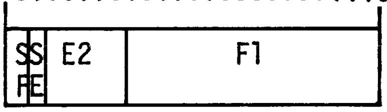
a. If the A and E registers contain:

01000000.0011101011010000000110110000000000000000000



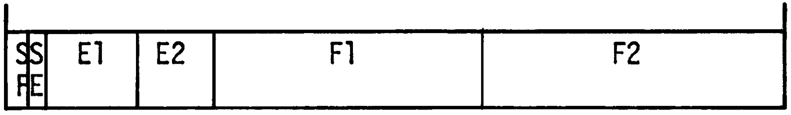
the result of SNR would be:

0100111010110100000001110



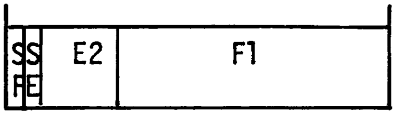
b. If the A and E registers contain:

001111111100111000110000101101000000000000000000



the result of SNR would be:

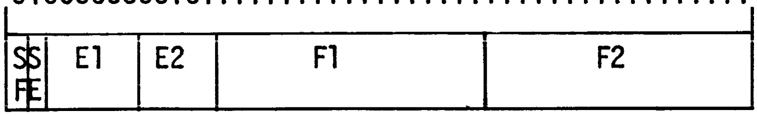
0011001110001100000101101



Note that, in this example, no change resulted from rounding.

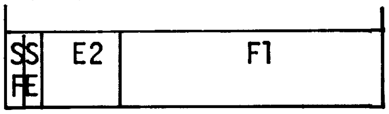
c. If the A and E registers contain:

010000000010111111111111111111111111111111111111



the result from SNR instruction would be:

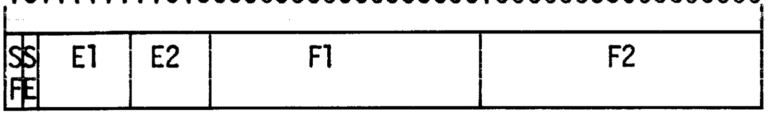
010011010000000000000000



Note the effect of rounding on both the characteristic and the fraction. The floating point value changed from 31.999... to 32.000.

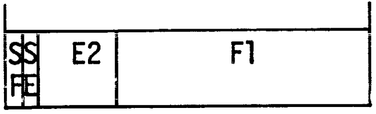
d. If the A and E registers contain the negative value:

101111111101000000000000000000100000000000000000



the result from executing SNR would be:

101100110000000000000000



This example shows the rounding of a negative value. The effect of SNR can be seen by converting the original number to its 2's complement form, executing the SNR and then converting the result to the negative form.

## INDEX CONTROL INSTRUCTIONS

Index control instructions operate on information in memory and in the two index registers: index A and index B. This information is 15 bits in size and corresponds to bits 09 through 23 of full-word data. The index control instructions are:

| Instruction Name | Mnemonic |
|---|---|
| Load Index A | LXA |
| Load Index B | LXB |
| Store Index A | SXA |
| Store Index B | SXB |
| Add to Index A | AXA |
| Add to Index B | AXB |
| Compare Index A and Memory | CXA |
| Compare Index B and Memory | CXB |
| Test and Increment Index A | TIA |
| Test and Increment Index B | TIB |
| Branch and Decrement Index A | BDA |
| Branch and Decrement Index B | BDB |

Name: Load Index A

Mnemonic: LXA

Op Code: 60

Timing: 1.92 μsec

Literal Class: 1

Execution Counter: CH

Registers Affected: XA, (EA)

Errors Possible: None

Symbolic Notation: (EA09-23) → XA

Description: The contents of bits 09 through 23 of the location specified by the effective address replace the contents of the index A register. The contents of the location loaded into the XA register remain unchanged.

Name: Load Index B
Mnemonic: LXB
Op Code: 61
Timing: 1.92 μsec
Literal Class: 1
Execution Counter: CH
Registers Affected: XB, (EA)
Errors Possible: None
Symbolic Notation: (EA09-23) → XB
Description: The contents of bits 09 through 23 of the location specified by the effective address replace the contents of the index B register. The contents of the location loaded into the XB register remain unchanged.

Name: Store Index A
Mnemonic: SXA
Op Code: 62
Timing: 2.24 μsec
Literal Class: 2
Execution Counter: CH
Registers Affected: XA, (EA)
Errors Possible: None
Symbolic Notation: XA → (EA09-23)
0 → (EA00-08)
Description: The contents of the index A register replace the contents of bits 09 through 23 of the location specified by the effective address, and zeroes replace the contents of bits 00 through 08 of this location. The contents of the index A register are unchanged.

Name: Store Index B

Mnemonic: SXB

Op Code: 63

Timing: 2.24 μsec

Literal Class: 2

Execution Counter: CH

Registers Affected: XB, (EA)

Errors Possible: None

Symbolic Notation: XB → (EA09-23)
0 → (EA00-08)

Description: The contents of the index B register replace the contents of bits 09 through 23 of the location specified by the effective address, and zeros replace the contents of bits 00 through 08 of this location. The contents of the index B register are unchanged.

Name: Add to Index A

Mnemonic: AXA

Op Code: 64

Timing: 2.24 μsec

Literal Class: 1

Execution Counter: CH

Registers Affected: XA, (EA)

Errors Possible: None

Symbolic Notation: (EA09-23) + XA → XA

Description: The contents (addend) of bits 09 through 23 of the location specified by the effective address are algebraically added to the contents (augend) of the index A register. The resulting sum replaces the previous contents of the index A register. The contents of the location containing the addend are unchanged.

Name: Add to Index B

Mnemonic: AXB

Op Code: 65

Timing: 2.24 μsec

Literal Class: 1

Execution Counter: CH

Registers Affected: XB, (EA)

Errors Possible: None

Symbolic Notation: (EA09-23) + XB → XB

Description: The contents (addend) of bits 09 through 23 of the location specified by the effective address are algebraically added to the contents (augend) of the index B register. The resulting sum replaces the previous contents of the index B register. The contents of the location containing the addend are unchanged.

Name: Compare Index A With Memory

Mnemonic: CXA

Op Code: 66

Timing: 2.56 μsec

Literal Class: 1

Execution Counter: CH

Registers Affected: PC, XA, (EA)

Errors Possible: None

Symbolic Notation: If XA > (EA09-23), then PC + 1 → PC

Description: The contents of the index A register are compared with the contents of bits 09 through 23 at the location specified by the effective address. If the contents of the index A register are greater, the contents of the program counter are incremented by 1 to skip the next instruction. If the contents of the index A register are not greater, no further action takes place and the program advances to the next instruction in sequence. The CXA instruction must not be immediately succeeded by a two-word instruction.

Name: Compare Index B With Memory

Mnemonic: CXB

Op Code: 67

Timing: 2.56 μsec

Literal Class: 1

Execution Counter: CH

Registers Affected: PC, XB, (EA)

Errors Possible: None

Symbolic Notation: If XA > (EA09-23), then PC + 1 → PC

Description: The contents of the index B register are compared with the contents of bits 09 through 23 of the location specified by the effective address. If the contents of the index B register are greater, the contents of the program counter are incremented by 1 to skip the next sequential instruction. If the contents of the index B register are not greater, no further action takes place and the program advances to the next instruction in sequence. The CXB instruction must not be immediately succeeded by a two-word instruction.

NOTE

CXA and CXB compare 15-bit unsigned values in the range $0 \leq X \leq 77777$. These instructions facilitate address comparisons but should not be used for algebraic comparisons where one value is negative (> 37777) and one value is positive (≤ 37777).

Name: Test and Increment Index A

Mnemonic: TIA

Op Code: 70

Timing: 2.56 μsec

Literal Class: 1

Execution Counter: CH

Registers Affected: PC, XA, (EA09-23)

Errors Possible: None

Symbolic Notation: If $XA < (EA09\text{-}23)$, then $XA + 1 \rightarrow XA$
IF $XA \geq (EA09\text{-}23)$, then $PC + 1 \rightarrow PC$ and $0 \rightarrow XA$

Description: The contents of the index A register are compared with the contents of bits 09 through 23 of the location specified by the effective address. If the contents of the index A register are less than the memory operand, the contents of the index A register are incremented by 1 and the program sequence proceeds to the next sequential instruction. If the contents at the A register are not less than the memory operand, the index A register is cleared and the program counter register is incremented by 1 to skip the next sequential instruction. This instruction must not be immediately succeeded by a two-word instruction in any program.

NOTES

- TIA and TIB accomplish compares on 15-bit unsigned values in the range $0 \leq X \leq 77777$. This facilitates address comparisons, but these instructions should not be used for algebraic compares.
- Also, TIA and TIB increment XA and XB, but no overflow can result from this addition.
- TIA and TIB are useful for nested loops and for accessing a buffer area where direction of access is important.

Example:

Add the values in a table where the addresses of the first and last words in the table are known.

```
Location  Op    M    I  XB  XA  Y or DISP  Interpretation
                       (Bits)
   .
   .
   .
  03577   LXA   4    0   0   0     0011    Address of first
                 (PC+0011)                 word → XA
  03600   LDS   2                 00000    Preclear A for adding
  03601   ADD   3    0   0   1     0000    Add a value
  03602   TIA   4    0   0   0     0007    If XA < LWA, XA + 1 → XA;
                 (PC + 007)                If XA ≥ LWA, skip
  03603   BRU   4    0   0   0     7776    XA < LWA; therefore, add
                                           another value.

  Constants Or Variable Data

  03610   X     X   (FWA)                  First Word Address
  03611   X     X   (LWA)                  Last Word Address
```

Name: Test and Increment Index B

Mnemonic: TIB

Op Code: 71

Timing: 2.56 μsec

Literal Class: 1

Execution Counter: CH

Registers Affected: PC, XB, (EA09-23)

Errors Possible: None

Symbolic Notation: If XB < (EA09-23), then XB + 1 → XB
If XB ≥ (EA09-23), then 0 → XB and PC + 1 → PC

Description: The contents of the index B register are compared with the contents of bits 09 through 23 of the location specified by the effective address. If the contents of the index B register are less than the memory operand, the contents of the index B register are incremented by 1 and the program sequence proceeds to the next sequential instruction. If the contents of the index B register are not less than the memory operand, the index B register is cleared and the program counter register is incremented by 1 to skip the next instruction in sequence. This instruction must not be immediately succeeded by a two-word instruction in any program. The notes in the description of the TIA instruction also apply to TIB.

Example:

Send the thirty 8-bit characters contained in ten consecutive memory locations to an output subroutine, one character at a time. Each character should be in bit positions 16 through 23 of the A register for output. The instruction BSP OUTPUT calls the subroutine. Assume the ten words are stored in locations 04760 through 04711.

```
Location  Op    M    I  XB  XA  Y or DISP  Interpretation
                      (Bits)
   .
   .
   .
 04703    LXA   2                 00000    0 → XA
 04704    LXB   2                 00000    0 → XB
 04705    LDE   4    0   1   0     0053    PC + 0053 + XB → E
 04706    LLL   2                 01710    Logical Left Shift long
                                           8 bits; char → A16-23
 04707    BSP        OUTPUT                Branch to output subroutine
 04710    TIA   2                 00002    If XA ≠ 2; XA + 1 → XA
                                           If XA = 2; 0 → XA; skip
 04711    BRU   4    0   0   0     7775    Branch to 04706
 04712    TIB   2                 00011    If XB ≠ 9; XB + 1 → XB
                                           If SB = 9; skip
 04713    BRU   4    0   0   0     7772    Branch to 04705
 04714     .                               Finished
           .
           .
```

This example shows nested loops, using both TIA and TIB. Each time the inner loop is completed, the XA count goes to zero, so the inner loop reinitializes itself.

Name: Branch and Decrement Index A

Mnemonic: BDA

Op Code: 72

Timing: 1.92 μsec if XA ≠ 0, 1.60 μsec if XA = 0

Literal Class: 2

Execution Counter: CH

Registers Affected: PC, XA, EA

Errors Possible: None

Symbolic Notation: If XA ≠ 0, then XA - 1 → XA and EA → PC

Description: The contents of the index A register are compared with zero. If the contents of the index A register equal zero, the instruction terminates and program sequence continues to the next instruction. If the contents of the index A registor are not equal to zero, the contents of the index A register are decremented by 1 and the effective address of the instruction replaces the previous contents of the program counter register (effecting a program branch to the location specified by the effective address).

NOTE

> BDA and BDB are useful for controlling indexing when direction of access through an area of memory is not important or when the index is being used only as a counter and not as an element in computing an effective address.

Example:

Compute a logical checksum of the ten words in locations 04630 through 04641, using XOR.

| Location | Op | M | I | XB | XA | Y or DISP | Interpretation |
|---|---|---|---|---|---|---|---|
| | | | | (Bits) | | | |
| • | | | | | | | |
| • | | | | | | | |
| • | | | | | | | |
| 04205 | LXA | 2 | 0 | 0 | 0 | 0011 | 9 → XA |
| 04206 | LDA | 2 | 0 | 0 | 0 | 0000 | 0 → A |
| 04207 | XOR | 4 | 0 | 0 | 1 | 0421 | A ⊕ (EA) → A |
| 04210 | BDA | 4 | 0 | 0 | 0 | 7777 | If XA ≠ 0; XA - 1 → XA and branch to 04207 |
| | • | | | • | | | |
| 04211 | • | | | • | | | |
| | • | | | • | | | |
| | • | | | • | | | |

Access to the table is bottom (higher address) to top (lower address).

Name: Branch and Decrement Index B

Mnemonic: BDB

Op Code: 73

Timing: 1.92 μsec if XB ≠ 0, 1.60 μsec if XB = 0

Literal Class: 2

Execution Counter: CH

Registers Affected: PC, XB, EA

Errors Possible: None

Symbolic Notation: If XB ≠ 0, then XB - 1 → XB and EA → PC

Description: The contents of the index B register are compared with zero. If the contents of the index B register equal zero, the instruction terminates and program sequence continues to the next instruction. If the contents of the index B register are not equal to zero, the contents of the index B register are decremented by 1 and the effective address of the instruction replaces the previous contents of the program counter register (effecting a program branch to the location specified by the effective address). The notes in the description of the BDA instruction also apply to BDB.

Example: Use the index B register as a counter to control execution of a subroutine six consecutive times.

| Location | Op | M | Tag | Y | Interpretation |
|---|---|---|---|---|---|
| • | | | | | |
| • | | | | | |
| • | | | | | |
| 05763 | LXB | 2 | | 00005 | 5 → XB |
| 05764 | BSP | | SUBR | | Branch to SUBR |
| 05765 | BDB | 4 | | 07777 | If XB ≠ 0, XB - 1 → XB |
| • | | | | | and branch to 05764. |
| • | | | | | |
| • | | | | | |
| • | | | | | |

## BRANCH INSTRUCTIONS

Branch instructions can be used to alter the sequence of instruction execution, either conditionally or unconditionally. When a branch is unconditional (or conditional and the branch condition is satisfied), the next instruction to be executed following the branch is in the location specified by the effective address of the branch instruction. When a branch is conditional and the condition is not satisifed, the next instruction to be executed is normally the instruction next in sequence following the branch instruction.

A normal branch is accomplished by replacing the contents of the program counter (PC) register, with the effective address in the branch instruction.

Abnormal conditions that affect branching are included in the individual instruction descriptions.

The branch instructions are identified in the following list, including branch instructions described in another grouping, which are denoted by an asterisk.

| Instruction Name | Mnemonic |
|---|---|
| Branch on Negative A Register | BRN |
| Branch on Zero A Register | BRZ |
| Branch Unconditionally | BRU |
| *Branch and Save Place | BSP |
| *Branch and Save Region of K Words | BSR |
| *Return From Subroutine | RFS |
| *Return From Interrupt | RFI |
| *Branch and Decrement Index A Register | BDA |
| *Branch and Decrement Index B Register | BDB |
| *Compare With Memory | CWM |
| *No Operation | NOP |

Name: Branch on Negative A Register

Mnemonic: BRN

Op Code: 23

Timing: 1.60 μsec

Literal Class: 2

Execution Counter: CD

Registers Affected: A, PC, PL00 (EMB), EA, FNC, CPS14 (FNV)

Errors Possible: Fence violation (FNV)

Symbolic Notation: None (given in description of test results)

Description: The set/reset condition of the sign bit of the A register is tested with one of the following results:

a. If bit 00 of the A register is 1 (negative) and the effective address is within the limits of the memory protection fence (FNC), the EMB bit (bit 00) of the PL register is reset and the effective address replaces the contents of the PC register; i.e.,

   If A00 = 1, then 0 → EMB and EA → PC

b. If bit 00 of the A register is 1 (negative) and the effective address is outside the limits of the memory protection fence (FNC), the action of BRN is dependent on memory fence violations and the setting of the EMB bit of the privilege and level register:

(1) If EMB = 0, the address in the PC register is inside the fence, and the effective address is outside the fence; the FNV bit of the CPS register is set (1 → CPS14), which activates a level 01 interrupt, and the PC register is unchanged.

(2) If EMB = 1, the address of the PC register is inside the fence, and the effective address is outside the fence, BRN branches and EMB is unchanged.

(3) If the address in the PC register and the effective address of the BRN are both outside the limits of the fence, BRN branches regardless of EMB, and EMB is unchanged.

If EMB = 1, EA → PC.

If both PC and EA are outside the fence, EA → PC.

If EMB = 0, PC is inside the fence and EA is outside the fence, 1 → FNV (level 01 interrupt).

(4) If bit 00 of the A register is 0 (positive), no further action occurs and operation proceeds to the next instruction in sequence.

4.155

Name: Branch on Zero A Register

Mnemonic: BRZ

Op Code: 24

Timing: 1.60 μsec

Literal Class: 2

Execution Counter: CD

Registers Affected: A, PC, EA, FNC, PL00 (EMB), CPS14 (FNV)

Errors Possible: Fence violation (FNV)

Symbolic Notation: None (given in descriptions of branch conditions)

Description: The contents of the A register are tested for 00000000 with one of the following results:

a. If the A register contains only zero bits and the effective address is within the limits of the memory protection fence (FNC), the EMB bit of the PL register is reset (0 → PL00) and the effective address replaces the contents of the PC register; i.e.,

   EA → PC
   0 → EMB

b. If the A register contains only zero bits and the effective address is outside the limits of the memory protection fence (FNC), the action of BRZ is dependent on memory fence violations and the setting of the EMB bit of the privilege and level register:

   (1) If EMB = 0, the address in the PC register is inside the fence, and the effective address is outside the fence, the FNV bit of the CPS register is set (1 → CPS14) activating a level 01 interrupt, and the PC register is unchanged.

(2) If EMB = 1, the address of the PC register is inside the fence, and the effective address is outside the fence, BRN branches and EMB is unchanged.

(3) If the address in the PC register and the effective address of the BRN are both outside the limits of the fence, BRN branches regardless of EMB, and EMB is unchanged.

If EMB = 1, EA → PC.

If both PC and EA are outside the fence, EA → PC.

If EMB = 0, PC is inside the fence, and EA is outside the fence; 1 → FNV (level 01 interrupt).

c. If the A register contains one or more binary one bits, no further action occurs and operation proceeds to the next instruction in sequence.

Name: Branch Unconditionally
Mnemonic: BRU
Op Code: 22
Timing: 1.28 μsec in mode 0, 2.24 μsec in mode 1, 1.60 μsec in modes 3 through 7
Literal Class: Not applicable -- literal class is not used
Execution Counter: CD
Registers Affected: EA, PC, FNC, CPS14 (FNV)
Errors Possible: Memory fence violation (FNV)
Symbolic Notation:

```
If FAOK = 1 + EMP = 1 + IAOK = 0, then
   EA → PC
If FAOK = 0 & EMB = 0 & IAOK = 1, then
   PC → EA
   1 → CPS14
```

Description: Memory fence values are checked, and if satisfactory, program control transfers to the location specified in the instruction.

The effective address is calculated and compared with memory fence parameters, and the executive mode bit is sensed. If program control (PC) is outside the memory fence, if the program has executive privilege (EMB = 1), or if program control and the effective address are both within the memory fence, then the branch is executed by loading the program counter with the calculated effective address. If program control is within the memory fence, the effective address is outside the memory fence, and the program does not have executive privilege (EMB = 0), then the branch is not executed and the fence violation bit is set (1 → CPS14).

NOTE

> The preceding information describes the intrinsic operation of the BRU instruction. There are also two generic operations of the BRU instruction that are defined by the FOX 1 Assembler in separate (expanded) mnemonics. These are:
>
> a. Return From Subroutine (RFS) -- Branch from the end of a subroutinte back to the calling program using the literal mode and an address of zero.
>
> b. No Operation (NOP) -- Branch to the next instruction in sequence using PC relative mode and a displacement of one.

LINKAGE INSTRUCTIONS

Branches in program control to enter or return from a subroutine are executed by four stored program instructions and one wired instruction. The instructions are:

| Instruction Name | Mnemonic |
|---|---|
| Branch and Save Place | BSP |
| Branch and Save Region | BSR |
| Return From Subroutine | RFS |
| Program Interrupt | PI (wired -- not programmable) |
| Return From Interrupt | RFI |

Name: Branch and Save Place

Mnemonic: BSP

Op Code: 26

Timing: 3.20 μsec normally, 4.48 μsec if trapping occurs

Literal Class: Special

Execution Counter: CC

Registers Affected: EA, PC, XT, (XT new), (XT new + 1), FNC, PL00(EMB), CPS14 (FNV)

Errors Possible: Memory Fence violation (FNV)

Symbolic Notation; XT + 1 → XT

PC → XT09-23 and $001_8$ → XT00-08

EA → PC

Description: Program control transfers unconditionally to a subroutine, and a means of returning control (at the conclusion of the subroutine) to the instruction immediately following the BSP is provided. BSP, along with BSR and RFS, are used to manipulate the XT register, which controls the expansion and contraction of the push-down stack. Generally, XT is not altered by any other instructions.

Execution of BSP causes the following:

a. XT + 1 → XT

This operation creates a new top-of-stack address, thus reserving a space for storage of a return address.

b. PC → XT09-23<br>
001 → XT00-08

This operation stores the return address (PC) into the word at the top-of-the-stack. At this point, PC contains the address of the word next in sequence after BSP. Also, the value 001 is set in bits 00 through 08 of that word. This value specifies the number of words reserved in the stack for this particular subroutine entry. Upon exit from the subroutine, the top-of-stack address will be decremented by that number. (Refer to the description of the RFS instruction.)

c. EA → PC

This operation is the unconditional branch to a subroutine.

Example:

Assume that the instruction BSP SUBR is in location 05731 and XT = 06100. Execution of the instruction gives the following:

| Before Execution | After Execution |
|---|---|
| XT = 06100 | XT = 06101 |
| PC = 05732 | PC = SUBR |
| (06101) = XXXXXXXX | (06101) = 00105732 |

Two special cases apply to the Branch And Save Place instruction. In each case, the BSP causes a hardware trap to activate a trap handler program. In normal operation, this trapping use of BSP (also BSR) is for calling system subroutines that operate outside the fence limits of the subroutine calling program. The two special cases are:

a. BSP specifies an immediate operand (literal mode, M = 2). This case causes a trap to memory location 00101, in which case the immediate operand specifies the particular subroutine being called.

b. The effective address of the BSP violates the memory fence specified by the memory protection fence register (FNC). It should be noted that this case is effective only if EMB = 0. If EMB = 1, no fence violation is possible.

   The fence-violating effective address specifies the particular subroutine being called.

These special cases cause similar action, as follows:

| Action | Interpretation |
|---|---|
| a. XT + 1 → XT | Establish a new top-of-stack; reserve one word for storage of the return address. |
| b. PC → XT09-23<br>001 → XT00-08 | Save place; the address of the instruction following BSP is stored into the word at the top of the stack and a word count of 1 is set in bits 00 through 08. |
| c. 1 → EMB | Set bit 00 of the privilege and level register; this gives complete privilege to the trap handler. |

NOTE

Trap handlers determine the validity of the subroutine call and then reset EMB before branching to the specified subroutine. Therefore, the memory fence is active during execution of the trap-initiated subroutine. According to the rules of fence violation shown in Table 2-2-3, the trap's branch to the subroutine is allowed and the subroutine can operate outside the fence, but any store instructions must store information only within the fence. When a Return From Subroutine instruction (RFS) returns control to the calling program, the branch is allowed from outside the fence to inside the fence. Thus, trapping does not give the calling program privileges it did not have before the trap.

| | | |
|---|---|---|
| d. | EA → XT + 1<br>or<br>Immediate Operand<br>→ XT + 1 | Save EA or immediate operand in a word-one location higher in memory than the new top-of-stack. |

NOTE

This value is used only to indicate to the trap handler the subroutine being called. Therefore, it is not stored in reserved stack space. This word and additional words in the area beyond the current top-of-stack location can be used as working storage by the subroutine. However, if the subroutine is reentrant or calls another subroutine, working storage must be protected. This can be done through proper use of a BSR instruction.

e. (1) If the special case is: BSP specifies an immediate operand (literal mode, M = 2), the next action is:

00101 → PC (Branch to 00101)

This is a trap to absolute location 00101, which is the subroutine literal trap in the real time executive system and is under control of the operating system.

(2) If the special case is: the effective address of BSP violates the memory fence..., the next action is:

00100 → PC (Branch to 00100)

This is a trap to absolute location 00100, which is the fence trap in the real time executive system and is under control of the operating system.

Special-Case Examples:

a. Execution of the instruction:

| Loc. | Op | M | Y | |
|---|---|---|---|---|
| 07613 | BSP | 2 | 00005 | (Literal Case) |

| Before Execution | After Execution |
|---|---|
| XT = 06101 | XT = 06102 |
| PC = 07614 | PC = 00101 (Trap) |
| (06102) = XXXXXXXX | (06102) = 00107614 |
| (06103) = XXXXXXXX | (06103) = 00000005 |
| EMB = X | EMB = 1 |

Stack allocation by BSP 2 00005 is:

| | | |
|---|---|---|
| 06101 | XXXXXXXX | Previous top-of-stack |
| 06102 | 00107614 | New top-of-stack |
| 06103 | 00000005 | Subroutine specification |

b. Execution of the instruction:

| Loc. | Op | M | Y | |
|---|---|---|---|---|
| 07701 | BSP | 0 | 00312 | (Memory fence violation) |

causes:

| Before Execution | After Execution |
|---|---|
| FNC = 07770400 | FNC = 07770400 |
| XT = 06110 | XT = 06111 |
| PC = 07702 | PC = 00100 (Trap) |
| (06111) = XXXXXXXX | (06111) = 00107702 |
| (06112) = XXXXXXXX | (06112) = 00000312 |
| EMB = 0 | EMB = 1 |

Note that the memory fence specifies operation within locations 04000 and 07770.

Stack allocation by BSP 0 00312 is:

| | | |
|---|---|---|
| 06110 | XXXXXXXX | Previous top-of-stack |
| 06111 | 00107702 | New top-of-stack |
| 06112 | 00000312 | Subroutine specification |

If, prior to execution, $77760 \leq (XT)+1 \leq 77777$, then execution of a BSP will cause invalid data to be placed in PC, XT, and (XT+1); this does not, however, invalidate the protection features of FOX 1.

If a fence violation occurs on the attempted store into (XT+1), FNV will be set and the branch will not be taken (PC is restored).

A programming example using BSP as a subroutine call is included with the description of RFS.

BSP is similar to the Branch and Save Region instruction.

Name: Branch and Save Region

Mnemonic: BSR

Op Code: 25

Timing: 4.16 μsec normally, 5.44 μsec if trapping occurs

Literal Class: Special

Execution Counter: CC

Registers Affected: EA, PC, XT, FNC, PL00(EMB), CPS14(FNV)

Errors Possible: Fence violation (FNV)

Symbolic Notation: XT + K → XT, where K equals the number of words in saved region
PC → XT09-23 and K → XT00-08
EA → PC

Description: Program control transfers unconditionally to a subroutine, and a means of returning control (at the conclusion of the subroutine) to the instruction that immediately follows the BSR instruction in memory is provided. In addition, BSR reserves a memory area for storage of data passed to the subroutine by the calling program and for saving the address to which control is to return. BSR, along with BSP and RFS, are used to manipulate the XT register, which controls expansion and contraction of the push-down stack. Generally, XT is not altered by any other instructions. The format of the two-word BSR instruction is:

WORD 1

<table>
<tr><td colspan="6">25</td><td colspan="18">EFFECTIVE ADDRESS PARAMETERS</td></tr>
<tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td></tr>
</table>

WORD 2

<table>
<tr><td colspan="9">K</td><td colspan="15">NOT USED</td></tr>
<tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td></tr>
</table>

Word 1 is in standard instruction-word format. It may specify absolute or relative addressing; however, specification of a literal operand (M = 2) is a special case.

In word 2, only bits 00 through 08 are used. Bits 09 through 23 are ignored. The contents of bits 00 through 08 (K) specify the number of memory words to be reserved in the stack. The value of K is equal to L + 1, where L is the number of words in the reserved data storage area. The extra word is used for saving the return address.

For reentrant subroutines that use the stack area higher in memory than the address in XT for working storage, or for any subroutine that uses this working storage area and calls another subroutine, the value of K must be sufficient to include the working storage and the subroutine arguments being passed to the subroutine being called, plus one location for the return address. For example, if a subroutine uses five words of working storage in locations XT + 1, ..., XT + 5, and calls another subroutine that requires reservation of three words for arguments, K would then be nine ($011_8$). A reentrant subroutine example is given later in this discussion.

Execution of BSR causes the following:

a. XT + K → XT

   This operation creates a new top-of-stack address. Thus, the memory area from the old top-of-stack address through the new top-of-stack address is reserved.

b. PC → XT09-23
   K → XT00-08

This operation stores the return address into the word at the top of the stack. Also, K is stored in bits 00 through 08 of this word in the following form:

<table>
<tr><td colspan="9">K</td><td colspan="15">RETURN ADDRESS</td></tr>
<tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td></tr>
</table>

Upon exit from the subroutine, the top-of-stack address will be decremented by K. (Refer to the RFS instruction.)

c. EA → PC

This operation is the unconditional branch to a subroutine.

Examples:

a. Assume the following BSR instruction is executed while XT = 06100.

| Location | Instruction | Interpretation |
|---|---|---|
| • | | |
| • | | |
| • | | |
| 05631 | BSR SUBR | First word of BSR Inst. |
| 05632 | 012 XXXXX | K = 10 (decimal) |

| Before Execution | After Execution |
|---|---|
| XT = 06100 | XT = 06112 |
| PC = 05633 | PC = SUBR |
| (06112) = XXXXX | (06112) = $\underbrace{012}_{\text{K}}\underbrace{05633}_{\text{Return Address}}$ |

4.174

b. This example uses the stack area beyond the address specified by XT as working storage. This subroutine is reentrant in that the subroutine call reserves sufficient space for both the working storage of a previous call to the subroutine and for the two arguments passed to the subroutine on each call.

This reentrant subroutine has two arguments passed to it in the push-down stack. It computes (Argument $1)^2$ + (Argument $2)^2$ and returns the double-precision result in the A and E registers.

Subroutine Calling Sequence

```
         Op                           Y DISP
Loc.     Code    M    I   XB   XA    or Symbol   Interpretation
                        (Bits)
          .
          .
          .
         LDL                          ARG1      Arguments 1 and 2 are
                                                loaded into A and E.

         STL     6    0    0    0     0003      Store arguments into
                      XT + 3                    XT + 3 and XT + 4.

         BSR                          SUBR  }   Two-word BSR instruction:
         00      5    0    0    0     0000  }   reserves five words.
          .
          .
          .
ARG1             XXXXXXXX                   }   Arguments passed to
ARG2             XXXXXXXX                   }   subroutine.
```

## Subroutine

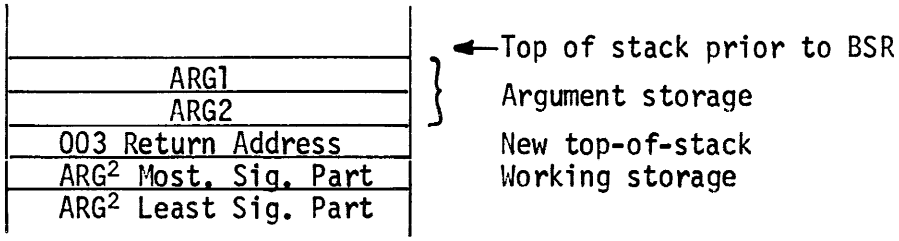
| Loc. | Op Code | M | I | XB | XA | Y DISP or Symbol | Interpretation |
|---|---|---|---|---|---|---|---|
| | | | | (Bits) | | | |
| SUBR | LDA | 6 | 0 | 0 | 0 | 7776<br>XT - 2 | ARG1 → A |
| | MPY | 6 | 0 | 0 | 0 | 7776<br>XT - 2 | ARG1$^2$ → A,E |
| | STL | 6 | 0 | 0 | 0 | 0001<br>XT + 1 | Store ARG1$^2$ →<br>Working Storage |
| | LDA | 6 | 0 | 0 | 0 | 7777<br>XT - 1 | ARG2 → A |
| | MPY | 6 | 0 | 0 | 0 | 7777<br>XT - 1 | ARG2$^2$ → A,E |
| | ADL | 6 | 0 | 0 | 0 | 0001<br>XT + 1 | Result → A,E |
| | RFS | | | | | | Return From Subroutine |

NOTES

- Note that, in the calling program, the arguments were stored in the stack by use of a positive displacement from XT. In the subroutine, the same arguments are referenced by use of negative displacements from XT. This is caused by the change made in XT by BSR.

- If the calling program were using n words of working storage (i.e., XT + 1, ..., XT + n), then the arguments would have been stored into (XT + n + 1 and XT + n + 2 and the BSR would reserve n + 3 words in the stack.

- Stack allocation and use in the example as shown is:



Two special cases apply to the branch and save region instruction. In each case, the BSR causes a hardware trap to activate a trap handler program. In normal operation, this trapping use of BSR (also BSP) is for calling system subroutines that operate outside the fence limits of the subroutine calling program. The two special cases are:

a. BSR specifies an immediate operand (literal mode, M = 2). This case causes a trap to memory location 00101, in which case the immediate operand specifies the particular subroutine being called.

b. The effective address of the BSR violates the memory fence specified by the memory protection fence register (FNC). It should be noted that this case is effective only if EMB = 0. If EMB = 1, no fence violation is possible.

The fence violating effective address specifies the particular subroutine being called.

These special cases cause similar action, as follows:

| | Action | Interpretation |
|---|---|---|
| a. | XT + K → XT | Establish a new top of stack; reserve K words. |

b. PC → (XT09-23)
K → (XT00-08)

Save place; the address of the instruction following BSR is stored into the word at the top of the stack and a word count of K is set in bits 00 through 08.

c. 1 → EMB

Set bit 00 of the privilege and level register; this gives complete privilege to the trap handler.

NOTE

Trap handlers determine the validity of the subroutine call and then reset EMB before branching to the specified subroutine. Therefore, the memory fence is active during execution of the trap-initiated subroutine. According to the rules of fence violation shown in Table 2-2-3, the trap's branch to the subroutine is allowed and the subroutine can operate outside the fence, but any store instructions must store information only within the fence. When a Return From Subroutine instruction (RFS) returns control to the calling program, the branch is allowed from outside the fence to inside the fence. Thus, trapping does not give the calling program privileges it did not have before the trap.

d. EA → XT + 1
or
Immediate
Operand → XT + 1

Save EA or Immediate Operand in a word-one location higher in memory than the new top of stack.

NOTE

This value is used only to indicate, to the trap handler, the subroutine being called. Therefore, it is not stored in reserved stack space. This word and additional words in the area beyond the current top-of-stack location can be used as working storage by the subroutine. However, if the subroutine is reentrant or calls another subroutine, working storage must be protected. This is done by making K of the BSR instruction large enough to accommodate the working storage of the subroutine being executed, plus the storage of arguments for the subroutine being called, plus one for the return address.

e. (1) If the special case is such that BSR specifies an immediate operand (literal, mode 2), the next action is:

00101 → PC (branch to 00101)

This is a trap to absolute location 00101, which is the subroutine literal trap in the FOX 1 Real Time Executive System and is under control of the programming system.

(2) If the special case is such that the effective address of BSP violates the memory fence, the next action is:

00100 → PC (branch to 00100)

This is a trap to absolute location 00100, which is the fence trap in the Real Time Executive System and is under control of the programming system.

Special case examples:

a. Execution of the instruction:

| Location | Op | M | Y | (Literal case) |
|---|---|---|---|---|
| 07613 | BSR | 2 | 00005 | |
| 07614 | 013 | X | XXXXX | |

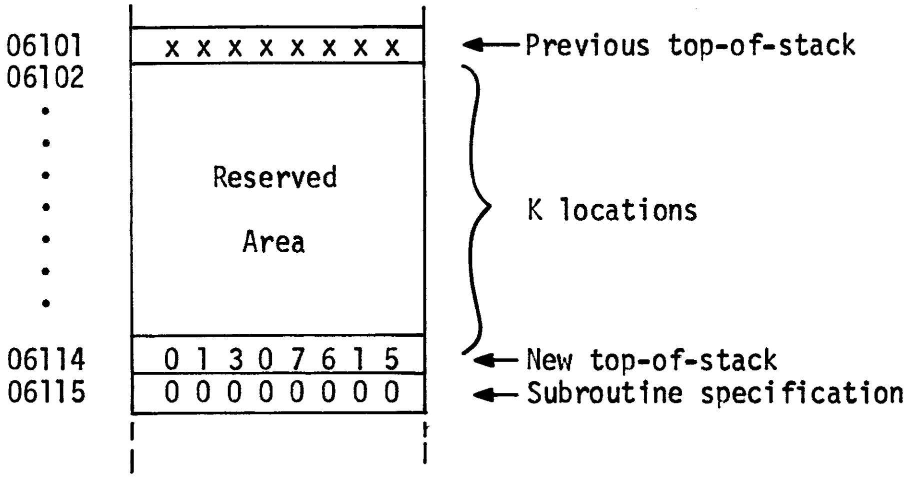
Before Execution

XT = 06101
PC = 07615
(06114) = XXXXXXXX
(06115) = XXXXXXXX
EMB = X

After Execution

XT = 06114
PC = 00101 (Trap)
(06114) = 01307615
(06115) = 00000005
EBM = 1

Stack allocation by this instruction is:



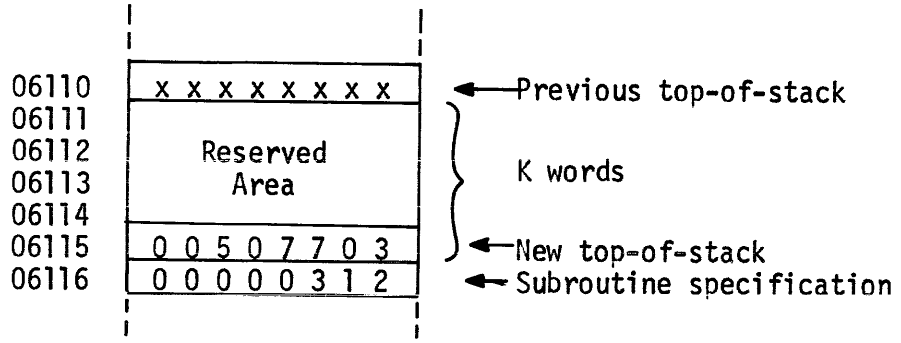
b. Execution of the instruction:

| Location | Op | M | Y | |
|---|---|---|---|---|
| 07701 | BSR | 0 | 00312 | (Memory fence violation) |
| 07702 | 005 | 0 | 00000 | |

| Before Execution | After Execution |
|---|---|
| FNC = 07770400 | FNC = 07770400 |
| XT = 06110 | XT = 06115 |
| PC = 07703 | PC = 01100 Trap |
| (06115) = XXXXXXXX | (06115) = 00507703 |
| (06116) = XXXXXXXX | (06116) = 00000312 |
| EMB = 0 | EMB = 1 |

Note that the memory fence specifies operation within locations 04000 through 07770.

Stack allocation by this instruction is:



BSR can affect XT, (XT) + K, (XT) + K + 1, PC, and PL00.

NOTE

Within the subroutine to which BSR branches, references to the reserved area can be made by using a negative displacement from the XT base.

Example:

```
LDA 6 0777
```

Under the circumstances of Example b., this instruction references location 06114.

If, prior to execution, $77760 \leq (XT) + K \leq 77777$, then execution of a BSR will cause invalid data to be placed in PC, XT, and (XT + K); this does not, however, invalidate the protection features of FOX 1.

If a fence violation occurs on the attempted store into $(XT_{new})$, FNV will be set and the branch will not be taken (PC is restored).

Address Stop operation is temporarily inhibited during the fetch of the implicitly addressed parameter (K) of the BSR instruction.

Name: Return From Subroutine
Mnemonic: RFS
Op Code: 22, Mode = 2, Effective Address = 00000
Timing: 3.52 μsec
Literal Class: Special
Execution Counter: CC
Registers Affected: XT, EA, PC, FNC, CPS14 (FNV)
Errors Possible: Memory fence violation (FNV)
Symbolic Notation: If FAOK = 1 + EMB = 1 + IAOK = 0, then

```
    XT - K → PC
    XT -K → XT
If FAOK = 0 & EMB = 0 & IAOK = 1, then
    PC → EA
    1 → CPS14
```

Description: Program control is returned to the instruction succeeding the branch (BSR or BSP) instruction in the subroutine-calling portion of the program unit. This instruction is an extended mnemonic or special case of the Branch Unconditionally (BRU) instruction recognized by the FOX 1 Assembler and coded as a BRU in the literal mode with an address of zero ($22200000_8$).

Every subroutine terminates with an RFS instruction. This instruction decrements the XT by the value in bits 00 to 08 of the current stack pointer (K). This restores the previous top-of-stack. At the same time, RFS causes a branch from the subroutine to the address in bits 09 to 23 of the current stack pointer. This is a return to the location following the BSR or BSP in the calling program unit.

The RFS instruction effects a branch on the basis of information established by a previously executed BSP or BSR instruction (i.e., return from a subroutine or trap routine). This subroutine return is effected as follows:

$$(XT09\text{-}23) \rightarrow PC$$
$$XT - (XT00\text{-}08) \rightarrow XT$$

Bits 09 through 23 of the memory word whose address is in the XT register replace the contents of the PC register. The contents of the XT register normally point to a memory location that contains the return location established by a BSP or BSR instruction. Thus, for subroutine usage, for every BSP or BSR instruction that branches to a subroutine, there is a corresponding RFS instruction in the subroutine. The linkage between the two instructions is provided by the address in the XT register.

Example using BSP, XT, and RFS:

a. Subroutine Calling Program (XT = 06703)

| Location | Instruction | Interpretation |
|---|---|---|
| . | . | |
| . | . | |
| . | . | |
| 05042 | . | |
| 05043 | 26 0 05100 | BSP, Branch to subroutine beginning at location 05100. Return address is 05044. |

After Execution of BSP

| | |
|---|---|
| PC = 05100 | Location of first instruction of subroutine. |
| XT = 06704<br>(XT) = 00105044 | New top-of-stack points to the top-of-stack location. Bits 00 through 08 of the word at the top of the stack specify number of words used in the stack; bits 09 through 23 specify return address. |

b. Subroutine

| Location | Instruction | Interpretation |
|---|---|---|
| 05100 | . | First location of subroutine |
| | . | |
| | . | |
| | . | Subroutine |
| | . | |
| | . | |
| | . | |
| | 22 2 00000 | RFS; end of subroutine. |

After Execution of RFS

| | |
|---|---|
| PC = 05044 | Return address taken from (XT). |
| XT = 06703 | Reset to presubroutine call value. |
| 06704 $(XT)_{Old}$ = 00105044 | Now outside the current stack entry and meaningless. |

Name: Program Interrupt

Mnemonic: PI

Op Code: None -- wired instruction

Timing: 37.44 μsec

Literal Class: None

Execution Counter: CG

Registers Affected: FNC, PL, PC, XC, XT, XB, A, E, CPI

Errors Possible: None

Symbolic Notation: v → EA
FNC → v
PL → v + 1
PC → v + 2
XC → v + 3
XT → v + 4
XB → v + 5
XA → v + 6
A → v + 7
E → v + 8
CPI → v + 9
v + 10 → FNC
v + 11 → PL
v + 12 → PC
v + 13 → XC
v + 14 → XT
v + 15 → XB

Description: This hard-wired instruction is executed at the conclusion of a stored-program instruction, following a program interrupt request which the interrupt hardware logic has determined to be at a higher priority level than the current program level. The interrupt vector (v) address is obtained from the interrupting level. The contents of ten addressable registers (addresses 77766 through 77777) are stored in the memory locations specified by the interrupt vector, v, and v + 1, v + 2,..., v + 9. These registers contain information that will be needed for subsequent resumption of the interrupt program. After the ten registers are stored, the contents of the next six memory locations (v + 10 through v + 15) are loaded into the registers with addresses 77766 through 77773. These six words provide operating environment assignments and initial operating conditions for the interrupt handler. Registers with addresses 77760 through 77765 and 77774 through 77777 are not changed. Since both the privilege and level and program counter registers are among the registers given new values by this action, the correct priority (IS field of PL) and the location of the first instruction (PC) of the interrupt handler program are obtained automatically. Then the interrupt handler begins execution unless an interrupt signal of higher priority arrives in the interrupt matrix during these actions. Refer to Section 5 of this manual for additional details on program interrupt functional operation.

The worst case delay time between an interrupting event and execution of the first instruction of the interrupt handler is 283.60 μsec. This time results from a traced MOV:

$$\begin{array}{lrl} 2.56 + 2.24 \times 64 = & 145.92 & \mu\text{sec} \\ M' = 1\text{: } 2 \times 2.24 & 4.48 & \mu\text{sec} \\ \text{ASE: } 0.64 \times 134 & \underline{85.76} & \mu\text{sec} \\ & 246.16 & \mu\text{sec MOV} \\ & \underline{37.44} & \mu\text{sec PI} \\ & 283.60 & \mu\text{sec} \end{array}$$

Name: Return From Interrupt
Mnemonic: RFI
Op Code: 16
Timing: 23.36 μsec
Literal Class: 2
Execution Counter: CG
Registers Affected: PC, PL00(EMB), PL01(RMP), CPS15(RPV)
Errors Possible: Register privilege violation (RPV)

```
Symbolic Notation: If EMB  = 0 and RMP  = 0, then
                      1 → CPS15 and abort instruction
                   If EMB = 1 + RMP = 1, then
                      (EA) → 77766
                      (EA + 1) → 77767
```

(EA + 2) → 77770

•

•

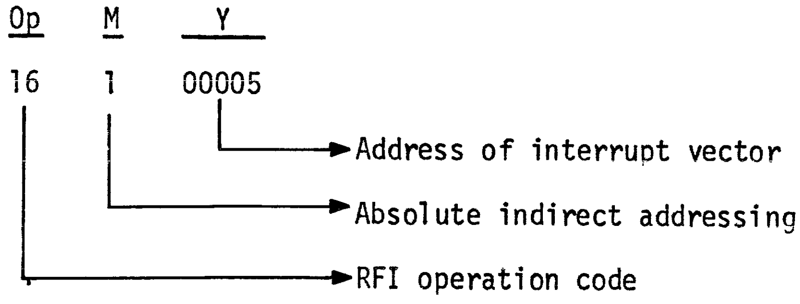
•

(EA + 9) → 77777

Description: This instruction is used to terminate interrupt-initiated programs and to restore preinterrupt contents to the ten registers with addresses 77766 through 77777. The PC register is one of the registers restored; therefore, RFI effects a return to the interrupted program.

Execution of RFI is privileged; that is, it is dependent upon the settings of the two privilege bits: EMB and RMP (bits 00 and 01 of the PL register). If neither EMB nor RMP is set, RFI cannot be executed. An attempt to execute RFI under this condition is a register privilege violation. This sets the RPV bit (bit 15) of the CPS register, which initiates a level 01 interrupt.

The effective address of the RFI instruction must specify the first location of the memory area used by hardware interrupt action as storage for the ten registers when interrupt occurred. This address is contained in the dedicated memory location assigned to the interrupt level from which RFI is returning; i.e., if operation is at interrupt level 05, for example, the effective address for RFI is the level 05 interrupt vector stored in location 00005. (Refer to the discussion of Interrupt Classes in Section 5.) The RFI instruction, in this case, if it has privilege, could be:



If location 00005 contains 00006317, the result is:

(06317) → FNC
(06320) → PL
(06321) → PC
(06322) → XC
(06323) → XT
(06324) → XB
(06325) → XA
(06326) → A
(06327) → E
(06330) → CPI

Refer to Section 5 of this manual for additional information on program interrupt operation sequences and related application of the RFI instruction.

## COMPARE WITH MEMORY INSTRUCTION

This instruction is microprogrammed to perform a large number of operations. Therefore, it is discussed as a class of instruction.

Name: Compare With Memory
Mnemonic: CWM
Op Code: 20
Timing: 5.44 μsec for single-precision; 5.76 μsec for double-precision when the first words differ; 7.68 μsec for double-precision when the first words do not differ
Literal Class: 1 for EA word 1, literal class 2 for EA word 2
Execution Counter: CE
Registers Affected: PC

4.192

Errors Possible: None

Symbolic Notation: Depends upon microprogramming

Description: This two-word instruction performs comparisons between the contents of arithmetic registers and the contents of memory locations specified in an effective address of the instruction. The various operations performed as a result of the comparisons are microcoded. The instruction has the following format:

WORD 1

| OP | | | | | | M | | | EA1 PARAMETERS | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

WORD 2

| R | | L | | | B | M | | | EA2 PARAMETERS | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

where:

OP: Operation Code. OP = $20_8$ = CWM

M: Mode of Addressing. When M = 2, word 1 is literal class 1 and word 2 is literal class 2.

EA1: Effective Address 1. The address of the location containing one of the operands (designated MV for memory value) to be compared.

R: Register Field. Specifies the register (s) containing the second operand (designated RV for register value) to be compared.

| R | Location of Comparison Operands | |
|---|---|---|
| | Operand 1 | Operand 2 |
| 00 | Literal value 00000000 | (EA1) |
| 01 | A | (EA1) |
| 10 | E | (EA1) |
| 11 | A,E | (EA1),(EA1) + 1 |

L: Logic Field. Determines the logical relationship between the register value (RV) and the memory value (MV), which will satisfy the comparison.

| L | Logical Relationship |
|---|---|
| 000 | RV00 = MV00 (only sign bits compared) |
| 001 | RV = MV |
| 010 | RV ≠ MV |
| 011 | RV < MV |
| 100 | RV > MV |
| 101 | RV ≤ MV |
| 110 | RV ≥ MV |
| 111 | Triple compare with an escape for each condition. The B field is ignored and the following operations occur: |

If RV < MV, then EA2 → PC (Branch)
If RV > MV, then PC + 1 → PC (Skip)
If RV = MV, then PC → PC (Continue)

B: Branch Field. This bit specifies the operation to occur (escape) if the comparison determined by the logic field is satisfied. The B field is ignored and the program continues to the next instruction if the comparison is not satisfied. Also, the B field is ignored if the logic field specifies a triple compare (L = 111), since this operation implies three different escapes based on the comparison results. If B = 1, the comparison is satisfied, and the logic field is not a triple compare, then a branch occurs (EA2 → PC). If B = 0, the comparison is satisfied, and the logic field is not a triple compare, then the next instruction is skipped (PC + 1 → PC).

EA2: Effective Address 2. The address to which the program will branch if the comparison is satisifed and branching is specified (B = 1).

Word 1 and bits 06 through 23 of word 2 are in standard instruction format.

The comparisons specified by the logic field (L) are correct for either fixed point or floating point numbers in either short or long format. Correct comparisons of floating point numbers occur because of the 2's complement and excess characteristic format.

The FOX 1 Assembler allows the use of 64 extended mnemonics defining the Compare With Memory variations. The assembler instructions specify two addresses: the memory value address and the branch address: and they generate a two-word instruction with the proper R, L, and B fields, depending on the extended mnemonic used. For skip instructions (B = 0), and (L ≠ 111), the second address (branch address) is not required and, if present, is ignored. Table 2-4-1 lists the CWM extended mnemonics and shows the R, L, and B fields generated for each.

NOTE

When a CWM branches (i.e., on a successful compare with B = 1 in word 2), if the branch address (EA2) violates the memory fence, a level 01 interrupt occurs. However, the illegal branch address is in the PC register at the time of interrupt and the interrupt handler has no means of determining the location of the instruction that causes the interrupt. CWM is the only instruction that causes this situation under normal use.

Table 2-4-1. CWM Extended Mnemonics

| MNEMONIC | NAME | R | L | B |
|---|---|---|---|---|
| BZEQ | Branch If Zero Is Equal To Memory | 00 | 001 | 1 |
| BZNE | Branch If Zero Is Not Equal To Memory | 00 | 010 | 1 |
| BZGT | Branch If Zero Is Greater Than Memory | 00 | 100 | 1 |
| BZLT | Branch If Zero Is Less Than Memory | 00 | 011 | 1 |
| GZGE | Branch If Zero Is Greater Than Or Equal To Memory | 00 | 110 | 1 |
| BZLE | Branch If Zero Is Less Than Or Equal To Memory | 00 | 101 | 1 |
| BZSE | Branch If Zero Sign Equals Memory Sign | 00 | 000 | 1 |
| BAEQ | Branch If A Is Equal To Memory | 01 | 001 | 1 |
| BANE | Branch If A Is Not Equal To Memory | 01 | 010 | 1 |
| BAGT | Branch If A Is Greater Than Memory | 01 | 100 | 1 |
| BALT | Branch If A Is Less Than Memory | 01 | 011 | 1 |
| BAGE | Branch If A Is Greater Than Or Equal To Memory | 01 | 110 | 1 |
| BALE | Branch If A Is Less Than Or Equal To Memory | 01 | 101 | 1 |
| BASE | Branch If A Sign Equals Memory Sign | 01 | 111 | 1 |
| BEEQ | Branch If E Is Equal To Memory | 10 | 001 | 1 |
| BENE | Branch If E Is Not Equal To Memory | 10 | 010 | 1 |
| BEGT | Branch If E Is Greater Than Memory | 10 | 001 | 1 |
| BELT | Branch If E Is Less Than Memory | 10 | 001 | 1 |
| BEGE | Branch If E Is Greater Than Or Equal To Memory | 10 | 110 | 1 |
| BELE | Branch If E Is Less Than Or Equal To Memory | 10 | 101 | 1 |
| BESE | Branch If E Sign Equals Memory Sign | 10 | 000 | 1 |
| BLEQ | Branch If A,E Is Equal To Memory | 11 | 001 | 1 |
| BLNE | Branch If A,E Is Not Equal To Memory | 11 | 010 | 1 |
| BLGT | Branch If A,E Is Greater Than Memory | 11 | 100 | 1 |
| BLLT | Branch If A,E Is Less Than Memory | 11 | 011 | 1 |
| BLGE | Branch If A,E Is Greater Than Or Equal To Memory | 11 | 110 | 1 |
| BLLE | Branch If A,E Is Less Than Or Equal To Memory | 11 | 101 | 1 |
| BLSE | Branch If A,E Sign Equals Memory Sign | 11 | 000 | 1 |

Table 2-4-1. CWM Extended Mnemonics (contd)

| MNEMONIC | NAME | R | L | B |
|---|---|---|---|---|
| SEZQ | Skip If Zero Is Equal To Memory | 00 | 001 | 0 |
| SZNE | Skip If Zero Is Not Equal To Memory | 00 | 010 | 0 |
| SZGT | Skip If Zero Is Greater Than Memory | 00 | 100 | 0 |
| SZLT | Skip If Zero Is Less Than Memory | 00 | 011 | 0 |
| SZGE | Skip If Zero Is Greater Than Or Equal To Memory | 00 | 110 | 0 |
| SZLE | Skip If Zero Is Less Than Or Equal To Memory | 00 | 101 | 0 |
| SZSE | Skip If Zero Sign Equals Memory Sign | 00 | 000 | 0 |
| SAEQ | Skip If A Is Equal To Memory | 01 | 001 | 0 |
| SANE | Skip If A Is Not Equal To Memory | 01 | 010 | 0 |
| SAGT | Skip If A Is Greater Than Memory | 01 | 100 | 0 |
| SALT | Skip If A Is Less Than Memory | 01 | 011 | 0 |
| SAGE | Skip If A Is Greater Than Or Equal To Memory | 01 | 110 | 0 |
| SALE | Skip If A Is Less Than Or Equal To Memory | 01 | 101 | 0 |
| SASE | Skip If A Sign Equals Memory Sign | 01 | 000 | 0 |
| SEEQ | Skip If E Is Equal To Memory | 10 | 001 | 0 |
| SENE | Skip If E Is Not Equal To Memory | 10 | 010 | 0 |
| SEGT | Skip If E Is Greater Than Memory | 10 | 100 | 0 |
| SELT | Skip If E Is Less Than Memory | 10 | 011 | 0 |
| SEGE | Skip If E Is Greater Than Or Equal To Memory | 10 | 110 | 0 |
| SELE | Skip If E Is Less Than Or Equal To Memory | 10 | 101 | 0 |
| SESE | Skip If E Sign Equals Memory Sign | 10 | 000 | 0 |
| SLEQ | Skip If A,E Is Equal To Memory | 11 | 001 | 0 |
| SLNE | Skip If A,E Is Not Equal To Memory | 11 | 010 | 0 |
| SLGT | Skip If A,E Is Greater Than Memory | 11 | 100 | 0 |
| SLLT | Skip If A,E Is Less Than Memory | 11 | 011 | 0 |
| SLGE | Skip If A,E Is Greater Than Or Equal To Memory | 11 | 110 | 0 |
| SLLE | Skip If A,E Is Less Than Or Equal To Memory | 11 | 101 | 0 |
| SLSE | Skip If A,E Sign Equals Memory Sign | 11 | 000 | 0 |
| TWBZ | Three-Way Branch On Zero Minus Memory | 00 | 111 | 0 |
| TWBA | Three-Way Branch On A Minus Memory | 01 | 111 | 0 |
| TWBE | Three-Way Branch On E Minus Mmeory | 10 | 111 | 0 |
| TWBL | Three-Way Branch On A,E Minus Memory | 11 | 111 | 0 |

## BIT MANIPULATION INSTRUCTION

Name: Bit Manipulation

Mnemonic: BIT

Op Code: 07

Timing: 6.72 μsec if bit to be modified is not in A; 5.76 μsec if bit is in A

Literal Class: 1 for EA word 1, literal class 2 for EA word 2

Execution Counter: CE

Registers Affected: PC, CPS, (EA)

Errors Possible: None

Symbolic Notation: Depends on microprogramming

Description: This two-word instruction provides the capability of sensing and manipulating the status of any specified bit in any memory location or addressable register. It also includes the capability to conditionally or unconditionally skip the next instruction. BIT has the following two-word format:

WORD 1

<table>
<tr><td colspan="6">07</td><td colspan="3">M1</td><td colspan="15">EA1</td></tr>
<tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td></tr>
</table>

WORD 2

<table>
<tr><td colspan="2">X</td><td colspan="4">OP2</td><td colspan="3">M2</td><td colspan="15">EA2</td></tr>
<tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td></tr>
</table>

Both words of the BIT instruction specify an effective address in the standard formats.

EA1 specifies an operand (contents of EA1), which has the following format:

<table>
<tr><td colspan="19" align="center">NOT USED</td><td colspan="5" align="center">BIT</td></tr>
<tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td></tr>
</table>

Bits 00 through 18 are ignored and bits 19 through 23 of this operand point to the specific bit to be manipulated. If the contents of bits 19 through 23 are greater than $23_{10}$ (or $10111_{2}$), the instruction becomes a no-operation, no bits are changed, and no skipping takes place, even if an unconditional skip were specified.

EA2 specifies the address of the word containing the bit to be manipulated. This address may be a memory address or it may be one of the addresses assigned to an addressable hardware register (refer to the discussion on Major Registers in Section 2). The following conditions control bit manipulation of addressable registers:

a. BIT may be used to sense the status of any bit in any addressable register. This can be used to determine if a skip should be executed.

b. BIT may be used at any time to modify any modifiable bit in the PC, XA, XB, XT, A, E, or CPI registers.

c. If the EMB or RMP bit of the privilege and level register is set, BIT may be used to modify any modifiable bit of the CPS, FNC, or PL registers.

d. If the EMB or SIP bit of the privilege and level register is set, BIT may set or reset any bit in the SIS register.

e. If the EMB or TCP bit of the privilege and level register is set, BIT may modify the ASL and ASU registers.

f. If the BIT instruction attempts to modify privileged registers when the appropriate privilege bit is not set, the corresponding violation bit in the central processor status register is set, and a level 01 interrupt occurs.

g. Attempts to modify the word switch register are allowed, but the result is no-operation.

OP2 (bits 02 through 05 of word 2) defines a secondary op code. Each bit specifies a particular function. The bit specifications of OP2 are shown in Table 2-4-2.

The four OP2 bits combine to form a total of 16 secondary operation codes. The FOX 1 Assembler provides extended mnemonics to specify each of the 16 combinations. When these extended mnemonics are used, the assembler generates a two-word BIT instruction with the proper secondary op code in word 2.

The four-bit op codes, the associated extended mnemonics, and the meaning of each are given in Table 2-4-3.

Table 2-4-2. Secondary Op Code Bit Specifications

| BIT POSITION | SPECIFICATION | INITIAL VALUE OF SB* | VALUE OF OB** | ACTION |
|---|---|---|---|---|
| 2 | Skip or continue when SB = 0 | 0 | 0 | Continue to next instruction. |
| | | 0 | 1 | Skip next instruction. |
| 3 | Skip or continue when SB = 1 | 1 | 0 | Continue to next instruction. |
| | | 1 | 1 | Skip next instruction. |
| 4 | Specifies final status of SB if initial value of SB = 0 | 0 | 0 | Leave SB = 0. |
| | | 0 | 1 | Set SB = 1. |
| 5 | Specifies final status of SB if initial value of SB = 1 | 1 | 0 | Reset SB = 0 |
| | | 1 | 1 | Leave SB = 1 |

* SB = Sensed Bit
** OB = Op Code Bit

Table 2-4-3. BIT Instruction Extended Mnemonics

| SECONDARY OP CODE | EXTENDED MNEMONIC | MEANING (SET = 1, RESET = 0) |
|---|---|---|
| 0000 | RBIT | Reset SB* and continue |
| 0001 | BIT | Leave SB unchanged and continue (NO-OP) |
| 0010 | CBIT | Change SB (alternate) and continue |
| 0011 | SBIT | Set SB and continue |
| 0100 | SKSR | Skip if SB set and reset bit |
| 0101 | SKS | Skip if SB set and leave SB unchanged |
| 0110 | SKSC | Skip if SB set and alternate SB |
| 0111 | SKSS | Skip if SB set and set SB |
| 1000 | SKRR | Skip if SB reset and reset SB |
| 1001 | SKR | Skip if SB reset and leave SB unchanged |
| 1010 | SKRC | Skip if SB reset and alternate SB |
| 1011 | SKRS | Skip if SB reset and set SB |
| 1100 | SKUR | Skip and reset SB |
| 1101 | SKU | Skip unconditionally |
| 1110 | SKUC | Alternate SB and skip unconditionally |
| 1111 | SKUS | Set SB and skip unconditionally |

*SB = Sensed Bit

Example:

Use BIT to sense the status of bit 00 of the CPI register. This bit is set when floating point underflow occurs. If the bit is set, reset it and branch to a location called UND. If it is reset, continue branch to NUND.

| Loc. | Op | M | Address | Explanation |
|---|---|---|---|---|
| | •<br>•<br>• | | | |
| 00500 | BIT | 2 | 00000 | Word 1 of BIT instruction provides literal bit number specification; bit 00 in this case. |
| 00501 | 04<br>(SKSR) | 2 | 77777 | Skip if bit is set, and reset the bit. Literal address 777777 references the CPI register. |
| 00502 | BRU | | NUND | Branch to NUND. |
| 00503 | BRU | | UND | Branch to location UND. |
| 00504 | •<br>•<br>•<br>• | | | |

### BYTE MANIPULATION INSTRUCTION

Name: Byte Manipulation

Mnemonic: BYT

Op Code: 06

Timing: 11.52 μsec + 0.32 μsec × (S + K)
    when RJ = 1 and $0 \leq K \leq 24$
  32.00 μsec + 0.32 μsec × (S + K)
    when RJ = 1 and $25 \leq K \leq 31$

3.84 μsec + 0.64 μsec × (P + K)
  when RJ = 0 and $P \geq S$
3.84 μsec + (0.64 μsec × K) + 0.32 μsec × (P + 5)
  when RJ = 0 and $S > P$

Literal Class: 2

Registers Affected: A register and CPI02-04

Execution Counter: CE

Errors Possible: Byte parity, byte zero, byte overflow

Symbolic Notation: None

Description: Operations are performed between limited fields (or bytes) of two words. One byte is in the A register and the other byte is in the memory location specified by the effective address portion of the instruction. Byte size can be from 1 to 24 bits and can begin at any bit position in either word. The result is left in the A register, unless the instruction is coded to prevent disturbing the contents of the A register.

BYT is a two-word instruction in which word 1 is in standard instruction format and read from memory in the program sequence. Word 2 must reside in the E register when word 1 is fetched from memory and defines both the actions to be executed and the data to be used. The format of these parameters in the E register is as follows:

| X | P | DC | LV | RJ | F | S | K |
|---|---|---|---|---|---|---|---|
| 00 01 02 | 03 04 05 06 07 | 08 | 09 | 10 | 11 12 13 | 14 15 16 17 18 | 19 20 21 22 23 |

E REGISTER PARAMETERS

where:

K: Byte length field specifies the number of bits to be manipulated in both words. The value of the K field normally should not exceed 24. The normal range of values of K is $01 < L < 24$.

If K = 00, the instruction becomes a rather long NOP. If K is in the range from 25 through 31, the least significant bits will be used twice. For example, if K = 25, the right-most bit will be used twice; or, if K = 31, the right-most seven bits will be used twice. Although this will not cause erroneous results for functions of load A, load complement in A, logical AND, and logical OR, errors can occur in operations that produce an overflow or an underflow. Therefore, values of K outside the normal limits of 01 through 24 should be used only with great caution.

S: Most significant bit (leftmost bit) of the byte specified by the effective address in instruction word 1. The range of S is: $00 \leq S \leq 23$. See Note 1.

P: Most significant bit (leftmost bit) of the byte in the A register. The range of P is: $00 \leq P \leq 23$. See Note 1.

F: Function field. The operation performed between the two bytes is specified as follows:

| F | Operation |
|---|---|
| 000 | Load A register |
| 001 | Load A register with the 1's complement |
| 010 | Add EA byte to A byte |
| 011 | Subtract EA byte from A byte |
| 100 | Logical AND |
| 101 | Logical OR |
| 110 | Logical exclusive OR |
| 111 | Logical exclusive OR inverse |

The logical operations are executed bit by bit. The arithmetic operations are performed on unsigned byte data; therefore, subtraction is a 2's complement addition. Arithmetic and logical operations involve carries in the normal manner within the specified byte. Carries are propagated outside the byte only if the RJ field = 1.

RJ: Right-justify field. This bit determines if right-justified operations are performed on the byte in the A register.

When RJ = 1, right-justified operations are performed between the K least significant bits of the A register and the EA byte. The P field is ignored. A carry or borrow may extend outside the A byte field specified by K, and the byte indicators of the central processor indicator register apply to the entire contents of the A register, regardless of the size of the A byte.

When RJ = 0, normal byte operations, rather than right-justified operations, are performed. The P field is used, carries and borrows exist only within the specified bytes, and the CPI indicators apply only to the specified bytes.

LV: Leave field. This bit determines if the central processor indicators (BO, BZ, BP) are to be cleared before the manipulation is executed, or if previous settings of these indicators are allowed to accumulate. If LV = 1, these indicators are not cleared. If LV = 0, these indicators are cleared before execution of the byte function.

DC: Don't-change field. This bit determines if the contents of the A register are changed or not by the byte function. If DC = 1, the contents of the A register are not changed, the operation proceeds, but only the central processor indicator is affected. If DC = 0, the byte function is executed, the results are stored in the A register, and the CPI bits function normally.

X: Unused bits.

NOTES

1. Fields S and P are interpreted modulo 24. Specifying P = 24 instead of P = 0 effects the same operation but increases instruction execution time by 8 μsec.

2. If the relationship of K to S or P is such that [(S or P) + K] > 23, then byte A or byte EA, or both, are treated as circular words; e.g., if P = 20 and K = 6, the byte A would consist of bits 20, 21, 22, 23, 00, 01 in left-to-right order.

The three bits of the central processor indicator register affected by the BYT instruction are:

BO: Byte overflow (CPI04). As the byte manipulation operation (F) proceeds, any arithmetic carry from the high-order bit of the result field will set the BO indicator. This may be a true overflow or an indication of going from negative to positive. BO may be used as the carry from BYT operation to BYT operation when operating in the continue mode.

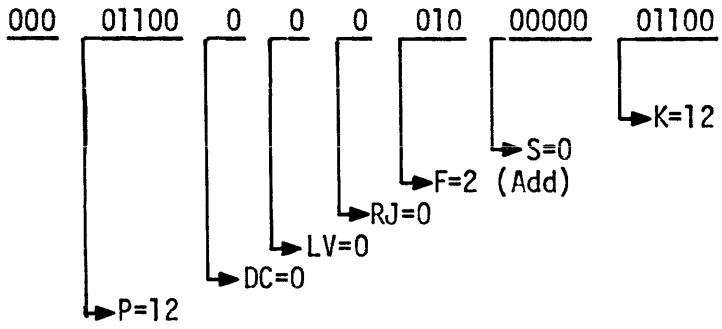
BZ: Byte zero (CPI03). As the operation (F) proceeds, any non-zero result bit will set BZ (BZ = 1). If BZ is clear (BZ = 0) at the end of the instruction, it indicates that the result of the BYT instruction is zero.

BP: Byte parity (CPI02). As the operation proceeds, each non-zero bit of the result changes the state of BP. Thus, BP being set indicates odd parity of the result.

Examples:

a. Use BYT to add bits 00 through 11 of the contents of EA to bits 12 through 23 of the A register.

(1) Load the E register with proper parameters to define the operation. The following word (in binary) is correct:

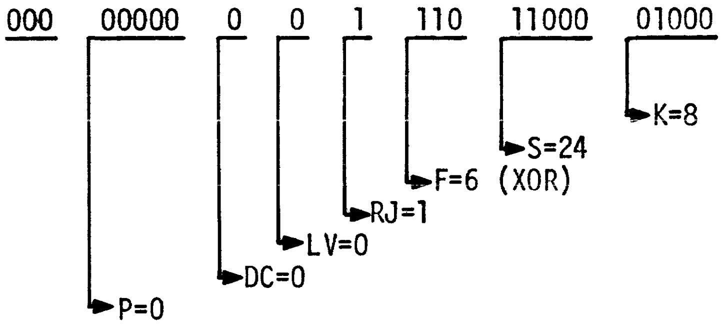


(2) Result of execution of BYT gives the following:

| Before Execution | | After Execution |
|---|---|---|
| A = 73210045 | | A = 73213746 |
| (EA) = 37010054 | | (EA) Unchanged |
| BP = X | Reset prior to execution | BP = 0 (even parity) |
| BZ = X | | BZ = 1 (nonzero results) |
| BO = X | | BO = 0 (no overflow) |

b. Use BYT to logically combine the three 8-bit bytes that make up the contents of EA, using exclusive OR; i.e., (EA00-07) ⊕ (EA08-15) ⊕ (EA16-23) → A16-23.

The parameters in the E register will be:



Note that although S = 24, the following program subtracts 8 from the S field prior to executing the BYT. Also, when RJ = 1, P is ignored and the result will be right-justified in the A register.

The following series of instructions will compute the logical exclusive OR result using the three 8-bit bytes in a word. The addresses used in the program are literals or relative to the PC register.

| Location (PC) | Operation | Mode | Address | Explanation |
|---|---|---|---|---|
| 00501 | LDA | 2 | 00000 | Load A with literal zero |
| 00502 | LXA | 2 | 00002 | Load XA with literal 2 |
| 00503 | LDE | 4 | 00005<br>(PC + 5) | Load E with contents of loc. 00510;<br>(PC + 5) |
| 00504 | SBL | 2 | 00400 | Subtract long, using 0000000000000400; this reduces S-field by 8 |
| 00505 | BYT | 4 | 00004<br>(PC + 4) | Byte manipulation between A register and loc. 00511;<br>(PC + 4) |

| Location (PC) | Operation | Mode | Address | Explanation |
|---|---|---|---|---|
| 00506 | BDA | 4 | 07776<br>(PC - 2) | If (XA) ≠ 0, (XA - 1) → XA; and branch.<br>If (XA) = 0, continue. |
| 00507 | BRU | EXIT | | Branch out of sequence |
| Storage (Octal Values) | | | | |
| 00510 | 00035410 | | | Initial setting for E register |
| 00511 | 43145672 | | | Three 8-bit byte word |

In this example, the three passes through the loop will cause the following.

Pass 1: $\text{Byte}_1$ = 10111010 (EA16-23)
A16-23 = $\underline{00000000}$

Result
A16-23 = 10111010

Pass 2: $\text{Byte}_2$ = 11001011 (EA08-15)
A16-23 = $\underline{10111010}$

Result
A16-23 = 01110001

Pass 3: $\text{Byte}_3$ = 10001100 (EA00-07)
A16-23 = $\underline{01110001}$

Result
A16-23 = 11111101

Final Result: A = 00000375

BP = 1
BZ = 1
BO = 0

## SPECIAL INSTRUCTIONS

The following instructions perform special functions.

| Name | Mnemonic |
|---|---|
| Decrement Memory | DEM |
| Multiple Move | MOV |
| Generate Effective Address | GEA |
| Set Privilege and Level Register | SPL |
| Halt | HLT |
| No Operation | NOP |

Name: Decrement Memory

Mnemonic: DEM

Op Code: 56

Timing: 3.52 μsec with no skip; 3.84 μsec with a skip

Literal Class: 2

Execution Counter: CC

Registers Affected: PC, (EA)

Errors Possible: None

Symbolic Notation: If $(EA) = 0$, $PC + 1 \rightarrow PC$
IF $(EA) \neq 0$, $(EA) - 1 \rightarrow (EA)$

Description: DEM subtracts 1 from the contents of the effective address unless the contents of EA equal zero. If the contents of EA equal zero, the next instruction following DEM is skipped (i.e., execution of DEM when the contents of EA equal zero causes the PC register to be incremented by 1). Because PC is always incremented by 1 prior to instruction execution, DEM effects the skip by incrementing PC once more.

Name: Multiple Move

Mnemonic: MOV

Op Code: 50

Timing: 2.56 μsec + 2.24 μsec × K

Literal Class: 1 for EA word 1 and class 2 for EA word 2.

Execution Counter: CG

Registers Affected: (EA2), (EA2 + 1) .... (EA2 + K - 1)

Errors Possible: None

Symbolic Notation: $(EA1) \rightarrow (EA2)$

$(EA1 + 1) \rightarrow (EA2 + 1)$

•

•

$(EA1 + N) \rightarrow (EA2 + N)$, where $1 \leq N \leq 64$.

Description: MOV is a two-word instruction for moving blocks of words from one storage area to another. The instruction has the following format:

WORD 1

| 50 | M | EA1 |
|---|---|---|
| 00 01 02 03 04 05 | 06 07 08 09 | 10 11 12 13 14 15 16 17 18 19 20 21 22 23 |

WORD 2

| K | M | EA2 |
|---|---|---|
| 00 01 02 03 04 05 | 06 07 08 09 | 10 11 12 13 14 15 16 17 18 19 20 21 22 23 |

MOV causes the contents of the K consecutive words beginning at the effective address of word 1 to replace the K consecutive words beginning at the effective address of word 2. Since K is a 6-bit field, K is subject to the restriction $0 \leq K \leq 63$, where K = 0 functions as though K = 64. If word 1 specifies a literal value (mode 2), then the literal value is placed in K-consecutive location beginning at address EA2. If word 2 specifies a literal, then the literal is used as EA2 (class 2 literal).

Examples:

a. Clear the $36_8$ locations beginning with the location whose address is in the index A register.

| | OP | M | I | XB | XA | DISP |
|---|---|---|---|---|---|---|
| | | | | (bits) | | |
| Word 1: | 50 | 2 | 0 | 0 | 0 | 0000 (Literal value = 0) |
| Word 2: | 36 | 3 | 0 | 0 | 1 | 0000 36 (octal) words,<br>EA2 = (XA) + 0 |

b. Move the $10_8$ words located at PC + $500_8$ through PC + $507_8$ into the $10_8$ locations at the top-of-the-stack (assumes these $10_8$ locations have been reserved by BSR).

| | OP | M | I | XB | XA | DISP |
|---|---|---|---|---|---|---|
| | | | | (bits) | | |
| Word 1: | 50 | 4 | 0 | 0 | 0 | 0500 EA1 = PC + $500_8$ |
| Word 2: | 10 | 6 | 0 | 0 | 0 | 7770 EA2 = XT + $77770_8$<br>or<br>EA2 = XT - $8_{10}$ |

Name: Generate Effective Address

Mnemonic: GEA

Op Code: 01

Timing: 1.28 μsec

Literal Class: 1

Execution Counter: CC

Registers Affected: E

Errors Possible: None

Symbolic Notation: EA → E09-23
0 → E00-08

Description: GEA replaces bits 09 through 23 of the contents of the E register with the generated effective address. Bits 00 through 08 of the E register are cleared to zero.

In a relocatable environment, where actual addresses are not known until a program is loaded into memory. The GEA instruction enables a relative program to generate absolute addresses for Channel IO, for use as an index, or for passing to subroutines.

If the address generated by GEA is to be used as an indirect vector, such as when the address is passed to a subroutine, it is the responsibility of either the calling program or the subroutine to add proper mode bits (M', bits 06 through 08) to the address (usually mode 3 for zero relative) to prevent use of the original instruction mode.

Name: Set Privilege and Level Register
Mnemonic: SPL
Op Code: 17
Timing: 1.92 μsec
Literal Class: 1
Execution Counter: CC
Registers Affected: PL19-23, CPS15
Errors Possible: Register privilege violation (RPV)
Symbolic Notation: If LL $\leq$ (EA19-23) $\leq$ UL, then (EA19-23) → PL19-23.
If not, 1 → CPS15

Description: SPL is used to alter the interrupt status (IS) field of the privilege and level register (PL19-23). The new value for the IS field is contained in bits 19 through 23 of the instruction operand (EA19-23). The PL register contains an upper limit (UL) and a lower limit (LL) for the IS field. The new setting must be within this range. If not, bit 15 of the central processor status register (RPV) is set (1 → CPS15), causing a level 01 interrupt.

See Section 5, Interrupt Structure, for a complete explanation of the use of IS, UL, and LL fields of the PL register.

Name: Halt
Mnemonic: HLT
Op Code: 00
Timing: 1.60 μsec
Literal Class: 1
Execution Counter: CD
Registers Affected: CPS23
Errors Possible: Halt instruction executed (HLT)
Symbolic Notation: 1 → CPS23
Description: Execution of the HLT instruction causes the HLT bit of the central processor status register (CPS23) to be set. Setting the HLT bit initiates a level 00 interrupt prior to the execution of another instruction. If the program in which the HLT instruction appears is operating at level 00 (IS field of the PL register = 00), another interrupt at level 00 is inhibited. This condition causes a computer halt. If the program in which the HLT instruction appears is operating at a level different than 00 (IS > 00), a level 00 interrupt occurs.

Name: No Operation
Mnemonic: NOP
Op Code: 22, Mode: 4, Displacemnt: 00001
Timing: 1.60 μsec
Literal Class: Not applicable -- mode is PC relative
Execution Counter: CD
Registers Affected: EA, PC, FNC, CPS14 (FNV)
Errors Possible: Memory fence violation

```
Symbolic Notation:  If FAOK = 1 + EMB = 1 + IAOK = 1, then
                       PC + 1 → PC
                    If FAOK = 0 & EMB = 0 & IAOK = 0, then
                       PC → EA
                       1 → CPS14
```

Description: Memory fence values are checked, and if satisfactory, program control branches to the next instruction in sequence.

The effective address is calculated and compared with memory fence parameters, and the executive mode bit is sensed. If program control (PC) is outside the memory fence, if the program has executive privilege (EMB = 1), or if program control and the effective address are both within the memory fence, then the branch is executed by incrementing the contents of the program counter. If program control is within the memory fence, the effective address is outside the memory fence, and the program does not have executive privilege (EMB = 0), then the branch is not executed and the fence violation bit is set (1 → CPS14).

The NOP command is an extended mnemonic of the BRU instruction used by the FOX 1 Assembler and accepted by the hardware as a BRU microinstruction.

## PROGRAMMED INPUT OUTPUT INSTRUCTION

One instruction is microcoded to provide all programmed commands to all input/output equipment. This one instruction and all of its features are described as follows.

Name: Programmed Input Output

Mnemonic: PIO

Op Code: 21

Timing: 3.88 to 52.16 µsec for literal addressing. Absolute addressing adds 0.32 µsec. Execution time for the RILS extended mnemonic for all interrupt levels is 9.60 µsec. Attempts to execute a PIO command without the appropriate privilege sets the IOU and takes 2.24 µsec. A PIO command to an unimplemented IO device sets the IOT and takes 16.64 µsec. Execution time for PIO commands other than RILS, which do not set IOU or IOT, is determined by the response time of the addressed device. These commands, when accepted but transfer no data or are rejected, and take from 3.52 to 26.88 µsec. These commands, when accepted and involving a data transfer (ACU = 1), take from 4.80 to 51.84 µsec.

Literal Class: 1

Execution Counter: CC

Registers Affected: A, PC, CPS12 (IOT), CPS13 (IOU), various device registers

Errors Possible: IO response time violation (IOT),
IO usage violation (IOU)

Symbolic Notation: None

Description: The PIO is a standard one-word memory reference instruction. Information required to initiate an IO function is contained in either an immediate operand or the operand referenced by the effective address of the instruction. The format of a PIO operand is:

| COMMAND | | | | | | | | | | | | | DTI | ACU | SKP | DEVICE ADDRESS | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

where:

COMMAND: This 13-bit field specifies the IO function to be performed. If the PIO instruction is in the literal mode, bits 00 through 05 contain the op code ($21_8$), bits 06 through 08 contain the mode ($2_8$), and bits 09 through 12 contain the command.

DTI: Data transfer in. This bit specifies the direction of a data transfer between the processor A register and the device, when the ACU bit is set to indicate that a transfer is to be executed. If the ACU bit is clear, the DTI bit is ignored. If ACU = 1 and DTI = 0, data are transferred out of the A register to the device. If ACU = 1 and DTI = 1, data are transferred into the A register from the device.

ACU: Accumulator usage. This bit determines if a data transfer is to occur between the processor and the addressed device. If ACU = 0, no data transfer occurs. If ACU = 1, a data transfer occurs between the A register and the device in the direction specified by the DTI bit.

SKP: Skip. This bit enables or disables a test of successful completion of a data transfer command (a command associated with a device data register, not a command associated with a device status register). When SKP = 1 in a data-oriented PIO instruction, the Skip signal is sent to the processor by the device if the acknowledge signal occurs and the reject signal does not occur. The Skip signal increments the contents of the program counter so that the instruction immediately following the PIO instruction is skipped (bypassed, omitted, not fetched or executed). The SKP bit allows the program to follow the PIO instruction with an instruction to branch (such as to an error-checking routine) if the PIO instruction is rejected, and to continue normally (skip) if the PIO instruction is not rejected. When SKP = 0, no skipping occurs.

DEVICE ADDRESS: This field specifies the IO device(s) selected to receive the PIO command. Each device decodes (at least) two addresses: one address, which is greater than 23, selects the device for normal PIO communication; a second address, which is less than 24, corresponds to the priority interrupt level of the device and selects it to read the interrupt status bit. (Refer to Table 2-2-6 in Section 2.) Only one device is selected by a PIO instruction with an address greater than 23. Up to 24 devices associated with a specific priority interrupt level are selected by a PIO instruction with an address less than 24. This latter usage allows the interrupt handler at each level to interrogate all appropriate devices with one instruction to determine the device requesting service.

When the PIO instruction is executed, the entire 24-bit operand is transmitted on the PIO bus and is available to the selected device. Most devices examine only bits 11 and 12 of the command field, making it possible to issue the majority of PIO instructions in the literal mode (M = 2; immediate operand). In this mode, the operation code (bits 00 through 05) and mode fields (bits 06, 07, 08) are transmitted on the PIO bus along with the command field (bits 09 through 12). The literal mode can not be used for PIO instructions in which command bits to the left of bit 09 are examined by the device.

Extended Mnemonics of PIO Instruction

The PIO instruction is microcoded to produce a family of commands for each IO device. These extended mnemonics are recognized by the assembler program and are followed by the mnemonic for the device and other data. Device address codes and associated mnemonic codes must be defined in accordance with the rules of the Assembly Language, to allow this portion of PIO instructions to be assembled. Table 2-4-4 lists the basic microcoded commands of the PIO instruction applicable to most IO devices. Table 2-2-6 gives the mnemonic and device address codes of standard IO devices of the FOX 1 system.

Table 2-4-4. PIO Instruction Extended Mnemonics

| COMMAND FIELD (00-12) | DTI FIELD (13) | ACU FIELD (14) | SKP FIELD APPLICABLE (15) | DEVICE ADDRESS FIELD (16-23) | EXTENDED MNEMONIC(S) | COMMAND NAME OF FUNCTION |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | No | Device Address | RST | Read Status |
| 2 | 1 | 1 | Yes | Device Address | RSTC,RSTCSK | Read Status and Clear |
| 1 | 0 | 1 | No | Device Address | WST | Write Status |
| 0 | 0 | 1 | Yes | Device Address | WDA, WDASK | Write Data |
| 0 | 1 | 1 | Yes | Device Address | RDA,RDASK | Read Data |
| 0 | 0 | 0 | Yes | Device Address | CIO,CIOSK | Initiate Channel IO |
| X | X | X | No | Interrupt Level | RILS | Read Interrupt Level Status |

Every IO device contains a device status register accessible to several PIO extended mnemonic commands. The function of bits in the device status register are:

Bit 00: Busy (BSY) bit. If BSY = 1, the IO device is busy (i.e., transmitting data, receiving data, or performing some operation that must run to completion) and is not available for immediate use. Commands received when a device is busy return Acknowledge and Reject signals to the CP. If BSY = 0, the IO device is not busy and is available for immediate use. Commands received when a device is not busy return an Acknowledge signal to the CP.

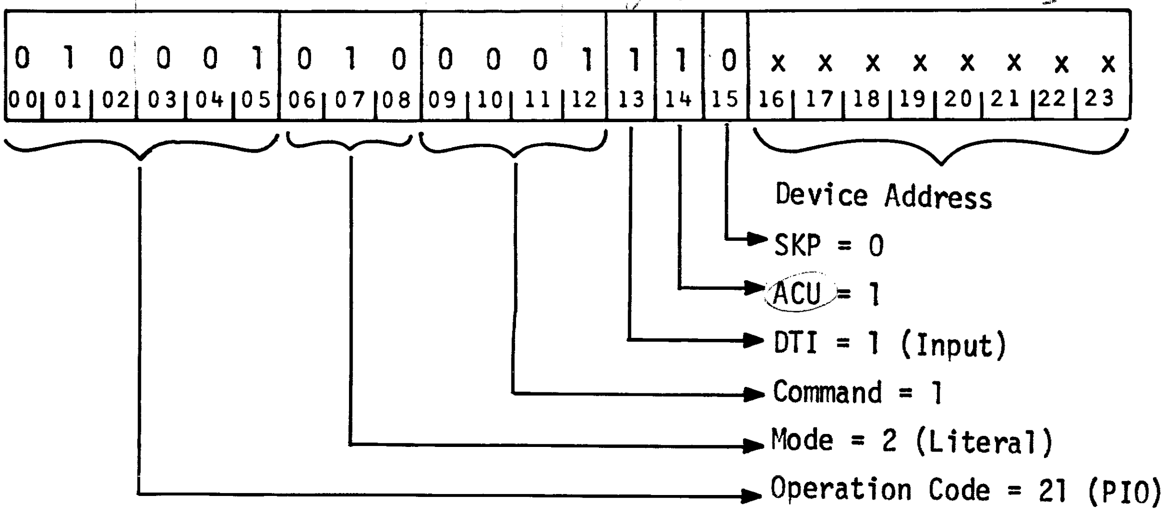
Bit 01: Device interrupt (DVI) bit. This bit is set when the IO device has transmitted or received data, or has completed an operation that required program service from the central processor. When this bit is set (DVI = 1), the device supplies a signal to the IO master. This signal sets a bit in the IO master to request a priority interrupt at the appropriate level for the IO device.

Bits 02 through 23: All bits except 00 and 01 are assigned to status or error conditions appropriate to the specific device.

Both BSY and DVI will not be set at the same time under normal operation for most IO devices.

Details of the extended mnemonics of the PIO instruction are as follows.

Name: Read Status
Mnemonic: RST
Op Code: 21
Timing: 4.80 to 51.84 μsec
Literal Class: 1
Command: 1, DTI: 1, ACU: 1, SKP: 0
Execution Counter: CC
Registers Affected: A, CPS13, device status register
Errors Possible: IO usage violation (IOU)
Symbolic Notation: Device status register → A
Description: The contents of the device status register are transferred into the A register. Every device accepts and executes RST (if selected by the device address) whenever the command is issued (even if the device is busy). The literal format of the instruction is:



Name: Read Status and Clear

Mnemonic: RSTC or RSTCSK if SKP = 1

Op Code: 21

Timing: 4.80 to 51.84 μsec

Literal Class: 1

Command: 2, DTI: 1, ACU: 1, SKP: X

Execution Command: CC

Registers Affected: A, PC, CPS13, device status register

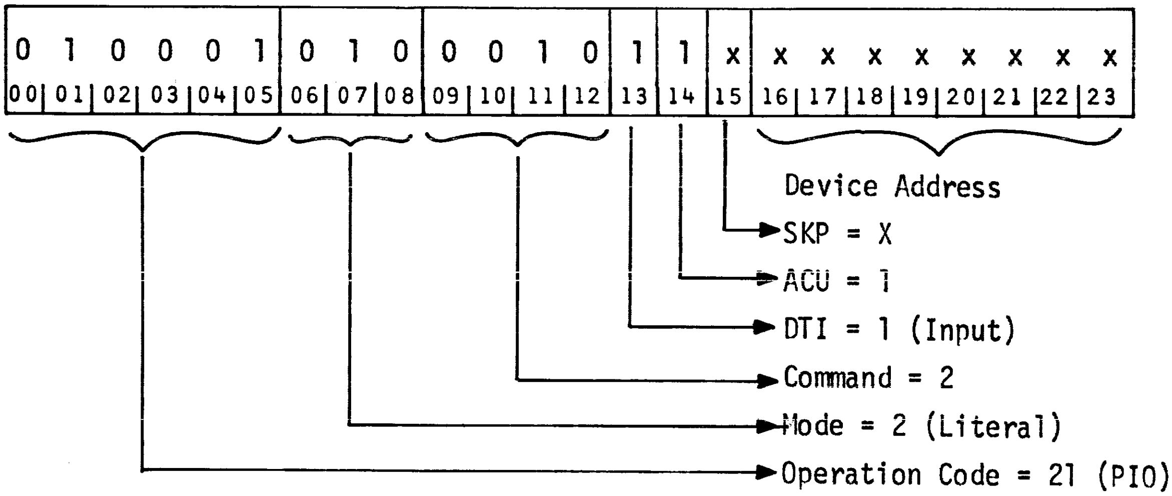
Errors possible: IO usage violation (IOU)

Symbolic Notation: Device status register → A

0 → device status register bit 01 (DVI)

Description: The contents of the device status register are transferred into the A register; then the device interrupt bit (DVI) of the device status register is cleared. In some devices, this command also resets other bits of the device status register; e.g., error or fault bits.

For most devices, RSTC is rejected if the device is busy (BSY = 1). The mnemonic RSTCSK is assembled as an RSTC command in which the SKP bit is set. The literal format of this instruction is:



| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | x | x | x | x | x | x | x | x | x |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Name: Write Status

Mnemonic: WST

Op Code: 21

Timing: 4.80 to 51.84 μsec

Literal Class: 1

Command: 1, DTI: 0, ACU: 1, SKP: 0

Execution Counter: CC

Registers Affected: A, CPS13, device status register

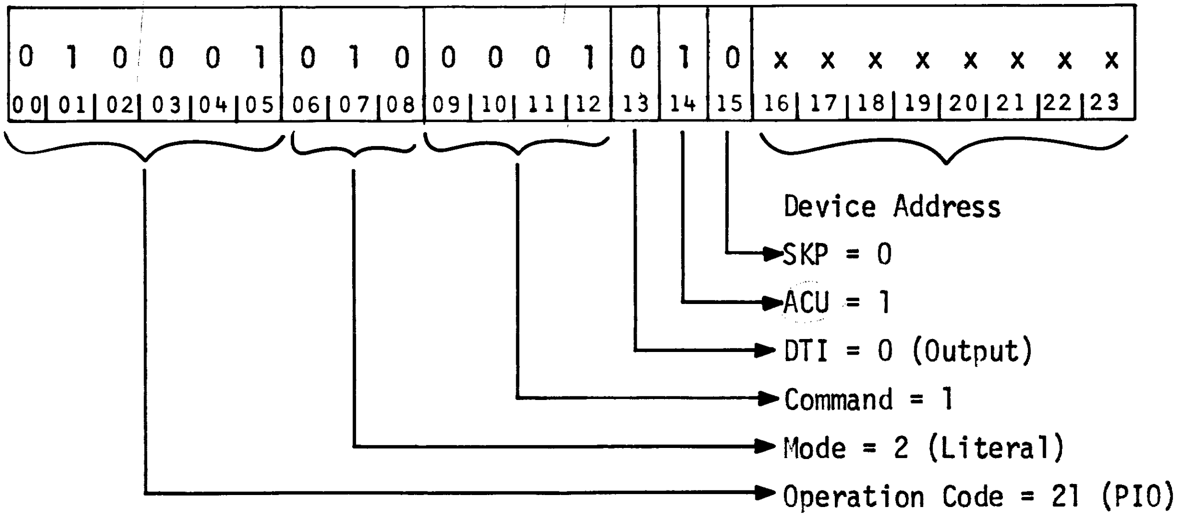
Errors Possible: IO usage violation (IOU)

Symbolic Notation: A → device status register

Description: The contents of the A register are transferred into the device status register. If bit 23 of the A register contains a 1, the device is reset and the status register is set to the conditions established by a master clear function.

WST is intended for use in diagnostic and error routines. It can be executed at any time (is never rejected) and allows the computer to override interrupt requests from a defective device (by clearing the device and/or the DVI bit of the device status register).

The literal format of this instruction is:



| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | x | x | x | x | x | x | x | x |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Name: Write Data

Mnemonic: WDA or WDASK if SKP = 1

Op Code: 21

Timing: 4.80 to 51.84 μsec

Literal Class: 1

Command: 0, DTI: 0, ACU: 1, SKP: X

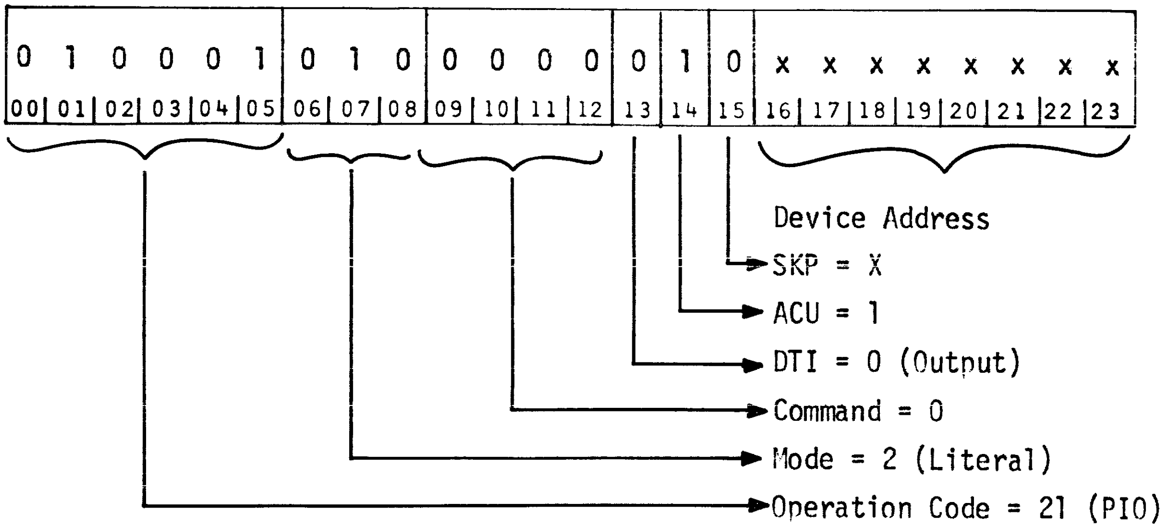
Execution Counter: CC

Registers Affected: A, PC, CPS12, CPS13, device data register

Errors Possible: IO response time violation (IOT),
IO usage violation (IOU)

Symbolic Notation: A → device data register
1 → device status register bit 00 (BSY)
0 → device status register bit 01 (DVI)

Description: The contents of the A register are transferred into the device data register; then the device interrupt bit (DVI) of the device status register is cleared. When WDA or WDASK is acknowledged, the busy bit (BSY) of the device status register is set. The literal format of the instruction is:



| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | x | x | x | x | x | x | x | x |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Name: Read Data

Mnemonic: RDA or RDASK if SKP = 1

Op Code: 21

Timing: 4.80 to 51.84 μsec

Literal Class: 1

Command: 0, DTI: 1, ACU = 1, SKP = X

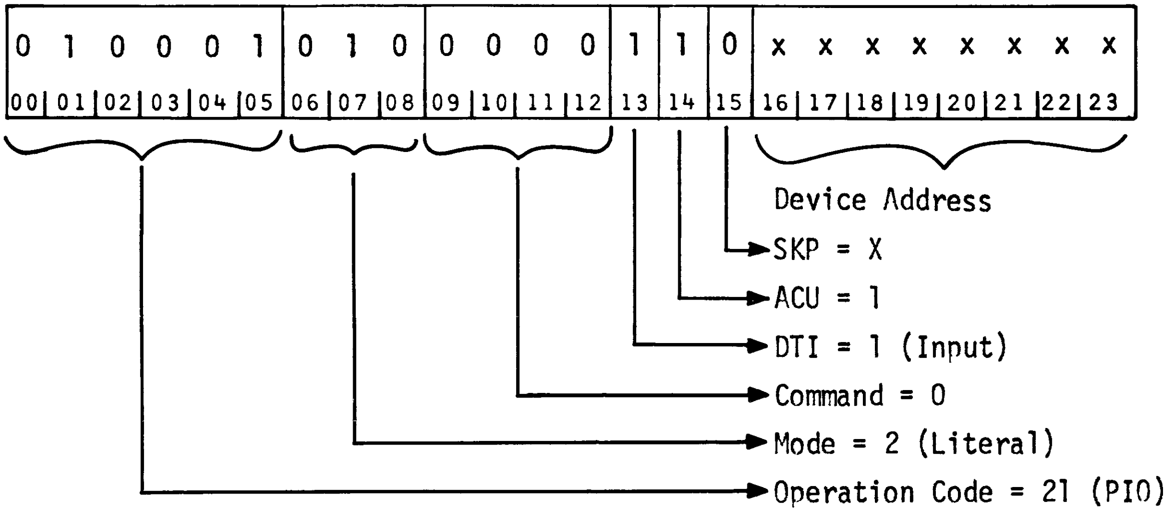
Execution Counter: CC

Registers Affected: A, PC, CPS12, CPS13, device data register

Errors Possible: IO response time violation (IOT),
IO usage violation (IOU)

Symbolic Notation: device data register → A
1 → device status register bit 00 (BSY)
0 → device status register bit 01 (DVI)

Description: The contents of the device data register are transferred into the A register; then the device interrupt bit (DVI) of the device status register is cleared. When RDA or RDASK is acknowledged by the device, the busy bit (BSY) of the device status register is set. The literal format of the instruction is:



| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | x | x | x | x | x | x | x | x |

Name: Initiate Channel IO

Mnemonic: CIO or CIOSK if SKP = 1

Op Code: 21

Timing: 4.80 to 51.84 μsec

Literal Class: 1

Command: 0, DTI: 0, ACU: 0, SKP: X

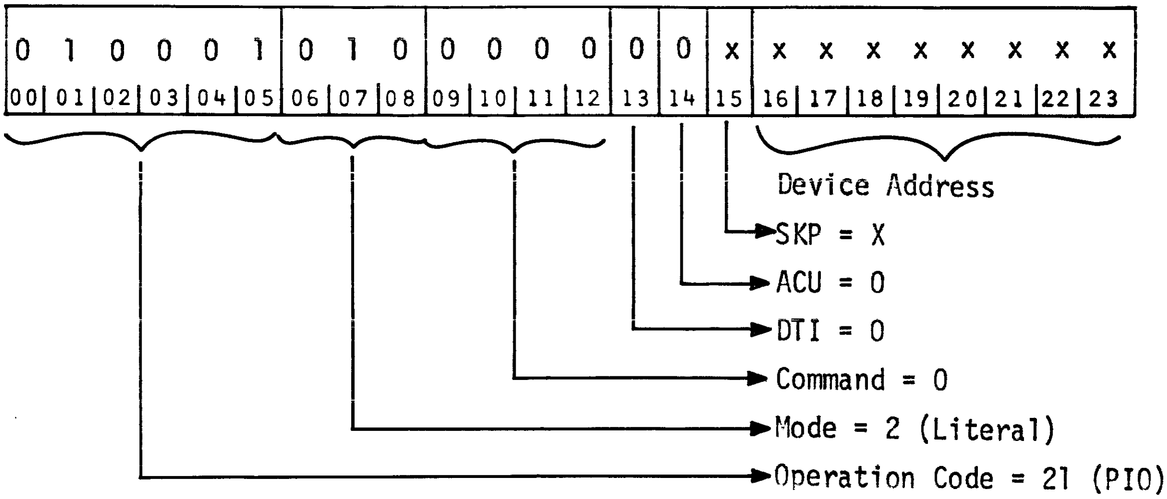
Execution Command: CC

Registers Affected: PC, CPS12, CPS13

Errors Possible: IO response time violation (IOT),
IO usage violation (IOU)

Symbolic Notation: 1 → device status register bit 00 (BSY)
0 → device status register bit 01 (DVI)

Description: Operations to execute a channel IO transfer are initiated in the device by this command. This instruction becomes a No Operation if the device busy bit is set (BSY = 1). If the device is not busy (BSY = 0) when this command occurs, an instruction skip occurs if SKP is set, the busy bit is set, and the device interrupt bit is cleared. The computer program continues and the device begins the operational sequence required to request channel IO service from the IO master. (For more details, refer to the description of Channel Input/Output in Section 2.) The literal format of the instruction is:



| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | x |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Name: Read Interrupt Level Status

Mnemonic: RILS

Op Code: 21

Timing: 9.60 μsec

Literal Class: 1

Command, DTI, ACU, SKP: Ignored

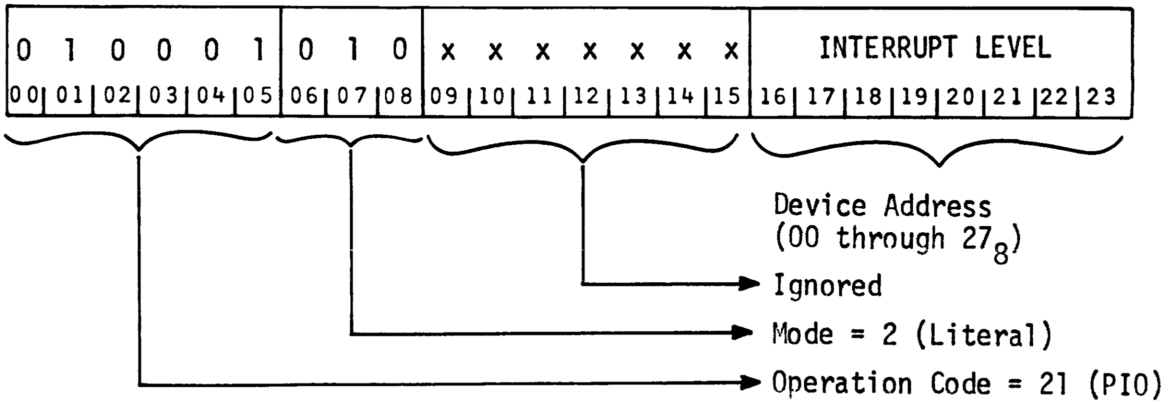
Execution Counter: CC

Registers Affected: A, CPS13, device status register bit 01

Errors Possible: IO usage violation (IOU)

Symbolic Notation: Device status register bit 01 → A

Description: The device interrupt bit (DVI) is read from the device status register into an assigned bit position of the A register for all devices operating at the interrupt level designated by the device address portion of the instruction. Bits 00 through 15 of the operand are ignored by all devices. The device address field is limited to within the range of 00 to $13_8$ in a central processor containing the basic 12 priority interrupt levels, and within the range of 00 to $27_8$ when the CP contains the priority interrupt option, expanding it to 24 levels. Since 24 devices can operate at each priority interrupt level, a 24-bit word can be read into the A register by one RILS command. This command is never rejected (i.e., always acknowledged) and is in the following format:



| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | x | x | x | x | x | x | x | INTERRUPT LEVEL | | | | | | | |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

When a device requires program service, it sets the DVI bit of the status register. Setting of the DVI bit sets a bit in the interrupt matrix of the CP to request an interrupt at the level assigned to the device. Hardware interrupts occur in a priority sequence, and initiate a short subroutine beginning at an address assigned to each level. If more than one device operates on a particular level, the subroutine for that level issues a RILS instruction. By analyzing the word read by the RILS, the subroutine can determine the device that initiated the interrupt. Information on the cause of the interrupt request can be obtained by issuing an RST or RSTC command to the device requesting service. (RSTC clears bit 01 of the device status register to remove the priority interrupt request.) After determining the nature of the request for programmed service, the subroutine performs minimal operations to service the device, sets up appropriate addresses and constants, and then requests complete service from the interrupt handler program of the operating system.

Detailed operation of the RILS command is as follows:

a. Check for IO usage. If no IO privilege is allowed to the program issuing the RILS command, the IOU bit of the central processor status register (CPS13) is set and the command is terminated. When the IOU bit is set, a level 01 interrupt is requested. If any IO privilege is allowed, the command operand is sent to the devices assigned to the specific interrupt level (by way of the COMO signal).

b. After 3.2 μsec, a dummy Acknowledge signal is sent to the CP, and the CP is prepared to receive the interrupt bits.

c. A DTIR signal is sent to the device synchronizers; the DVI bits of all device status registers at the specified level are put on the 24 PIO data lines and, after 3.2 μsec, a signal, indicating that the data are on their way, is sent (a dummy ACK signal).

d. The DVI bits from the devices are transferred into the A register.

e. The instruction terminates with the interrupt status of the (up to) 24 devices in the A register.

The diagram in Figure 2-4-3 shows the relationships between device interrupt bits, the A register, and the interrupt matrix.

## INSTRUCTION SUMMARY

The FOX 1 instruction complement is listed by execution counter in Table 2-4-5 to facilitate study of the commands by means of the mechanization tables (time/function charts of Boolean operations used in the theory of operation manual). Appendix B of this manual also lists the instructions numerically by operation code and alphabetically by mnemonic.

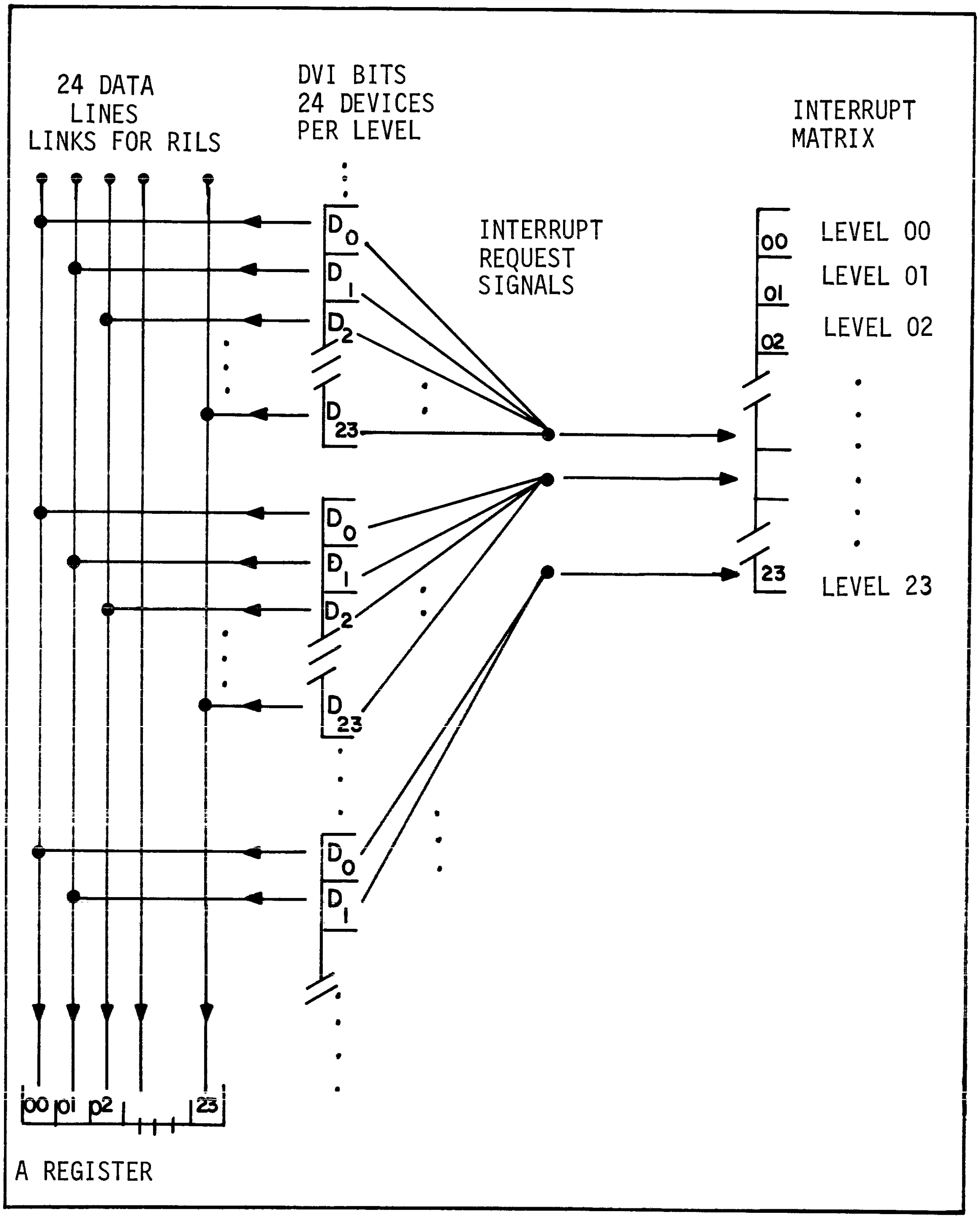


Figure 2-4-3. Interrupt and RILS Action

Table 2-4-5. Instructions Listed By Execution Counter

| COUNTER | MNEM. | OP | MOA | CLASS | PARA. |
|---|---|---|---|---|---|
| CA | RLE | 40 | 1 | 1 | 4.49 |
| ▲ | NMS | 41 | 1 | 2 | 4.77 |
| ▼ | NML | 42 | 1 | 2 | 4.82 |
| CA | SHF | 43 | 1 | 1 | 4.51 |
| CB | ADD | 10 | 1 | 1 | 4.91 |
| ▲ | ADL | 11 | 1 | 1 | 4.94 |
|  | SUB | 12 | 1 | 1 | 4.97 |
|  | SBL | 13 | 1 | 1 | 4.100 |
| ▼ | MPY | 14 | 1 | 1 | 4.108 |
| CB | DIV | 15 | 1 | 1 | 4.103 |
| CC | GEA | 01 | 0 | 1 | 4.219 |
| ▲ | SPL | 17 | 1 | 1 | 4.222 |
|  | PIO | 21 | 1 | 1 | 4.230 |
|  | RFS* | 22 | 0 | S | 4.178 |
|  | BSR | 25 | 0 | S | 4.169 |
|  | BSP | 26 | 0 | S | 4.159 |
|  | STL | 45 | 0 | 2 | 4.21 |
| ▼ | LDL | 51 | 1 | 1 | 4.12 |
| CC | DEM | 56 | 1 | 2 | 4.215 |
| CD | HLT | 00 | 0 | 1 | 4.224 |
| ▲ | AND | 02 | 1 | 1 | 4.29 |
|  | IOR | 03 | 1 | 1 | 4.31 |
|  | LLC | 04 | 1 | 1 | 4.15 |
|  | XOR | 05 | 1 | 1 | 4.33 |
|  | BRU* | 22 | 0 | S | 4.156 |
|  | NOP* | 22 | 0 | X | 4.225 |
|  | BRN | 23 | 0 | 2 | 4.154 |
|  | BRZ | 24 | 0 | 2 | 4.155 |
|  | STA | 44 | 0 | 2 | 4.19 |
|  | STE | 46 | 0 | 2 | 4.20 |
|  | LDE | 52 | 1 | 1 | 4.11 |
|  | LDA | 53 | 1 | 1 | 4.10 |
| ▼ | EAM | 54 | 1 | 2 | 4.24 |
| CD | MST | 55 | 1 | 2 | 4.22 |
| CE | BYT | 06 | 1 | 2 | 4.207 |
| ▲ | {BIT | 07 | 1 | 1 | 4.197 |
|  | BIT | 07 | 1 | 2 |  |
| ▼ | {CWM | 20 | 1 | 1 | 4.192 |
| CE | CWM | 20 | 0 | 2 |  |
| CF | FAS | 30 | 1 | 1 | 4.114 |
| ▲ | FSS | 31 | 1 | 1 | 4.116 |
|  | FMS | 32 | 1 | 1 | 4.118 |
|  | FDS | 33 | 1 | 1 | 4.120 |
|  | FAL | 34 | 1 | S | 4.115 |
|  | FSL | 35 | 1 | S | 4.117 |
|  | FML | 36 | 1 | S | 4.119 |
| ▼ | FDL | 37 | 1 | S | 4.121 |
| CF | SNR | 47 | 0 | 2 | 4.122 |
| CG | PI** | - | - | - | 4.183 |
| ▲ | RFI | 16 | 1 | 2 | 4.185 |
| ▼ | {MOV | 50 | 1 | 2 | 4.216 |
| CG | MOV | 50 | 0 | 2 |  |
| CH | LXA | 60 | 1 | 1 | 4.132 |
| ▲ | LXB | 61 | 1 | 1 | 4.133 |
|  | SXA | 62 | 0 | 2 | 4.134 |
|  | SXB | 63 | 0 | 2 | 4.135 |
|  | AXA | 64 | 1 | 1 | 4.136 |
|  | AXB | 65 | 1 | 1 | 4.137 |
|  | CXA | 66 | 1 | 1 | 4.138 |
|  | CXB | 67 | 1 | 1 | 4.139 |
|  | TIA | 70 | 1 | 1 | 4.140 |
|  | TIB | 71 | 1 | 1 | 4.142 |
| ▼ | BDA | 72 | 0 | 2 | 4.145 |
| CH | BDB | 73 | 0 | 2 | 4.148 |

* Op code 22 literal mode = RFS. Op code 22 not literal mode = BRU. Op code 22 using PC-relative mode and a displacement of 00001 = NOP.

** Program interrupt is not an instruction, but is executed like one by the CG counter.

# 5/INTERRUPT STRUCTURE

Operations executed by the FOX 1 system are oriented to a priority interrupt system. The term "priority interrupt" implies that the central processor interrupts current activity to service a higher priority request for system resources. Execution of all programs is initiated, either directly or indirectly, by a priority interrupt.

Because interrupting is an orderly process of closing down execution of one program and initiating execution of another, the interrupted program may be resumed at a later time without loss of information or control. Interrupt activity (i.e., interrupt handling) is a combined function of hardware provisions and programming techniques.

The interrupt structure enables the central processor to establish and control up to 24 interrupt levels, numbered 00 through 23. Twelve levels are standard; the additional 12 are optional. The priority of an interrupt is determined by its level. Level 00 interrupts have the highest priority, with the degree of priority decreasing as the level number increases. Level 23 interrupts have the lowest priority.

The central processor receives interrupt signals from numerous sources throughout the FOX 1 system. These signals are collected in a hardware interrupt matrix (IM). The IM can be visualized as a 24-by-24 matrix in which each horizontal row corresponds to a priority level, and each vertical column represents one interrupt source at each level. Each level can accept request signals from 24 sources; therefore, the basic interrupt structure can accept 288 (12 × 24 = 228) interrupts, and the optionally expanded interrupt structure can accept 576 (24 × 24 = 576) interrupts.

Interrupt sources are classified as:

a. IO when they originate in input/output or peripheral equipment; e.g., drum, card punch, analog input, and so forth.

b. Hardware when they occur as bits in the CPS register. These interrupts are caused by alarm conditions in the CP; e.g., power failure, arithmetic overflow, and so forth.

c. Software when they occur as bits in the SIS register.

Interrupt sources for each level are usually, although not necessarily, limited to one classification. This simplifies programmed determination of the cause of an interrupt.

The PIO instruction (used to service IO interrupts) contains an 8-bit device address field; therefore, it can select 256 devices. However, device addresses 000 through 023 are reserved for the RILS instruction used by the interrupt handler program for each level. This means that 232 (256 - 24 = 232) addressable devices can be included in the system and connected to the IM.

After each new instruction is fetched from memory, but before address modification begins and before the program count is changed, the CP scans the interrupt matrix to determine if an interrupt request is pending (active) at a priority level higher than that of the program in progress. If not, address modification begins and the program continues uninterrupted. However, if a higher priority interrupt request is pending, the current instruction is aborted, parameters of the current program are stored (including the PC so the current instruction can be fetched again later), and program control branches to the interrupt handler program for the pending level.

## PROGRAM PRIORITIES

Every program has a priority level. This level is specified by the IS field of the privilege and level register (PL). A program's priority level prevents interrupt at levels with priorities equal to or less than the current level specified by the IS. If one or more interrupt levels of higher priority than the level specified by IS becomes active (i.e., a level of lower numeric value than IS), interrupt occurs. If more than one level is active, the one of highest priority (lowest numerical value) is served.

For example, if the current program is operating at level 08 (IS = 08), then interrupt requests at levels 08 through 23 will not interrupt the current program; interrupt requests at levels 00 through 07 will cause interrupt. Thus, a level 08 program may be interrupted by a level 07 interrupt signal, which activates a level 07 program; the level 07 program may be interrupted in turn by a level 06 interrupt signal, which activates a level 06 program, and so forth. The interrupted programs remain in an active but suspended state until all interrupts at higher levels have been serviced and the interrupt requests are turned off. Termination of an interrupt handler at one level restarts the interrupted program at the next lower priority active level at the point where it was interrupted. In this manner, several programs may be in a suspended (interrupted) state at any given time.

Although a program has an established operating priority level, it is possible, during execution of the program, to change that level within a certain range. Both the established level and the level range are specified in bits 09 through 23 of the PL register:

| UL | | | | | LL | | | | | IS | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Where:

UL = upper level of the range (i.e., lowest priority)
LL = lower level of the range (i.e., highest priority)
IS = current priority

The Set Privilege and Level Register instruction (SPL, operation code 17) may be used to change IS. Any new setting of IS must satisfy the relationship

$$LL \leq \text{New IS} \leq UL.$$

For example, assume bits 09 through 23 of the PL register for a program contain:

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

$UL = 15_{10}$ $LL = 5_{10}$ $IS = 8_{10}$

Then the operating level may be changed from 08 to any value from 05 through 15. Thus, if it is determined that this program operating at level 08 is too important to be interrupted, the chances of interruption can be reduced by resetting the IS to 05. This operation inhibits interrupts below level 04. In the same manner, the program in this example can be reduced in priority to level 15, which allows the program to be interrupted by level 14 and higher instead of level 07 and higher.

When the priority is lowered, a possibility exists for destroying the return to the interrupted program. For this reason, the initial IS will normally be equal to the UL. Lowering of the priority level should be done only with great caution and complete understanding of the interrupt process.

In the previous example, if the priority is lowered from level 08 to level 15, another interrupt at level 08 before completion of the current program of level 15 can make it impossible to return to the program in progress when the first level 08 interrupt occurred. In addition, if the level 08 interrupt request is still set when the program priority is lowered, the level 08 program will self-interrupt, destroying the return to the interrupted program.

## INTERRUPT DEDICATED MEMORY

Memory locations 00000 through 00027 are dedicated to the interrupt structure. One memory location is assigned to each interrupt level: location 00000 to level 00, location 00001 to level 01,..., location 00027 to level 23.

The words in these locations contain an interrupt vector in bits 09 through 23 (bits 00 through 08 are not used). The interrupt vector is an address that points to the first word of a 16-word storage area reserved for each interrupt level. In operations under control of the real time executive, specific interrupt vectors are asigned, and the 16-word area is reserved at the time interrupt handlers are loaded into memory.

| NOT USED | | | | | | | | | INTERRUPT VECTOR | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

An interrupt vector and its associated 16-word area accommodate the interrupt hardware action. The first ten words of this area are used as storage for the contents of the ten addressable registers that define the interrupted program's assigned operating environment (memory fence, privilege and level, and index common pointer) and its current operating conditions (program counter, index top-of-stack pointer, indexes A and B, accumulator and arithmetic extender, and central processor indicator). The last six words of the reserved area contain information that defines the interrupt handler operating environment (memory fence, privilege and level, and index common pointer) and initial operating conditions (program counter, index top-of-stack pointer, and index B).

## INTERRUPT HARDWARE ACTION

On detection of an interrupt request at a higher priority level than the current program level, the CP hardware initiates the hard-wired Program Interrupt instruction to execute a series of actions:

a. The interrupt vector (v) is obtained from the location assigned to the interrupting level. This vector points to the first of 16 words reserved for this particular interrupt level.

b. Then the contents of ten addressable registers (addresses 77766 through 77777) are stored into the memory locations specified by the interrupt vector, v, and v + 1, v + 2..., v + 9. These registers contain information that will be needed for subsequent resumption of the interrupted program. Register contents are stored in the order of their addresses:

| Register | Register Address | Operation |
|---|---|---|
| Memory Fence | 77766 | FNC → v |
| Privilege and Level | 77767 | PL → v + 1 |
| Program Counter | 77770 | PC → v + 2 |
| Index Common | 77771 | XC → v + 3 |
| Index Top-of-Stack | 77772 | XT → v + 4 |
| Index B | 77773 | XB → v + 5 |
| Index A | 77774 | XA → v + 6 |
| Accumulator | 77775 | A → v + 7 |
| Arithmetic Extender | 77776 | E → v + 8 |
| Central Processor Indicator | 77777 | CPI → v + 9 |

c. After the ten registers are stored, the contents of the next six memory locations (v + 10 through v + 15) are loaded into the registers with addresses 77766 through 77773. These six words provide operating environment assignments and initial operating conditions for the interrupt handler. Registers with addresses 77760 through 77765 and 77774 through 77777 are not changed. This information is loaded in the following order:

| Operation | Register Address | Register |
|---|---|---|
| v + 10 → FNC | 77766 | Memory Fence |
| v + 11 → PL | 77767 | Privilege and Level |
| v + 12 → PC | 77770 | Program Counter |
| v + 13 → XC | 77771 | Index Common |
| v + 14 → XT | 77772 | Index Top-of-Stack |
| v + 15 → XB | 77773 | Index B |

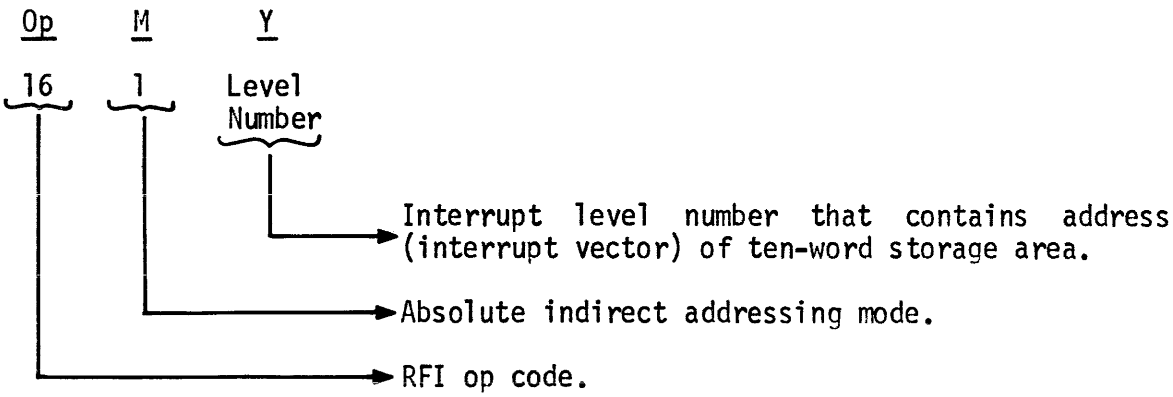
Since both the privilege and level and program counter registers are among the registers given new values by this action, the correct priority (IS field of PL) and the location of the first instruction (PC) of the interrupt handler program are obtained automatically.

Following execution of the Program Interrupt instruction (the three preceding actions), the interrupt handler begins operating unless an interrupt request of higher priority arrived in the interrupt matrix during these actions. If a higher priority interrupt request is pending, another Program Interrupt instruction is executed to perform the store and load actions again at the new level.

Figure 2-5-1 is an example of interrupt action resulting from a level 15 interrupt. In this example, the interrupted program (program A) is operating at level 16 (bits 19 through 23 of the PL register are 10000), and the instruction in location 04576 (PC = 04576) is about to be executed. Following interrupt, level 15 interrupt handler (program B) begins execution at location 03500 (new PC = 03500) on priority level 15 (bits 19 through 23 of the new PL are 01111). The level 15 vector supplies the address of the first word of the 16-word store-and-load area.

## RETURN FROM INTERRUPT

When an interrupt handler has completed servicing an interrupt, control is returned to the interrupted program by use of a Return From Interrupt instruction (RFI, operation code 16). The RFI instruction reloads the ten addressable registers (addresses 77766 through 77777) with the information that was stored by interrupt hardware action. The effective address of RFI for this purpose is the address of the memory location containing the first of the ten words to be loaded. Therefore, the following instruction is the last instruction executed in an interrupt servicing program:


Op
M
Y
16
1
Level Number
Interrupt level number that contains address (interrupt vector) of ten-word storage area.
Absolute indirect addressing mode.
RFI op code.

```
                                 LOC. OF           VECTOR
                                 VECTOR
   INTERRUPT AT LEVEL 15          00017:           0003455

                                   INTERRUPT STORE AND LOAD AREA

   PROGRAM A REGISTERS
   AT TIME OF INTERRUPT          LOCATION CONTENTS BEFORE INTERRUPT

77760: )                         03455:   XXXXXXXX )
77761: |                         03456:   XXXXXXXX |
77762: |   NOT AFFECTED          03457:   XXXXXXXX |
77763: >   BY INTERRUPT          03460:   XXXXXXXX |
77764: |                         03461:   XXXXXXXX > X = ANY VALUE
77765: )                         03462:   XXXXXXXX |
77766:     04770400(FNC)         03463:   XXXXXXXX |
77767:     51040260(PL,IS=16)    03464:   XXXXXXXX |
77770:     00004576(PC)          03465:   XXXXXXXX |
77771:     00004700(XC)          03466:   XXXXXXXX )
77772:     00004727(XT)          03467:   03770350 )
77773:     00077773(XB)          03470:   51042217 | INITIAL REGISTER
77774:     00000004(XA)          03471:   00003500 | VALUES FOR
77775:     03612407(A)           03472:   00003700 > PROGRAM B
77776:     00000021(E)           03473:   00003720 |
77777:     20000000(CPI)         03474:   00000000 )

                    FOLLOWING INTERRUPT ACTION:

   PROGRAM B REGISTERS

77760: )                         03455:   04770400 )
77761: |                         03456:   51040260 |
77762: |                         03457:   00004576 |
77763: >   UNCHANGED             03460:   00004700 | SAVED PROGRAM A
77764: |                         03461:   00004727 > REGISTERS
77765: )                         03462:   00077773 |
77766:     03770350(FNC)         03463:   00000004 |
77767:     51042217(PL,IS=15)    03464:   03612407 |
77770:     00003500(PC)          03465:   00000021 |
77771:     00003700(XC)          03466:   20000000 )
77772:     00003720(XT)          03467:   03770350 )
77773:     00000000(XB)          03470:   51036357 |
77774: )   UNCHANGED             03471:   00003500 > UNCHANGED
77775: >                         03472:   00003700 |
77776: |                         03473:   00003720 |
77777: )                         03474:   00000000 )
```

Figure 2-5-1. Interrupt Action Example

Return from a level 08 interrupt, then, is accomplished by:

| Op | M | Y |
|---|---|---|
| 16 | 1 | 00010 |

If the level 08 interrupt vector is 005671, the effective address of 16 1 00010 is 05671.

A symbolic representation of the results of RFI follows:

| Operation | Register Address | Register |
|---|---|---|
| v → FNC | 77766 | Memory Fence |
| v + 1 → PL | 77767 | Privilege and Level |
| v + 2 → PC | 77770 | Program Counter |
| v + 3 → XC | 77771 | Index Common |
| v + 4 → XT | 77772 | Index Top-of-Stack |
| v + 5 → XB | 77773 | Index B |
| v + 6 → XA | 77774 | Index A |
| v + 7 → A | 77775 | Accumulator |
| v + 8 → E | 77776 | Arithmetic Extender |
| v + 9 → CPI | 77777 | Central Processor Indicator |

Following RFI, with the registers restored, an interrupted program restarts operation at the point of interruption (PC is reset by RFI to the pre-interrupt value) with all operating conditions as they were prior to the interrupt.

RFI is a privileged instruction; that is, its execution is dependent upon the privileges allowed to the program in which the instruction occurs. The privilege and level register (PL) specifies a program's privileges. If both the EMB and RMP bits of PL are clear (bit 00 = 0 and bit 01 = 0), RFI cannot be executed. An attempt to execute RFI without privilege is a register privilege violation that causes a level 01 interrupt, and it would be a responsibility of the level 01 interrupt handler to react to the execution of an unprivileged RFI instruction.

## INTERRUPT CLASSES

Each priority level accommodates three interrupt classes:

a. Software Interrupts
b. IO Interrupts
c. Hardware Interrupts

Any one class, or the three classes simultaneously, may generate an interrupt at a single level. The interrupt request signals at each level are logically ORed into the 24-bit interrupt matrix; i.e., signals from one or more of the three interrupt classes at a given level set the same level bit in the interrupt matrix.

For each interrupt class, there may be one or more interrupt sources. For example, in the IO interrupt class, up to 24 IO devices may be assigned to a specific level. Also, any program that has software interrupt privilege can be the source of a software class interrupt. Hardware class interrupts are basically on two or three levels only but may have several interrupt sources at each level.

In a typical situation, levels are assigned so only one interrupt class is associated with a given level. Table 2-5-1 shows such a level assignment.

When several sources of more than one class generate interrupts at the same level, it is the responsibility of the interrupt handler to interrogate the system to determine the source of the interrupt and initiate the proper program to handle the interrupt.

### Software Interrupts

Software interrupts are generated by setting a bit in the software interrupt status register (SIS, address 77764). SIS is a 24-bit register

Table 2-5-1. Interrupt Priority Assignment

| PRIORITY LEVEL | CLASS | INTERRUPT SERVICING PROGRAM |
|---|---|---|
| 0 | Hardware | Security Programs (Power Failure, Memory Parity, Halt Instruction, etc.) |
| 1 | Hardware | Error Handlers (IOT, IOU, FNV, RPR, SIV, ILS)** |
| 2 | IO | Drum and Disk Handlers |
| 3 | IO | Clock Handlers and High Priority Process IO |
| 4 | Software | Real Time Executive |
| 5 | Software | Process Control Consoles Handlers |
| 6 | IO | Peripherals Handlers (PTR,PTP, TTY, CDR, CDP, PRINT)*** |
| 7 | Software | IO Drivers |
| • | | |
| • | | |
| • | | |
| • | | |
| • | | |
| • | | |
| N*-1 | Hardware | Address Stop Module Servicing Routine |
| N* | Software | Background Monitor and Background Programs |

*N = Number of interrupt levels implemented
**IOT = IO timer, no response from device
IOU = IO usage violation
FNV = Memory protection fence violation
RPV = Register protection violation
SIV = Software interrupt protection violation
ILS = Illegal instruction
***PTR = Paper Tape Reader
PTP = Paper Tape Punch
TTY = Teletypewriter
CDR = Card Reader
CDP = Card Punch
PRINT= Line Printer

in which bit positions correspond to priority interrupt levels. Settings in SIS are transmitted to the interrupt matrix.

A software interrupt can be generated by any program that has software interrupt privilege or register manipulation privilege. This privilege is specified in the privilege and level register by either bit 00, bit 01, or bit 02 being set. Any attempt to alter SIS without privilege causes a level 01 interrupt.

A program with software interrupt privilege that sets a bit in SIS at a level higher than its own priority level is immediately interrupted; if it sets an SIS bit at a level equal to or less than its own priority, interrupt is deferred until a program of a lower level is operating.

For example, a program operating at level 05 can cause an interrupt by setting bit 04 of SIS. Also, a program at level 05 may set bit 06 of SIS, and a level 06 interrupt would occur only after all interrupts of higher priorities have been serviced and the IS field of the PL register is greater than 06.

The Bit Manipulation instruction (BIT, operation code 07) is normally used to set a bit in SIS. BIT can be used for this purpose only if the EMB or SIP bit of the privilege and level register is set. BIT is a two-word instruction (refer to Section 4) that occupies two contiguous memory locations.

Example:

The following BIT instruction sets bit 05 of SIS and, assuming the current program is operating at level 06 or lower, interrupt occurs immediately:

```
Location
  (PC)     Op   M     Y

   .
   .
 03501     07   2   00005   Two-word BIT instruction
 03502     03   0   77764
       ◄────────────────────Interrupt here
 03503     .    .     .
           .    .     .
           .    .     .
           .    .     .
```

where:

Word 1 07 - is the BIT operation code
2 - specifies literal mode
00005 - is the immediate operand that specifies the bit to be manipulated

Word 2 03 - is a secondary operation code that specifies set bit and continue
0 - specifies absolute direct mode
77764 - is the address of SIS (i.e., the location of the word to be modified)

Interrupt occurs in this case before execution of the instruction in location 03503.

The instructions in the following example also set SIS bits if the operating program has register manipulation privilege (EMB or RMP set in the PL register.)

Example:

Assume a program operating on level 08. The two instructions in locations 04632 and 04633 set SIS bits 05, 06, and 07.

### Instructions

| Location (PC) | Op | M | EA | Interpretation |
|---|---|---|---|---|
| • | • | • | • | |
| • | • | • | • | |
| 04632 | 51 | 4 | 00036 | Load Long (LDL) instruction, using PC relative addressing. A = 01600000, E = 01600000. |
| 04633 | 55 | 0 | 77764 | Masked Store (MST) instruction into SIS Interrupt |
| • | | | | |
| • | | | | |
| • | | | | |

### Constants

| | | |
|---|---|---|
| 04670 | 01600000 | 1 bit in positions 05, 06, 07 |
| 04671 | 01600000 | Mask for Masked Store |
| • | | |
| • | | |
| • | | |

A level 05 interrupt occurs immediately after executing the instruction in location 04633. Interrupts at levels 06 and 07 will not occur until the operating priority level is returned to 08. If each of the interrupt handlers has register manipulation privilege and terminates with an RFI instruction, the following sequence of events occurs:

1. Level 05 interrupt.

2. Interrupt hardware action stores registers of level 08 program and loads registers for level 05 program.

3. Interrupt handler is executed and resets bit 05 of SIS.

4. RFI reloads level 08 program registers.

5. Immediate level 06 interrupts before resumption of level 08 program.

6. Interrupt hardware action stores registers of level 08 program and loads registers for level 06 program.

7. Interrupt handler is executed and resets bit 06 of SIS.

8. RFI reloads level 08 program registers.

9. Immediate level 07 interrupt before resumption of level 08 program.

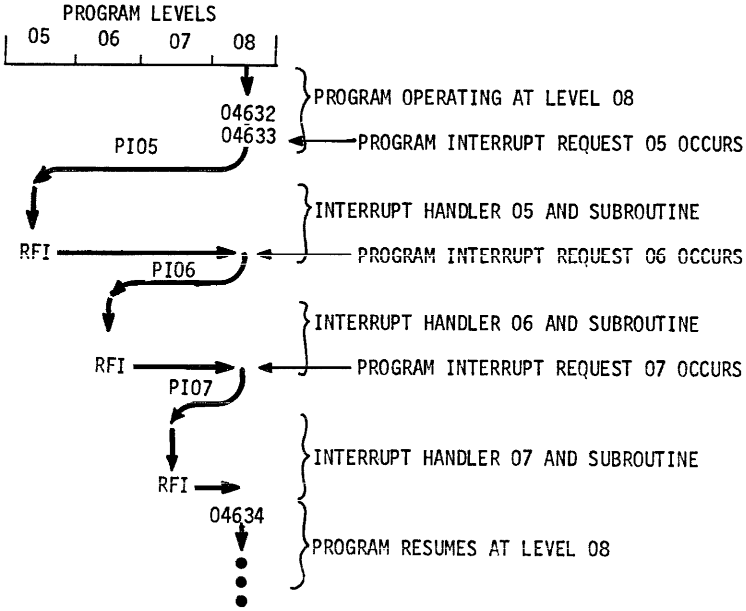
10. Interrupt hardware action stores registers of level 08 program and loads registers for level 07 program.

11. Interrupt handler is executed and resets bit 07 of SIS.

12. RFI reloads level 08 program registers.

13. Level 08 program resumes operation with PC = 04634.

These events are illustrated in the following diagram (vertical lines indicate program execution):


PROGRAM LEVELS
05
06
07
08
04632
04633
PROGRAM OPERATING AT LEVEL 08
PI05
PROGRAM INTERRUPT REQUEST 05 OCCURS
INTERRUPT HANDLER 05 AND SUBROUTINE
RFI
PI06
PROGRAM INTERRUPT REQUEST 06 OCCURS
INTERRUPT HANDLER 06 AND SUBROUTINE
RFI
PI07
PROGRAM INTERRUPT REQUEST 07 OCCURS
INTERRUPT HANDLER 07 AND SUBROUTINE
RFI
04634
PROGRAM RESUMES AT LEVEL 08

When more than one interrupt class is accommodated at a given level, and software interrupt is one of those classes, it is necessary for the interrupt handler at that level to be able to distinguish between classes. It is also necessary that the interrupt request signal at that level be reset to prevent re-interrupt at the same level following termination of the interrupt handler.

For example, if both software and IO interrupts are accommodated at level 09, the instructions to determine which class caused a level 09 interrupt may be:

| Location | Op | M | EA | Interpretation |
|---|---|---|---|---|
| α | 07 | 2 | 00011 | Bit manipulation instruction that checks bit 09 of SIS. If bit 09 is set, the next instruction resets it and skips the next instruction. If bit 09 is not set, the next instruction in sequence is executed. |
| α + 1 | 04 | 0 | 77764 | |
| α + 2 | Branch to IO interrupt servicing program. | | | |
| α + 3<br>.<br>.<br>.<br>.<br>.<br>. | Software interrupt servicing program. | | | |

IO Interrupts

Input/output interrupts are generated by individual IO devices. Up to 24 devices may be assigned to a single interrupt level.

When a device requests interrupt, that device sets an interrupt bit in its device status word.

Bit 00 of the device status word is the device busy bit (BSY). BSY is set while the device is executing a PIO command and indicates a busy condition. BSY is reset when the device is not busy.

Bit 01 of the device status word is the device interrupt indicator (DVI). This bit is set when the device is requesting interrupt; otherwise, DVI is reset.

BSY and DVI are normally not set simultaneously. A command that causes a device to read or write data resets DVI and sets BSY when the data transfer begins. Also, the CP can reset DVI with a Read Status and Clear command.

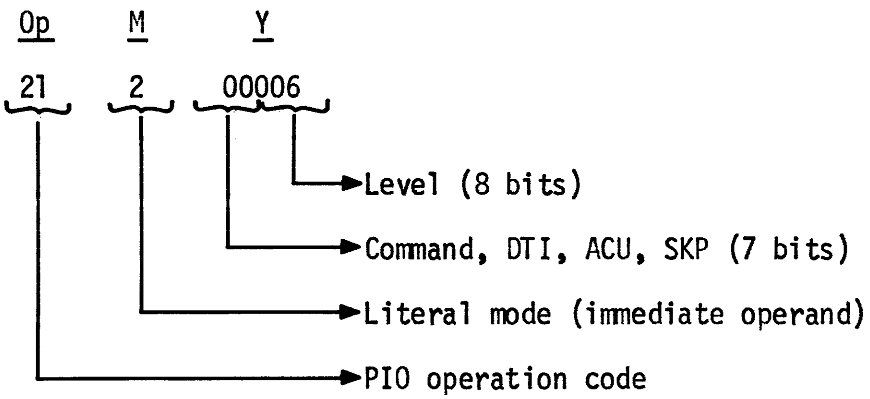
Bits 02 through 23 of the device status word have specific uses according to the device.

If any bit in the device status word is set, that device is said to "have status".

When the DVI bit of a device status word is set, an interrupt request signal is sent to the interrupt matrix on the level assigned to that device. Because it is possible for any one of 24 devices to generate an IO interrupt on a single level, one of the functions of the interrupt handler at that level is to determine the specific source of the interrupt.

To determine if the interrupt at a particular level was caused by an IO device, an interrupt handler issues a Read Interrupt Level Status (RILS) command and then analyzes the status word read into the A register to determine the device that has status. One RILS command reads the content of the DVI bit from the status register of all devices at that interrupt level. This information is read into the A register as a 24-bit status word that can be tested bit-by-bit to determine the device (or devices) having a set DVI bit. Then the interrupt handler issues a Read Status (RST) command to the device that has status and then analyzes this device status word read into the A register to determine the cause of the interrupt request (error, IO complete, and so forth). Both the RILS and RST commands are microcoded (extended mnemonics of the) Programmed Input Output (PIO) instructions, with an operation code of $21_8$. Refer to Section 4 for the format and detailed operation of the RILS and RST commands.

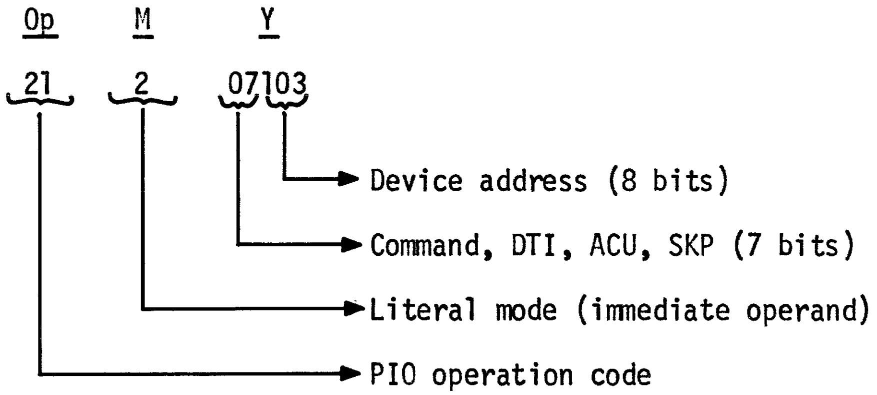
Since the significant bits of the PIO operand are in bit positions 12 through 23, the RILS and RST instructions can use an immediate operand (literal mode). For example, the following RILS instruction reads the status of all devices at level 06:


Op
M
Y
21
2
00006
Level (8 bits)
Command, DTI, ACU, SKP (7 bits)
Literal mode (immediate operand)
PIO operation code

The result of this instruction is that the DVI bit of each device status word at level 06 is transmitted along a PIO data bus line to the A register. Each device is assigned a line on the PIO data bus for this purpose. The bit position is not the same as the device address.

By scanning the A register for 1's, the device (or devices) that have status can be determined by the interrupt handler. A Read Status command, directed to a device that caused a 1 in the A register, results in that device's status word being read into the A register. A test on bit 01 of the A register will then indicate if that device requests interrupt.

For example, if bit 03 of the A register is set during a RILS command, and the device that transmits its status to that bit has the device address 103, the device's status word is read by the RST instruction:


Op
M
Y
21
2
07103
Device address (8 bits)
Command, DTI, ACU, SKP (7 bits)
Literal mode (immediate operand)
PIO operation code

This instruction causes the device status word of device 103 to be transmitted into the A register. Then the status bits may be tested to determine whether a completed operation (DVI set, usually) or an error condition was the cause of that device interrupt.

## Hardware Interrupts

Standard hardware interrupts are assigned to levels 00 and 01. If the address stop capability is implemented, address stop interrupt is assigned to level N - 1, where N is the number of levels (N = 12 or 24). Hardware interrupts are generated by an abnormal or fault condition in the system.

Hardware interrupt request signals are recorded as settings in the central processor status register (CPS) and are transmitted from the CPS to the interrupt matrix. The standard hardware interrupts are listed in Table 2-5-2.

Table 2-5-2. Standard Hardware Interrupts

| LEVEL | CPS BIT | CAUSE OF INTERRUPT |
|---|---|---|
| 00 | 21 | Main power failure |
| 00 | 22 | Memory parity error |
| 00 | 23 | Execution of a halt instruction |
| 01 | 12 | IO timer; failure of an IO device to respond within a prescribed unit |
| 01 | 13 | Attempt to use IO without privilege; privileges specified by bits 03, 04, 05 of PL register |
| 01 | 14 | Memory fence violation |
| 01 | 15 | Register privilege violation |
| 01 | 16 | Software privilege violation |
| 01 | 17 | Illegal instruction or attempt to access unimplemented memory |
| 01 | 18 | Fixed point arithmetic overflow |
| 01 | 19 | Floating point characteristics overflow |
| N-1 | 0 | Address stop |

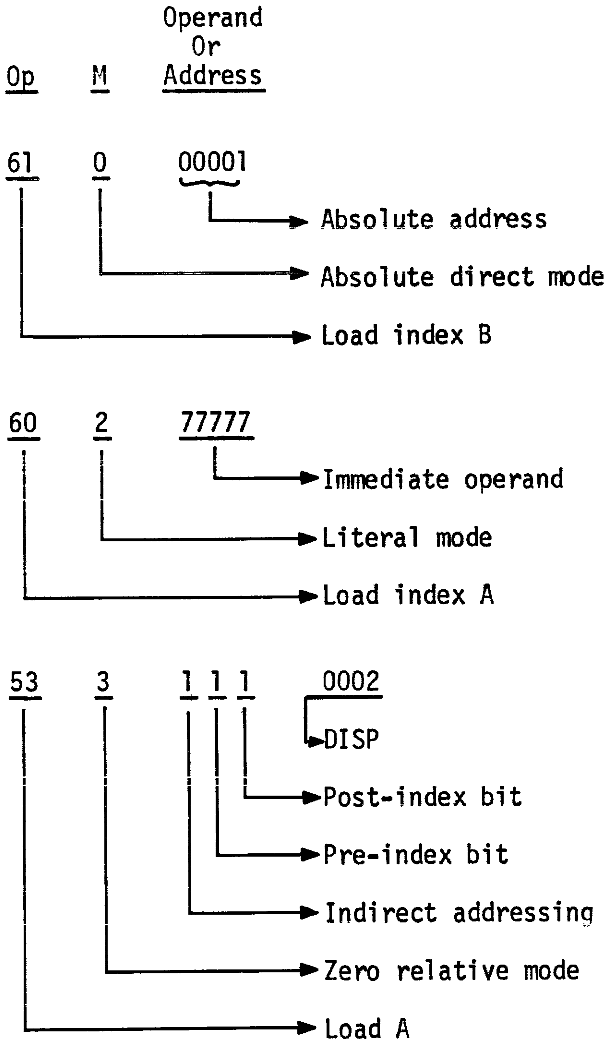
A hardware interrupt handler interrogates the CPS register bits to determine the exact cause of interrupt, and resets the appropriate bit in CPS to prevent immediate reinterrupt. Bit Manipulation commands (BIT) can be used to test and reset CPS bits (address 7765).

To properly service a hardware interrupt, access to the instruction or instructions that generated the interrupt is often required. The following example illustrates one method of accessing an offending instruction.

Example a:

Assume that the effective address of an instruction at memory location 04671 causes a fence violation interrupt to level 01 and that one of the functions of the level 01 interrupt handler is to analyze the instruction that caused the interrupt. Any instructions (except BSP, BSR, and instructions that only read information from memory without altering it in any way) can cause this interrupt. (Refer to Section 4 for BSP and BSR instruction.) In this case, the PC register contains 04672 at the time of interrupt, assuming a one-word instruction. After interrupt, this address is stored in location v + 2, where v is the interrupt vector of level 01. One exception to this occurs if the offending instruction is a Compare With Memory (refer to Section 4).

The following three instructions load the offending instruction into the A register:



Result

Level 01 vector is loaded into index B for preindexing; V → XB

The value 77777 (effectively -1) is loaded into index A for post-indexing; 77777 - XA

Contents of effective address replace contents of A; EA → A

EA = (0 + DISP + XB) + XA

or

EA = (2 + v) + 77777

Exampble b:

If the offending instruction is in memory location 04365 (PC at the time of interrupt) and the contents of location v + 2 after interrupt is 04366, then the previous instructions (in example a) load the A register with the contents of 04365.

TRAPS

The CP provides two memory traps. A trap is a hardware-forced branch to a specific location.

The two memory locations with the absolute addresses 00100 and 00101 are trap locations. Only two instructions can cause a trap:

a. BSP - Branch and Save Place
b. BSR - Branch and Save Region

If the effective address of either of these instructions violates the memory protection fence, the central processor traps to location 00100 (fence violation trap).

If either instruction specifies the literal mode of addressing (immediate operand, M = 2), the central processor traps to location 00101 (literal trap).

In either case, trapping action forces a 1 into the EMB bit (bit 00) of the PL register. This gives all privileges to the trap handling program. Also, trapping action causes the effective address (fence violation trap) or immediate operand (literal trap) to be stored in the memory location that is one location beyond the current top of the push-down stack.

Refer to the descriptions of BSP and BSR instructions in Section 4 for the effect of these instructions on the XT register.

Trapping allows a program to branch to subroutines (such as operating system or executive subroutines) that are stored outside its memory fence without causing a memory fence violation (level 01 interrupt). Because trapping does not change the priority level of the operating program, subroutines that are called by trapping operate on the same priority level as the calling program. In addition, an interrupt takes longer than a trap; thus, time is saved by calling system subroutines this way.

# 6/OPERATING PROCEDURES

Manual procedures presented here are limited to those associated with the optional power distribution unit and those associated with off-line use of the processor control panel.

## POWER DISTRIBUTION UNIT OPERATION

Information presented here relates to the controls and indicators, and operating procedures for the optional power distribution unit (PDU).

### Controls and Indicators

Function and organization of the PDU are described in Volume 1, Section 3 of the FOX 1 Hardware System Overview and Reference Manual. Table 2-6-1 presents the functions of controls and indicators on the Auxiliary Power Control Panel (02AnA1A1), the Main and Standby Power Control Panel (02AnA1A2), and the Motor Generator Control Panel (02AnA1A3).

### Power Turn-On

Before starting the motor generator, set the STDBY POWER/MAIN POWER transfer switch to MAIN POWER. Set the MAIN/STBY POWER and MAIN/STDBY DISTRIBUTION circuit breakers to off (down). To start the motor generator, press the START switch. Observe that the MOTOR RUN indicator, the three AC INPUT indicators, and the MAIN POWER indicators are all illuminated. When the motor generator has attained normal running speed, the MAIN/STBY POWER and MAIN STDBY DISTRIBUTION circuit breakers may be turned on (up).

Table 2-6-1. PDU Controls and Indicators

| REFERENCE DESIGNATION | CONTROL OR INDICATOR | FUNCTION |
|---|---|---|
| 02AnA1A1DS1 | AUX POWER Green Indicator Lamp | Lights to indicate the presence of 50/60 Hz line power at the distribtuion terminal block. |
| 02AnA1A1CB1-CB5 | Branch Circuit Breakers | Protect and distribute 50/60 Hz line power to noncritical circuits. |
| 02AnA1A2M2 | OUTPUT VOLTAGE Meter | Monitors output voltage at distribution block. |
| 02AnA1A2M1 | OUTPUT CURRENT Meter | Monitors output current drain on power source at distribution terminal block. |
| 02AnA1A2DS2 | STBY POWER Red Indicator Lamp | Lights to indicate the presence of standby power at the distribution terminal block (the STDBY POWER/MAIN POWER switch is in the STDBY POWER position). |
| 02AnA1A2S1 | STDBY POWER/MAIN POWER Transfer Switch | Selects the power source supplied to the distribution terminal block. In the STDBY POWER position, 50/60 Hz line power is supplied. In the MAIN POWER position 400 Hz power from the motor generator is supplied. |

Table 2-6-1. PDU Controls and Indicators (contd)

| REFERENCE DESIGNATION | CONTROL OR INDICATOR | FUNCTION |
|---|---|---|
| 02AnA1A2DS1 | MAIN POWER Green Indicator Lamp | Lights to indicate the presence of 400 Hz power at the distribution terminal block (the STDBY POWER/MAIN POWER switch is in the MAIN POWER position). |
| 02AnA1A2CB1 | MAIN/STBY POWER Circuit Breaker | Protects and distributes power from the distribution terminal block to the MAIN/STDBY DISTRIBUTION circuit breakers. |
| 02AnA1A2CB2-CB7 | MAIN/STDBY DISTRIBUTION Circuit Breakers | Protects and distributes power from the distribution terminal block to the six critical branch circuits. |
| 02AnA1A3DS2-DS4 | AC INPUT Green PHASE 1/PHASE 2/ PHASE 3 Indicator Lamps | Light to indicate the presence of each phase of the 3-phase 50/60 Hz line power at the motor starter. |
| 02AnA1A3S1 | START Switch | When pressed, applies power to the motor generator. |
| 02AnA1A3DS1 | MOTOR RUN Green Indicator Lamp | Lights to indicate the energized condition of the motor generator. |
| 02AnA1A3S2 | STOP Switch | When pressed, removes power from the motor generator. |

## Transferring to Standby Power

When transferring to standby power, it is recommended that the motor generator be deenergized by pressing STOP switch, and that MAIN/STBY POWER and MAIN/STDBY DISTRIBUTION circuit breakers also be set to off. To transfer power, operate the STDBY POWER/MAIN POWER switch to the STDBY POWER position. Observe that the MAIN POWER indicator is extinguished, and the STBY POWER indicator is illuminated. Turn on the MAIN/STBY POWER and MAIN/STDBY circuit breakers to distribute power to the using units.

## Power Turn-Off

To remove output power from the power distribution unit(s), press the STOP switch and observe that the MOTOR RUN indicator becomes extinguished. It is not necessary to turn off any of the circuit breakers.

## CENTRAL PROCESSOR OFF-LINE OPERATION

Procedures for the operation of the FOX 1 Control Computer System may be classed by their use with the system on-line or off-line; off-line being the state before the foreground (process) programs are activated. The steps necessary to place a newly installed or newly modified system on-line include bootstrap loading, system generation, and off-line testing. Bootstrap loading places the machine in a state in which off-line utility programs and the system diagnostics are easily accessible to the operator. System generation tailors the operating system to both the machine configuration and the process control requirements of the particular installation. This section of the reference manual covers off-line operating procedures exclusive of system generation and testing. For the additional steps to be taken to go on-line, and for information on the subsequent on-line control of the job stream, refer to the FOX 1

Programmer's Software Manuals or other manuals appropriate to the specific task to be performed.

Control Panel

The central processor control panel enables programming, operating, or maintenance personnel to perform the following:

a. Start up and shut down the central processor and IO devices.

b. Display and alter the contents of memory and the principal registers.

c. Control instruction fetch and execution.

d. Control execution of diagnostic and drum track zero utility programs.

A locked cover over the lower third of the panel prevents unauthorized or accidental intervention during normal system operation. As a further safeguard, the START switch, CLEAR switch, and the REGISTER SELECT switch are disabled until the machine is placed in HALT status.

Control Panel Indicators and Switches - Figure 2-6-1 shows the layout of the control panel. A list of controls and indicators with their functions follows:

a. REGISTER DISPLAY - twenty-four lamps (set in groups of three) in ACTIVE status display information as it is transferred over the central processor bus (CPB). In HALT status, the buffered

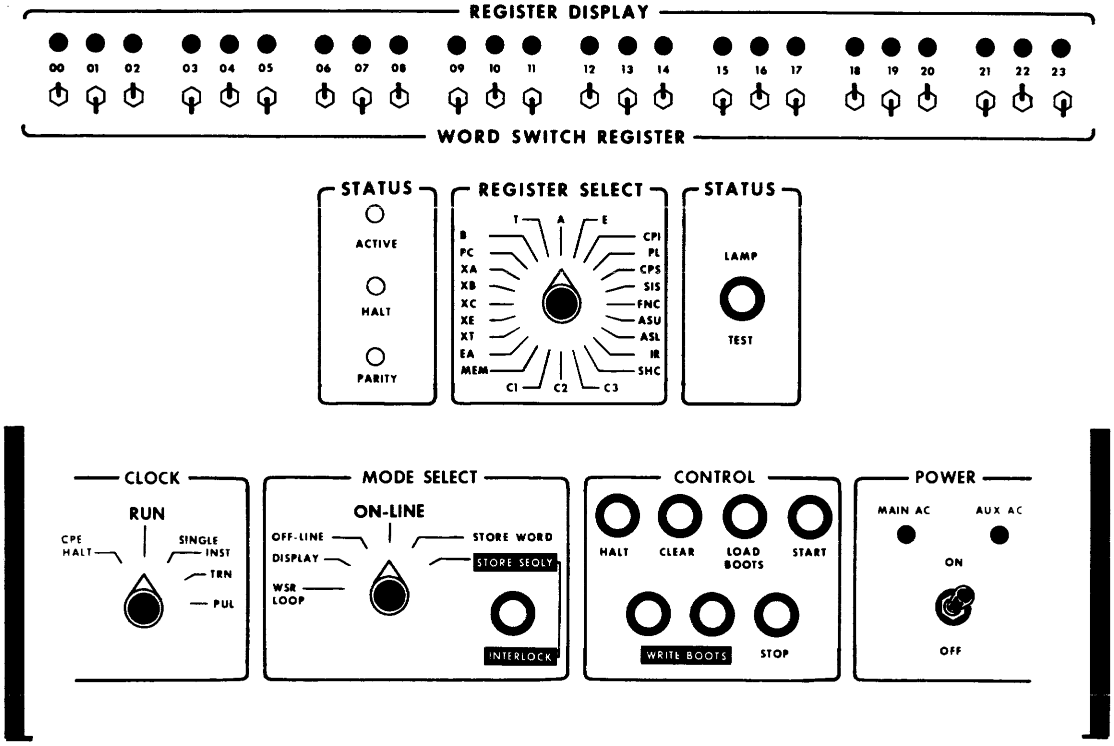


Figure 2-6-1. Central Processor Control Panel

display indicates the contents of the register, counters, control flip-flops, or memory location addressed by the REGISTER SELECT switch.

b. WORD SWITCH REGISTER (WSR) - twenty-four toggle switches form a read-only, addressable register. With these switches, the user can select diagnostic program options, control console utility programs, or enter addresses and data into memory and hardware registers.

c. STATUS Indicators (three lamps)

(1) ACTIVE - The central processor is executing instructions.

(2) HALT - The central processor is in a standby state, with CPS bit 23 (halt) set.

(3) PARITY - The parity error bit (22) of the CPS register is set. In normal operation (CLOCK set to RUN and MODE SELECT to ON-LINE), the error flip-flop is immediately cleared by the program to prevent further interrupts.

d. LAMP TEST Switch - This momentary pushbutton switch completes a filament path for testing only the STATUS indicator lamps.

e. CLOCK Selector (five-position rotary switch)

(1) RUN - The processor executes instructions until the HALT or STOP switch is pressed, or until a halt instruction is encountered while the program is operating at priority level 00.

(2) SINGLE INSTruction - The processor executes one instruction each time START on the CONTROL sector of the panel is pressed, allowing the user to "walk through" a program.

(3) CPE HALT - Instructions are executed as in the RUN switch position, unless a computer parity error (CPE) occurs. CPE halts the processor after the current instruction is completed.

(4) SINGLE TRN (Single Train) - The clock distribution system transmits only one cycle of clock pulses (four phases) each time the START switch (on the CONTROL sector of the panel) is pressed.

(5) SINGLE PULse - The clock distribution system transmits one pulse (sequentially by phase) each time the START switch is pressed.

f. REGISTER SELECT

This 24-position switch may be used in the HALT status in conjunction with the MODE SELECT switch to select memory and the principal registers for display or alteration. The shift counter and various control flip-flops may be displayed. Figure 2-6-2 identifies the information available on positions C1 through C3. Refer to the FOX 1 Central Processor Maintenance Manual, Theory of Operation section, for an explanation of the function of the listed counters, registers, and flip-flops. When the central processor is in the ACTIVE status, the REGISTER SELECT switch is disabled.

g. MODE SELECT Switch (six-position rotary switch)

(1) ON-LINE - This is the normal mode position. When the user changes the machine status from HALT to ACTIVE by pressing START (on the CONTROL sector of the panel), the central processor continues to execute instructions until a new halt condition is encountered.

(2) OFF-LINE - The central processor operates as in the ON-LINE mode, except that process interrupts are prevented.

(3) WSR LOOP - The central processor picks up the contents of the WORD SWITCH REGISTER as the instruction to be executed. The program counter is incremented each cycle even though the instruction word from memory is not used.

| | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C1 | | | | | | CFM5 | CFM4 | CFM3 | CFM2 | CFM1 | | | | | | | AS | OF | | MRKØ | MRK1 | MRK2 | MRK3 | MRK4 |
| | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

| | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C2 | OPS1 | OPS2 | OPS3 | MMM1 | MMM2 | AMC3 | AMC2 | AMC1 | CBM4 | CBM3 | CBM2 | CBM1 | CAM1 | CAM2 | CAM3 | CAM4 | CDM1 | CDM2 | | | CHM1 | CHM2 | | |
| | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

| | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C3 | CEM4 | CEM3 | CEM2 | CEM1 | | | | | CCM4 | CCM3 | CCM2 | CCM1 | CGM4 | CGM3 | CGM2 | CGM1 | | | | | | | | |
| | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

*Mnemonics refer to the following hardware:
Address Modification Counter -- AMC1 - AMC3
Interrupt Register -- MRKØ - MRK4
Execution Counters
CA M1 - M4
CB M1 - M4
CC M1 - M4
CD M1 & M2
CE M1 - M4
CF M1 - M5
CG M1 - M4
CH M1 & M2
Operation State Counter -- OPS1 - OPS3
Manual Mode Counter -- MMM1 & MMM2
Miscellaneous
Adder Save -- AS
Overflow -- OF

Figure 2-6-2. C1, C2, and C3 Register Display Identification*

(4) DISPLAY - Pressing START in this mode transfers to the REGISTER DISPLAY the contents of the memory location selected by the WORD SWITCH REGISTER. (See Note)

(5) STORE WORD - In this mode, pressing START transfers the contents of the WORD SWITCH REGISTER into the hardware register addressed by the REGISTER SELECT, or - in the MEMory position of the switch - into the memory location addressed by the EA register. (See Note)

(6) STORE SEQLY (store sequentially) - This mode may be used to load sequential data or instruction words into memory or to clear it. For sequential loading of program data (done with CLOCK set to SINGLE INSTruction), each time the START switch is pressed, the contents of the WORD SWITCH REGISTER are loaded into the memory location specified by the EA register, and the address in EA is incremented by one. For clearing memory, all 24 WORD SWITCH REGISTER switches are set to zero with the CLOCK in RUN to write zeros continuously throughout memory. (See Note)

h. INTERLOCK - Used in conjunction with STORE SEQLY, this switch prevents inadvertent destruction of the contents of more than one position of memory. If the INTERLOCK switch is not pressed in the clear memory procedure, the machine halts automatically after altering only one word.

NOTE

Used in conjunction with the REGISTER SELECT switch and the WORD SWITCH REGISTER. Refer to Control Panel Procedures.

i. CONTROL SWITCHES (seven momentary pushbutton switches)

(1) START - All manual and automatic operations are initiated from the HALT STATUS by pressing START.

(2) CLEAR - Effective only in the HALT status, it returns all registers and flip-flops to their initial state. All central processor registers are cleared to zero except PL (set to 0:40000000:) and PC (set to $00070_8$). The clear signal is transmitted over the PIO cable to reset control registers and flip-flops in all IO synchronizers.

(3) HALT - Brings the processor to the HALT status after execution of the current instruction. The program can be continued in sequence from this point without loss of information.

(4) STOP - When the central processor is in a loop such that the HALT switch is ineffective, this switch stops the operation by clearing the execution counters. This will probably cause the loss of information in registers or memory.

(5) LOAD BOOTS - Activates hardware that reads drum track zero into memory locations $00000\text{-}01777_8$. CLEAR must be pressed first for LOAD BOOTS to be effective.

(6) WRITE BOOTS - This pair of switches enables writing on drum track zero. Both switches must be pressed while the program to write the track zero programs is being executed. Refer to Auxiliary Bootstrap Loading Procedure.

j. POWER Section

(1) ON/OFF Switch - Enables the distribution of power to the system. Because the switch latches in the ON position, its handle must be pulled outward and down to remove system power.

(2) MAIN AC Lamp - Indicates power to the central processor.

(3) AUX AC (Auxiliary AC) Lamp - Indicates power to the blowers and IO devices.

Control Panel Procedures -

NOTE

All panel functions require HALT status. If the HALT status indicator is not on, press HALT. To bring the machine to HALT when the current operation cannot be completed, press STOP.

a. Display Hardware Register - Set REGISTER SELECT to desired hardware register.

b. Store Into Hardware Register

(1) Set WORD SWITCH REGISTER switches to coding for desired data word.

(2) Set REGISTER SELECT to desired hardware register.

(3) Set CLOCK to SINGLE INSTruction.

(4) Set MODE SELECT to STORE WORD.

(5) Press START.

c. Display Memory (Single Word)

(1) Set WORD SWITCH REGISTER switches to address of desired memory location.

(2) Set REGISTER SELECT to MEMory.

(3) Set CLOCK to SINGLE INSTruction.

(4) Set MODE SELECT to DISPLAY.

(5) Press START.

d. Store Into Memory (Single Word)

(1) Store address of desired memory location in EA register (as in procedure b.).

(2) Set WORD SWITCH REGISTER to coding for desired data word.

(3) Set REGISTER SELECT to MEMory.

(4) Set CLOCK to SINGLE INSTruction.

(5) Set MODE SELECT to STORE WORD.

(6) Press START.

e. Store Into Memory Sequentially

(1) Store address of desired initial memory location in EA register (as in procedure b, using initial memory location for data word, and EA for desired register).

(2) Set WORD SWITCH REGISTER switches to coding for desired data word.

(3) Set REGISTER SELECT to MEMory.

(4) Set CLOCK to SINGLE INSTruction.

(5) Set MODE SELECT to STORE SEQLY (store sequentially).

(6) Press START.

(7) Set WORD SWITCH REGISTER switches to coding for next data word in sequence.

(8) Press START.

(9) Repeat steps (7) and (8) for additional words.

f. Clear Memory

(1) Set WORD SWITCH REGISTER switches to all zeros or other desired pattern.

2) Set REGISTER SELECT to MEMory.

(3) Set CLOCK to RUN.

(4) Set MODE SELECT to STORE SEQuentially.

(5) Simultaneously press START and INTERLOCK.

(6) Release START and INTERLOCK. Machine should continue to run.

NOTE

If INTERLOCK is not pressed along with START, only one location is altered and machine stops automatically.

(7) Press HALT to stop the operation.

g. Single Instruction Loop

(1) Set WORD SWITCH REGISTER switches to coding for desired instruction word.

(2) Set CLOCK to SINGLE INSTRUCTION (or RUN).

(3) Set MODE SELECT to WSR LOOP.

(4) Press START.

h. Scan for Parity Error

(1) Store all zeros (or other starting address) in PC register (as in procedure b.).

(2) Set WORD SWITCH REGISTER switches as follows:

53 4 00000 (LDA $+Ø).

(3) Set REGISTER SELECT to PC.

(4) Set CLOCK to CPE HALT.

(5) Set MODE SELECT to WSR LOOP.

(6) Press START. Machine will stop with PARITY lamp lit and PC indicating the location of the error within two words. Display MEMory (as in procedure c.) in the vicinity of PC.

(7) If no parity error exists, the machine will stop with PARITY lamp lit at the first address beyond implemented memory. For machines with maximum memory, press HALT to stop operation.

j. Display Memory Sequentially (Using WSR Loop)

(1) Store desired first address in PC register (as in procedure b.).

(2) Set WORD SWITCH REGISTER switches as follows:

53 4 00000 (LDA $+Ø).

(3) Set REGISTER SELECT to A.

(4) Set CLOCK to SINGLE INSTRUCTION.

(5) Set MODE SELECT to WSR LOOP.

(6) Press START. REGISTER DISPLAY will indicate contents of memory at location addressed by PC.

(7) Repeat step (6) for additional words.

## SYSTEM START UP

Programs for initiating FOX 1 operation and the operating system are preloaded on the drum prior to system delivery. As long as these programs remain intact, the hardware bootstrap (see below) can be used for startup. To go from an erased drum (or one with unusable information) to an operational system, the same programs are available as binary streams on paper tape or cards. These tapes or card decks may be loaded by using an auxiliary, manually entered bootstrap program. Both methods for initial program loading are described below. For further information, refer to the System Generation Guide and the System Operation Guide.

### Initial Program Loading Using the Hardware Bootstrap

The FOX 1 central processor requires a program to obtain the basic instructions for reading in additional programs. A "bootstrap" program built into the hardware automatically - in response to the LOAD BOOTS key - reads loader programs stored on drum track zero into the first 1024 words of memory and selects one for execution.

Drum track zero contains six "loader" programs. Since two WRITE BOOTS pushbuttons must be pressed to permit writing on track zero, these programs are well protected from accidental erasure. The user selects the loader appropriate to his purpose by using one of the WORD SWITCH REGISTER settings shown in Figure 2-6-3. The WSR is decoded to provide the address for an N-way branch at memory location $00070_8$. When that branch instruction is executed, the program counter is set to the beginning of the desired loader program.

| | | (shown in octal) | |
|---|---|---|---|
| GDL | Gen Drum Loader | 4 ☒ f t a ☒ a d d | 00 02 04 13 18 23 |
| | fta = First Track Address<br>add = Two High-Order Digits of Core Address | | |
| DBL | Diag Boot Loader | 2 0 0 0 0 0 0 0 | 00 23 |
| DCPL | Diag Ctrl Prog Ldr | 1 0 0 0 0 0 0 0 | 00 23 |
| KTTYL | KTTY Loader | 0 4 ☒ | 00 05 06 23 |
| GCIL | Gen Core Image Ldr | | |
| | Load Operating Sys, Initialize, & Start | 0 1 0 ☒ | 00 08 09 23 |
| | Halt for Operator Intervention | 0 1 4 ☒ | 00 08 09 23 |
| KOLL | Off-Line Loader | | |
| | PTR Tape Input | 0 0 0 0 0 0 0 4 | 00 23 |
| | Card Input | 0 0 0 0 0 0 0 5 | 00 23 |
| | TTY Tape Input: Set TTY to TTs | 0 0 0 0 0 0 0 6 | 00 23 |
| | Card Input: Halt before executing Loader | 0 0 0 0 0 0 0 7 | 00 23 |

Figure 2-6-3. WSR Settings for Drum Track Zero Loader Programs

Start Up Procedure -

(1) Press HALT.

(2) Press CLEAR.

(3) Set WORD SWITCH REGISTER switches as indicated in Figure 2-6-3 for the desired program.

(4) Press LOAD BOOTS.

(5) Press START.

Initial Program Loading Using Manually Entered Bootstrap Program

The following procedure will take a system with core, drum, and registers in an unknown state to the point where drum track zero utility programs, the KTTY console utility program, and the diagnostics for the central processor and its standard peripherals are all easily accessible to the operator.

Load Drum Track Zero -

(1) Press HALT.

(2) Starting at location $00070_8$, manually enter the auxiliary bootstrap program (see Table 2-6-2), using the WORD SWITCH REGISTER switches and procedure e. for a store into memory sequentially operation.

Table 2-6-2. Auxiliary Bootstrap Program

| OCTAL | | ASSEMBLER LANGUAGE | |
|---|---|---|---|
| Location | Contents | Op Code | Address |
| 00070 | 21203040 | RDA | PTR |
| 00071 | 24000070 | BRZ | 0:70: |
| 00072 | 43200120 | RLS | 16 |
| 00073 | 43200310 | RLL | 8 |
| 00074 | 21203440 | RDSK | PTR |
| 00075 | 22000074 | BRU | 0:74: |
| 00076 | 71200002 | TIB | (2 |
| 00077 | 22000072 | BRU | 0:72: |
| 00100 | 46311300 | STE | 0:1300:,AX,NBREL |
| 00101 | 70200477 | TIA | (0:477: |
| 00102 | 22000072 | BRU | 0:72: |
| 00103 | 22001300 | BRU | 0:1300: |
| 00104 | 00000000 | DATA | 0 |

(3) Set CLOCK to RUN.

(4) Set MODE SELECT to OFF-LINE.

(5) Place the DTZRO program tape in the paper tape reader.

(6) Press CLEAR.

(7) Set WORD SWITCH REGISTER to 00000000$_8$.

(8) Prepare paper tape reader and teletype.

(a) Place tape in paper tape reader and set switch to RUN.
(b) Teletype MODE SWITCH to K.

(c) Teletype power switch to ON.
(d) Teletype line switch to ON LINE.

(9) While holding down both WRITE BOOTS pushbuttons, press START.

(10) Tape will load and teletype will begin typing drum track zero status message. WRITE BOOTS pushbuttons may be released after teletype begins typing.

If the teletype types only "END", drum track zero is successfully loaded at this point. If drum track zero has not been correctly loaded, a list of all discrepancies precedes the typing of "END". The format of this list is word number, correct contents, and apparent contents. Some of the possible reasons for discrepancies are failure to hold down both WRITE BOOTS buttons during the drum loading, a damaged drum, or an inoperable central processor or drum synchronizer.

Load KTTY Onto Drum Track One -

(1) When the message "END" signals that drum track zero has been loaded, place KTTY in the paper tape reader.

(2) Set WORD SWITCH REGISTER to $00000000_8$.

(3) Press CLEAR.

(4) Press LOAD BOOTS.

(5) Press START.

(6) The teletype bell signals that KTTY has been loaded and is in the external calling mode.

(7) Type "CD10000-17777.1/Ø" to transfer KTTY to the drum. A message "CD*" signals that KTTY is now loaded on drum track one.

## OFF-LINE TELETYPE CONSOLE PROGRAM (KTTY)

For many off-line operations, the control panel will only be needed to load the boostrap, loader, and KTTY programs. From that point on, KTTY, an off-line utility program, speeds and simplifies store and display operations by providing teletype input and output. The binary words used internally in memory and the drum can be entered in octal coding from the Teletype keyboard or printed out by the TTY or line printer as a string of octal characters. The operator can initiate IO operations with the drum, paper tape reader-punch, and the line printer.

### Operator Request Formats

KTTY accepts operator requets in one of the formats shown by Table 2-6-3. Two alpha characters specify the function: inspect and change core or drum (IC or ID), fill core or drum (FC or FD), search core or drum for a value under mask (SC or SD), compare tape to new core or new drum (NC or ND), or move information between core and drum (CD or DC), or between core or drum and an IO device (CP,CT or DP, DT). Five octal characters are expected for core addresses, eight for word values, three for drum track address, and four for drum word address. Insignificant leading zeros need not be keyed; the program right-justifies entries automatically. The end of an octal entry field is signalled by keying a space or any other nonoctal character (except "E" or "rubout"). KTTY would respond to "IC7Ø-" by typing eight octal characters representing the contents of memory location $00070_8$. Should more than the expected

## Table 2-6-3. KTTY Operator Request Formats

| OPERATION | KEYBOARD ENTRY FORMAT |
|---|---|
| Core to Drum | CDfwa-lwa.fta/fw- |
| Core to Drum, Module | CDMfwa-lwa.fta/fw- |
| Core to Printer | CPfwa-lwa. |
| Core to Tape | CTfwa-lwa. |
| Core to Tape, Module | CTMfwa-lwa. |
| Drum to Core | DCfwa-lwa.fta/fw- |
| Drum to Core, Module | DCMfwa-lwa.fta/fw- |
| Drum to Printer | DPfta/fw.lta/lw- |
| Drum to Tape | DTfta/fw.lta/lw- |
| Drum to Tape, Module | DTMfta/fw.lta/lw- |
| Execute | E |
| Fill Core with value | FCfwa-lwa.value- |
| Fill Drum with value | FDfta/fw.lta/lw.value- |
| Inspect Core | ICadd- |
| Inspect Drum | IDtrk/fw- |
| Load program | L |
| Load Object module | Obias.fta/fw- |
| Scan for New Core | NC- |
| Scan for New Drum | ND- |
| Search Core for value | SCfwa-lwa.value.mask- |
| Search Drum for value | SDfta/fw.lta/lw.value.mask- |
| Tape to Core | TC- |
| Tape to Drum | TD- |

NOTES:

If one track contains all the drum data to be transferred, a delimiter may be substituted for the last track address:

DPfta/fw.-/lw-

The header on tape contains the read in address for tape to core and tape to drum operations.

ENTRY PARAMETERS

| SYMBOL | DEFINITION | NO. CHAR'S EXPECTED |
|---|---|---|
| add | Core address | 5 |
| bias | Program relocation | 5 |
| fta | First track address | 3 |
| fw | First word (drum) | 4 |
| fwa | First word address (core) | 5 |
| lta | Last track address | 3 |
| lw | Last word (drum) | 4 |
| lwa | Last word address (core) | 5 |
| mask | Mask (control word) | 8 |
| trk | Track (one per ID request) | 3 |
| value | Value (data) word | 8 |

number of digits for a parameter be entered, the excess high order (leftmost) digits are ignored. Entering two delimiters in succession or a delimiter immediately after the request code is equivalent to entering a zero value for that parameter.

Inspect and Change Requests - The inspect and change mode improves upon the control panel display and store functions, and extends this capability to include the drum. The method of handling inspect and change requests differs according to the hardware addressed. Core information is printed word by word with the option to change the contents of the currently addressed word before proceeding to the next. KTTY displays and stores the contents of a drum location through an internal program buffer holding the contents of one full drum track. On each request cycle, a word from the drum image in the buffer is printed out and may be changed by keying in a new word. The information in the buffer area is transferred to the drum when the request is terminated. The contents of hardware registers may be inspected or changed through a 16-word register image block in KTTY. A request specifying a register prints the corresponding word from the register image, and any change is entered in the image, not the actual register. Information is transferred from the 16-word block to the addressable registers when an execute request causes KTTY to call a user program.

Execute Requests - KTTY may be used to initiate another program. Prior to the execution of a user program, appropriate register values should be present in the register image block. The following set of values is initially provided when KTTY is loaded:

| Register | Contents | |
|---|---|---|
| CPS | 3600 0000 | 32K core |
| FNC | 0000 7777 | "Backward" fence |
| PL | 4007 6001 | Exec mode; UL=ØØ, LL=31, IS=Ø1 |
| PC | 2000 | |
| XT | 1000 | |

To transfer information from the 16-word register image block in KTTY to the actual registers, "E" (execute) is keyed. Once the new values are loaded, the user program is initiated. KTTY always responds to "E" by attempting to execute an instruction at the memory location indicated by the program counter.

Core to Drum/Drum to Core (CD, CDM, DC, DCM) Requests - The information transferred between core and the drum may be in either a core image or a modular format. In the modular format - indicated by an "M" after the request code - a 16-word block of initial values for the addressable registers precedes the program instructions and data.

Only instruction and data words are transferred without the modular format option. The information for the 16-word block used in the modular format comes from the KTTY register image area. This information may be the result of prior user program activity, or may be manually stored through an inspect and change request. The default values will be used if no other information has been entered.

The operator should check the accuracy of each request - especially the direction of the transfer - before entering the final delimiter. Once started, a drum transfer takes place too quickly to be aborted. Entering "CD" in place of "DC" would wipe out the drum information meant to be read in.

Core to Print/Drum to Print (CP,DP) Requests - Often it is more expedient to print out an entire area of core or drum than to inspect individual words. KTTY prints core or drum "dumps" in blocks of eight words to the data line. The address of the first word in the data line - a core address, or track and word for the drum - is followed by the contents of eight consecutive memory or drum words. Parameters for starting and ending addresses are resolved to the nearest even eight locations that include the desired data. With the initial address of each successive line automatically increased by eight, printing continues until the line containing the last requested address has been completed. The next address beyond the dump area is printed with an asterisk (*) following, to signify completion of the request.

At system generation, KTTY is set up to handle core to print and drum to print requests on the line printer, if one is available. When the Teletype must be used, the starting address for each line of print precedes the data line; the line printer places the starting address on the same line as the data. Only the first of successive lines of identical data is printed. A caret (^) before the starting address of the next line of new information indicates that all lines omitted contain the same data as that above the caret. The operator can stop a long printout by pressing the Manual Interrupt button (MNI) on the side of the teletype. MNI must be held down until it is tested at the end of the print line.

Paper Tape IO (CT, CTM, DT, DTM, TC, TD) Requests - Figure 2-6-4 illustrates the four paper tape formats that KTTY can read and punch. The lower portion of the figure indicates how three tape frames are assembled into one memory word, or conversely, one memory word punched into three tape frames. The tape leader is unpunched; the first frame with at least one channel punched marks the beginning of a record. In tapes created by an external calling mode request, a header record indicates the source -- core or drum -- and the address from which it was punched. Because of this unique header, only paper tapes created in the external calling mode of KTTY can be read in by an external request. Also, the source indicated in the header read from tape must match the destination specified in a read request: a tape punched from the drum must be read into the drum; a request to read it into core would be rejected.

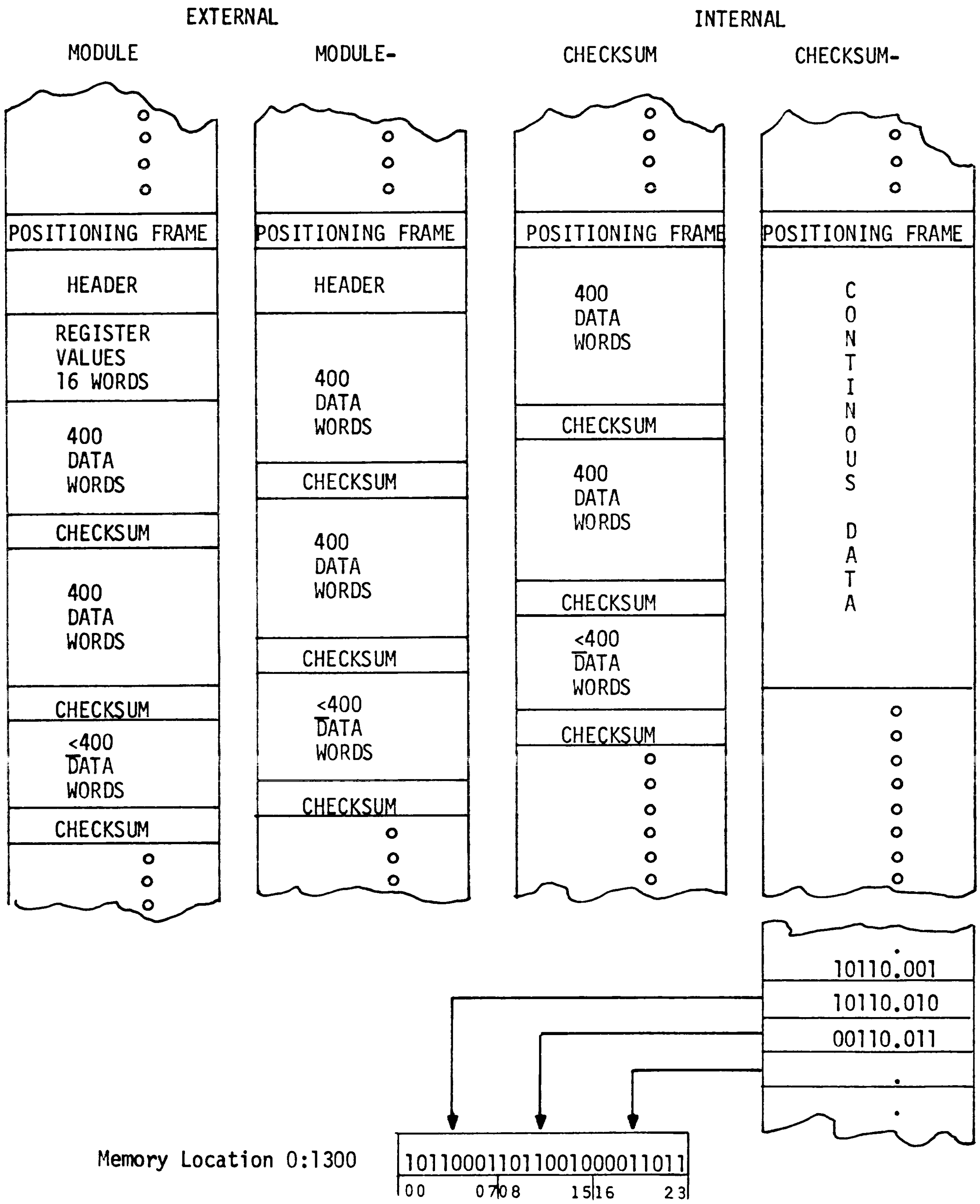


Figure 2-6-4. KTTY Paper Tape Formats

Programs may be preceded by a 16-word block containing initial values for the addressable registers. This modular format is called for by placing an "M" after the IO request code - i.e., "CTM" for core-to-tape module.

A 24-bit checksum taken after each 400 word (1200 frame) block and after the last word requested is punched in three frames following the block or last word. A checksum is accumulated as each segment is read into KTTY's internal buffer. No user core locations are modified (tape to core) unless the punched checksum for the segment agrees with that calculated by KTTY during read in. Similarly, only successfully checksummed segments are transferred to the drum (tape to drum). A checksum error is indicated by the message: "CER".

Operating Procedures

KTTY is permanently resident on drum track one, and loads into the highest available core. KTTY requires approximately $02000_8$ words of permanent drum storage and approximately $04000_8$ words in core (because of the buffer area that can hold a full track of drum information). An additional $400_8$ words is required for the off-line loader (KOLL). Parameters such as the number of drum tracks, amount of core, and equipment available are set at system generation. Using the appropriate startup procedure with WSR SWITCH 03 on, the operator selects KTTYL (KTTY loader permanently resident on drum track zero). The Teletype bell signals that KTTY has been loaded and is in the external calling mode - ready for TTY keyboard input. A message in one of the formats shown in Table 2-6-3 indicates to the program the desired operation.

An inspect core or inspect drum request is initiated by keying in "IC" or "ID", the octal address, and a delimiter; KTTY stays in the inspect and change mode until rubout is pressed. The Teletype types the contents of the memory or drum word currently selected, followed by one space. The operator may accept the displayed word, or respond with a new word (in octal) to be entered. In either case, he then chooses whether to display the next location by pressing carriage return, or the previous location by pressing line feed. Any delimiter other than line feed or rubout has the same effect as carriage return. If rubout is typed immediately following the octal field, the contents of that location are not changed.

Keying a period (.) terminates a series of sequential addresses, but leaves KTTY in the IC/ID mode. To store in a nonsequential location, the operator ends the current request with a period, then keys in the desired new address. KTTY responds with the contents of the new location. From that point, operation continues sequentially in the new series of addresses.

To change tracks during an inspect drum sequence, the operator keys a period (to close the current request), rubout (to transfer the KTTY buffer contents to the drum), and another ID request specifying the new track address.

IO operations are initiated from the Teletype by keying in a request code and a set of parameters as shown in Table 2-6-3. The request is terminated when the selected operation has run to completion, and KTTY is returned to the operator request entry point.

With KTTY loaded in core and another program active, the operator can summon KTTY by halting the central processor, setting SIS 00 from the control panel, and restarting the CP. System registers are saved (in the register image block of KTTY), all flags (control flip-flops) are cleared, and the KTTY restart point entered.

Operating Summary

The following is a summary of the operator controls for KTTY;

(1) Carriage return - As the final delimiter or only entry in reply to the IC/ID data word printed, causes KTTY to store the new data word (if one was entered), and print out the word at the next higher address.

(2) Line feed - As the final delimiter or only entry in reply to the IC/ID data word printed, causes KTTY to store the new data word (if one was entered), and print out the word at the next lower address.

(3) Rubout (RO)

(a) Following an octal field, rubout cancels that request and returns KTTY to the external request mode (to await a new keyboard entry).

(b) As the first character entered, rubout terminates an inspect core or inspect drum operation.

(4) Aborting a request

(a) Manual Interrupt button: may be used to abort any KTTY operation that produces a printout. The sequence is stopped at the end of the line being printed. The button must be held down because the interrupt is recognized only at the end of a print line.

(b) Aborting a punch operation: most readily done by turning off power on the paper tape punch.

(c) Aborting an invalid request: if gross errors, such as ending address less than starting address, are detected in the request parameters, the request is automatically aborted and an error message typed.

(5) "E", execute, starts a user program.

(6) "L" calls the off-line loader (KOLL) to bring a user program into core from the device specified by the WSR (see Figure 2-6-3). After loading is completed, control is returned to KTTY.

(7) "O" calls the object module loader (KLOM), which will load into core the procedure section of an assembled program. The origins assigned at assembly time will be displaced (relocated) by the amount specified in the bias field.

(8) Error messages printed by the teletype:

| | |
|---|---|
| (a) Computer parity error | Pxxxxx |
| (b) Invalid request | Hxxxxx |
| (c) Checksum error | CER |

NOTE

x's represent the program counter setting when the error was detected.

# 7/DRUM

Each FOX 1 Central Processor contains a drum memory subsystem which provides rapid access to relatively large quantities of data. Two choices of drum capacity are available: 256 tracks or 512 tracks (1 track = 1024 words). Up to 1024 words (the contents of one track) may be transferred to or from the drum in a single drum revolution. Drum revolution time is a function of the primary power frequency: 17.0 milliseconds for 60 Hz power and 20.4 milliseconds for 50 Hz power.

The drum synchronizer interfaces with both the Programmed IO and Channel IO facilities. Programmed IO handles interrupts, status transmission, and the initial drum-activating PIO instruction. Data transmission is handled by a high-speed channel IO. The standard drum PIO device address is $160_{10}$ $(240_8)$; its CIO assignment is channel 0; and its interrupt is assigned as level 02.

## DRUM STATUS REGISTER

The drum synchronizer contains the drum status register in which various operating and error conditions are recorded in addition to the standard BSY and DVI status bits. The format of the drum status register is:

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BSY | DVI | CPE | NOP | CHR | EOR | TME | DPE | WRT | x | x | x | x | x | SECTOR ADDRESS COUNTER | | | | | | | | | |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

These fields are defined as:

| Bits | Mnemonic | Definition |
|---|---|---|
| 00 | BSY | Busy. BSY is set upon receipt of an Initiate Channel IO (CIO) and is reset when this command is terminated. |
| 01 | DVI | Device Interrupt. DVI is set under one of the following conditions:<br>a. The number of words specified in the Channel Order Triple Word has been successfully transferred and the EOI bit of the Channel Order Triple Word is set.<br>b. CIO is terminated by an error condition specified by setting of bits 05, 06, or 07.<br>DVI is reset by execution of a CIO command. It can be cleared by an RSTC command and can be set or reset by a WST command. |
| 02 | CPE | Computer Parity Error. CPE is set if a parity error is detected during transfer of data into or out of CP core memory. If CPE is set, DVI is also set to cause an interrupt. CPE can be cleared by an RSTC command, and can be set or reset by a WST command. |
| 03 | NOP | No Operation. The NOP bit of the Channel Order Triple Word is recorded here and, if set, indicates that no data are to be transferred during this command. |
| 04 | CHR | Chaining Request. The CHR bit of the Channel Order Triple Word is recorded here and, if set, indicates that chaining is to occur. |
| 05 | EOR | End Of Record. EOR is set during a drum read or write operation when word 1023 has been read or written but the number of words specified in the Channel Order Triple Word has not been transferred, (i.e., automatic switching from track to track is not allowed). When EOR becomes set, the CIO command is terminated, DVI is set, and interrupt occurs.<br>EOR can be cleared by RSTC or can be written into by a WST command. |

| Bits | Mnemonic | Definition |
|---|---|---|
| 06 | TME | Timing Error. TME is set if the drum synchronizer loses synchronization with the drum. Synchronization is checked once per word on every drum revolution. When TME is set, DVI is also set, and if a transfer is in progress, the word will not be transferred, and an interrupt occurs.<br>TME can be cleared by RSTC or can be written by a WST command. |
| 07 | DPE | Drum Parity Error. DPE can be set only during a Read Data command. It is set if a parity error is detected in a word being read from the drum. When DPE becomes set, the command is terminated after the word containing the parity error is transferred into core memory, DVI is set, and interrupt occurs. |
| 08 | WRT | Write. WRT is set if the last or current command is a Write Data command. WRT is reset otherwise. |
| 09-13 | --- | Not used. |
| 14-23 | --- | Sector Address Counter. This field holds the sector number of the word being transmitted at any given time. This field is of use in drum maintenance or in a drum handler program for optimization purposes. |

## DRUM ADDRESSING

Any track address is allowed for drum transmission. However, writing on track 0000 is inhibited except when executed as a WRITE BOOT operation initiated from the CP Control Panel.

Reference to track 0000 through standard CIO and the sequence instruction process results in the command being accepted and executed in the normal manner except that no words are written. Termination is according to the termination bits of the instruction which initiates the writing.

Since the same synchronizer is used with both drum sizes, it is possible to send the synchronizer a track number larger than the size of the drum. If this happens, data are lost and not written for a Write Data command and meaningless data are input for a Read Data command.

## SPEED OF OPERATIONS

During one drum revolution, 1024 words (maximum) can be read or written. The average track access time is 8.5 milliseconds. Track selection, which is completely electronic, takes approximately 30 microseconds.

The continuous data rate is one computer word transmitted every 15.4 microseconds. An address counter within the synchronizer keeps constant record of the drum position for two reasons:

a. no more than one revolution is required to access any data after receiving a new instruction (word 1 of COTW),

b. by sequencing instructions, only three words will be missed between instructions.

To accomplish b above, the following must be true:

a. the previous channel order must have specified CHR = 1 and EOI = 0,

b. the previous operation must not cause an interrupt at its termination because of TME, CPE, DPE, or EOR.

Track switching requires approximately three word times. Therefore, transmissions may be terminated on one track and continued on another track without loss of a revolution if the first word transmitted on the new track is four sectors beyond the last word transmitted on the previous track and all conditions stated above are met.

For example, chaining from word 1000 of one track to word 1001 on the same or a different track cannot be done without losing a revolution. However, chaining from word 1000 of one track to word 1004 of the same or a different track can be done without an extra revolution.

Under normal conditions the gap between word 1023 and word 0000 of a track is sufficient (1 millisecond) to allow interrupt and restart of CIO without loss of a revolution. A normal EOI interrupt can be handled easily in such a situation without revolution loss, but error handling must be held to a minimum number of instructions. Thus, if all conditions for continuing transmission without time loss are met, chaining from word 1023 of a track to word 0000 of the same or a different track is assured.

## DRUM INSTRUCTION REPERTOIRE

In normal operation, programs to read and write data on the drum are controlled and executed by the Drum Librarian and other programs of the FOX 1 Operating System. These programs contain PIO instructions that address the drum synchronizer and effect operation of the drum. To address the drum, programs must have system IO privilege (SIOP) and PIO instructions must contain a device address of $160_{10}$ ($240_8$). These instructions can be assembled if this address code is defined with a mnemonic in the assembler prior to running the assembler, i.e.,

```
DRUM EQU 0:240
```

Extended mnemonics of the PIO instruction applying to the drum are presented in the following list (assuming literal mode) and described in detail in the succeeding paragraphs. All other instructions (instructions other than PIO containing a device address of $240_8$) are ignored by the drum.

| Mnemonic | Octal |
|---|---|
| RILS 02 | 21200002 |
| RST DRUM | 21207240 |
| RSTC DRUM | 21213240 |
| RSTCSK DRUM | 21213640 |
| WST DRUM | 21205240 |
| CIO DRUM | 21200240 |
| CIOSK DRUM | 21200640 |
| PNOP DRUM | 21215240 |

Name: Read Interrupt Level Status

Mnemonic: RILS 02

Op Code: 21

Timing: 9.60 μsec

Literal Class: 1

Command: X, DTI: X, ACU: X, SKP: X, Device Address: level 02

Registers Affected: A, DVI of all devices on interrupt level 02

Errors Possible: None

Symbolic Notation: DVIn → An

Description: This instruction reads the status of the device interrupt bits of all equipment at interrupt level 02 into preassigned bit positions of the A register. Since the drum is usually the only device assigned to this level, this instruction transfers the content of the DVI (bit 01 of the drum status register) into bit 00 of the A register.

All PIO instructions (Op = $21_8$) having device address codes from 00 through $27_8$ are decoded as RILS instructions. In RILS instructions, bits 09 through 15 are ignored, but the instructions are executed as if bits 13 and 14 were set (DTI = 1, ACU = 1).

The RILS instruction is used to test for a drum interrupt as follows:

| Op | M | Y | Result |
|---|---|---|---|
| 21 | 2 | 00002 | RILS at level 02. All DVI bits at level 02 → A |
| 23 | 2 | DRMINT | Branch on Negative A Register (BRN) to branch to DRuMINTerrupt routine if the drum DVI is set. |

Name: Read Status

Mnemonic: RST DRUM

Op Code: 21

Timing: 4.80 to 52.16 μsec normally, 2.24 μsec if executed without IO privilege.

Literal Class: 1

Command: 1, DTI: 1, ACU: 1, SKP: 0

Registers Affected: A, drum status register, CPS12, CPS13

Errors Possible: IO usage violation (IOU), IO timer violation (IOT)

Symbolic Notation: Drum status register → A

Description: The contents of the drum status register are loaded into (replace the previous contents of) the A register.

Example 1: Use RST DRUM to check for drum busy.

| Op | M | Y | Result |
|---|---|---|---|
| 21 | 2 | 07240 | RST DRUM. Drum status register → A. |
| 23 | 2 | DRMBSY | Branch on Negative A Register (BRN). Branch to DRuMBuSY subroutine if drum is busy or continue if drum is not busy. |

Example 2: Check for a drum-initiated interrupt on level 02 and then analyze the contents of the status register to determine the cause of the interrupt.

| Loc | Mnemonic | Address | Result |
|---|---|---|---|
|  | RILS | 02 | Level 02 DVI's → A |
|  | BRN | DRUMI | Branch if drum interrupt. |
|  | BRU | Other | Branch if not drum |
| DRUMI | RSTCSK | 160 | Read status of drum and clear after reading. This allows transmission to proceed if chaining is specified. |
|  | LLS | 02 | CPE → A00 |
|  | BRN | CPEI | Branch if CPE interrupt. |
|  | LLS | 03 | EOR → A00 |
|  | BRN | EORI | Branch if EOR interrupt. |
|  | LLS | 01 | TME → A00 |
|  | BRN | TMEI | Branch if TME interrupt. |

•
•
•

Operation complete; normal interrupt. Proceed with normal processing.

Name: Read Status and Clear

Mnemonic: RSTC DRUM

Op Code: 21

Timing: 4.80 μsec normally, 2.24 μsec if executed without IO privilege

Literal Class: 1

Command: 2, DTI: 1, ACU: 1, SKP: 0

Registers Affected: A, drum status register, CPS12, CPS13

Errors Possible: IO usage violation (IOU), IO timer violation (IOT)

Symbolic Notation: If BSY = 1, command rejected, proceed to next instruction
If BSY = 0, drum status register → A, then
0 → DVI, CPE, EOR, TME, and DPE

Description: If the drum is busy (BSY = 1), the command is rejected and the program advances to the next sequential instruction. If the drum is not busy (BSY = 0), the contents of the drum status register are loaded into (replace the previous contents of) the A register; then the DVI, CPE, EOR, TME, and DPE (bits 01, 02, 05, 06, and 07, respectively) bits of the drum status register are cleared.

Name: Read Status and Clear and Skip if Busy
Mnemonic: RSTCSK DRUM
Op Code: 21
Timing: 4.80 μsec to 52.16 μsec normally, 2.24 μsec if executed without IO privilege
Literal Class: 1
Command: 2, DTI: 1, ACU: 1, SKP: 1
Registers Affected: A, drum status register, CPS12, CPS13
Errors Possible: IO usage violation (IOU), IO timer violation (IOT)
Symbolic Notation: If BSY = 1, command rejected, proceed to next instruction
If BSY = 0, drum status register → A
0 → DVI, CPE, EOR, TME, DPE
PC + 1 → PC

Description: If the drum is busy (BSY = 1), the command is rejected and the program advances to the next sequential instruction. If the drum is busy (BSY = 0), the contents of the drum status register are loaded into (replace the previous contents of) the A register; the DVI, CPE, EOR, TME, and DPE (bits 01, 02, 05, 06, and 07, respectively) bits of the drum status register are cleared; and the program counter is incremented to skip the next sequential instruction.

Name: Write Status
Mnemonic: WST DRUM

Op Code: 21

Timing: 4.80 μsec to 52.16 μsec normally, 2.24 μsec if executed without IO privilege

Literal Class: 1

Command: 1, DTI: 0, ACU: 1, SKP: 0

Registers Affected: A, drum status register, CPS12, CPS13

Errors Possible: IO usage violation (IOU), IO timer violation (IOT)

Symbolic Notation: If A23 = 0, then A00 → BSY
A01 → DVI
A02 → CPE
A04 → CHR
A05 → EOR
A06 → TEM
A07 → DPE
If A23 = 1, then 0 → drum status register bits 00-07

Description: The least significant bit of the A register is sampled and, depending upon the result of this sampling, appropriate data writing operations are performed on the drum status register. If A23 is clear (A23 = 0), the contents of A00 through A02 and A04 through A07 are loaded into (replace the previous contents of) corresponding bits of the drum status register. If A23 is set (A23 = 1), bits 00 through 08 of the drum status register are cleared. Bits 09 through 23 of the drum status register are unaffected by the WST instruction. The WST instruction is not used in normal drum operations but is designed for use in maintenance diagnostic programs. It should also be noted that a WST command executed with a 1 in the least significant bit of the A register (A23 = 1), prevents further use of the drum for between one and two revolutions (16.7 milliseconds to 33.4 milliseconds for 60 Hertz power), even though the BSY bit is cleared.

Name: Initiate Channel IO

Mnemonic: CIO DRUM or CIOSK DRUM if SKP = 1

Op Code: 21

Timing: 4.80 μsec to 52.16 μsec normally, 2.24 μsec if executed without IO privilege

Literal Class: 1

Command: 0, DTI: 0, ACU: 0, SKP: 0 for CIO or 1 for CIOSK

Registers Affected: CPS12, CPS13, PC if SKP = 1

Errors Possible: IO usage violation (IOU), IO timer violation (IOT)

Symbolic Notation: If BSY = 1, command rejected, proceed to next instruction

If BSY = 0, command acknowledged, if SKP = 1, than PC + 1 → PC

1 → BSY
0 → DVI
1 → SIRF

Description: If the drum is busy (BSY = 1) when this instruction is issued, the command is rejected and the program advances to the next sequential instruction. If the drum is not busy (BSY = 0) when this instruction is issued, the command is acknowledged and executed. If the skip bit is set (SKP = 1), the program counter is incremented by 1 to skip the next sequential instruction. The busy bit is set, the device interrupt bit is cleared, and the sequence instruction request flip-flop is set (1 → SIRF). This instruction terminates and the computer program continues.

Setting of the sequence instruction request flip-flop causes the drum synchronizer to supply a Sequence Instruction Request to the Central Processor IO master. Drum operations to effect a data transfer with core memory are then under control of the IO master, as dictated by the conditions specified by the Channel Order Triple Word (COTW) stored in core memory.

Upon receiving the Sequence Instruction Request for service on channel 0 from the drum synchronizer, the IO master initiates a CIO transfer sequence as follows:

1. Obtain indirect address from NXAR0 (dedicated memory location 00050 for channel 0). This location must be preloaded with the address of a memory word which contains the address of the Channel Order Triple Word (COTW) for the transmission.

2. Read the first word of the COTW (containing the track number and start word) from core memory and send that word to the drum synchronizer.

3. Read the second word of the COTW (containing the word count) and store it in the High Speed Channel Data Word Count Register (CNT0).

4. Read the third word of the COTW (containing the first word address) and place it in the High Speed Channel Word Address Register (DAR0). This is the address of the first word to be transmitted.

5. Replace the contents of NXAR0 with the address of word four of the COTW.

6. Data transmission via channel IO facility begins.

The Channel Order Triple Word for drum operation has the following form:

Word 1

<table>
<tr><td>O<br>U<br>T</td><td>C<br>H<br>R</td><td>E<br>O<br>I</td><td>N<br>O<br>P</td><td colspan="10">TRACK NUMBER</td><td colspan="10">START WORD</td></tr>
<tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td></tr>
</table>

OUT If OUT = 0, reading begins from the track and word specified in bits 04-23.

If OUT = 1, writing begins into the track and word specified in bits 04-23. This bit becomes the WRT bit (bit 08) of the drum status register.

CHR If CHR = 0, no chaining will occur and the CIO command will terminate after the number of words specified in word two of the COTW have been transmitted.

If CHR = 1, chaining occurs when the number of words specified in the word count has been transmitted. This becomes the CHR bit (bit 04) of the drum status register.

EOI If EOI = 0, no interrupt will occur following the transmission specified by this COTW.

If EOI = 1, completion of the transmission specified by this COTW will cause DVI to be set and interrupt will occur through PIO.

NOP If NOP = 0, data will be transmitted according to OUT.

If NOP = 1, the command is executed but no data are transmitted. This bit becomes the NOP bit (bit 03) of the drum status register.

TRACK NUMBER The track of the drum into which or from which the transmission will occur.

START WORD The track sector number (0 through $1023_{10}$) which identifies the first word of the specified track which is to be involved in this transmission. This sector number is placed into the Sector Address Counter of the drum status register.

Word 2

<table>
<tr><td colspan="12">IGNORED</td><td colspan="12">WORD COUNT</td></tr>
<tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td></tr>
</table>

The word count for drum transmission should be less than $1024_{10}$ since track switching is not automatic.

Word 3

| IGNORED | | | | | | | | | FIRST WORD ADDRESS | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Word 4 (Optional)

| | | | | | | | | | CHAINING LINK ADDRESS | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

This address identifies the location of another COTW to be used in chained transmission. If CHR = 0 in word 1, this fourth word is unnecessary.

Example: Check the drum for busy. If not busy, clear the status register. Then, with a CIO command, initiate drum transmission to write the $500_{10}$ words beginning at location DAT1 on drum track $100_{10}$ ($144_8$) starting at sector 0, and then replace those $500_{10}$ words in memory with the $500_{10}$ words beginning at track $101_{10}$, sector $510_{10}$. Interrupt when finished.

| Loc. | Op | Address | Result |
|---|---|---|---|
| BEG | RSTCSK | $160_{10}$ | Read Status and Clear,<br>Skip if BSY = 0 |
| | BRU | BEG | Wait for BSY = 0 |
| | GEA | COTW1 | Address of COTW → E |
| | STA | TEMP | E → TEMP |

| Loc. | Op | Address | Result |
|---|---|---|---|
| | GEA | TEMP | Address of TEMP → E |
| | STE | 00050 | Store address of TEMP into NXARO |
| | GEA | DAT1 | Address of DAT1 → E |
| | STE | FWA1 | First Word Address → COTW1 |
| | STE | FWA2 | First Word Address → COTW2 |
| | GEA | COTW2 | Address of COTW2 → E |
| | STE | LINK | E → Chain Link |
| | CIO | 160 | Initiate Channel IO |
| | •<br>•<br>• | | |
| TEMP | XXXXXXXX | | Temperary storage for address of COTW1 |
| COTW1 | 60310000 | | OUT = 1, CHR = 1, EOI = 0, NOP = 0, Track + 144, Sector = 0 |
| | 00000764 | | $500_{10}$ = Word Count |
| FWA1 | XXXXXXXX | | First Word Address, generated by program |
| LINK | XXXXXXXX | | Chaining Link Address, generated by program |
| COTW2 | 10312776 | | OUT = 0, CHR = 0, EOI = 1, NOP = 0, TRACK = $101_{10}$, Sector = $510_{10}$ |
| | 00000764 | | $500_{10}$ = Word Count |
| FWA2 | XXXXXXXX | | First Word Address, generated by program. |

Since there is no further chaining, COTW2 needs no Chaining Link Address.

Name: No Operation

Mnemonic: PNOP DRUM

Op Code: 21

Timing: 3.52 to 26.88 μsec normally, 2.24 μsec if executed without IO privilege.

Literal Class: 1

Command: $\geq$ 3, DTI: 0, ACU: 1, SKP: 0

Registers Affected: CPS12, CPS13

Errors Possible: IO usage violation (IOU), IO timer violation (IOT)

Symbolic Notation: None

Description: This instruction performs no operation on data and is used only as a diagnostic command to check the response of the signal interface between the drum synchronizer and the Central Processor. This command is used to check generation of Acknowledge signals produced by the drum synchronizer in response to Command Out signals, Data Input Request signals (when DTI = 1), and Data Output Available signals (when DTI = 0) supplied by the Central Processor. If both bits 13 and 14 of this instruction are set (DTI = 1 & ACU = 1), this command clears the A register.

In summarizing the drum instructions:

a. RSTC, RSTCSK, CIO, and CIOSK commands are acknowledged, rejected, and not executed if the BSY bit is set; the BSY bit must be cleared to execute these commands.

b. RILS, RST, WST, and PNOP commands are executed (not rejected) regardless of the state of the BSY bit.

DRUM OPERATION

There are no manual procedures involved in normal operation of the drum. The operator should, however, periodically check the pressure gauges on the regulator of the helium supply as described under "Pressure System". Access to these gauges is obtained by loosening the four twist-lock screws at the right hand side of the circuit breaker/fuse panel (below the CP power supplies) and swinging this panel open on its hinges.

A small panel on the drum mechanism provides red indicator lamps that light to denote conditions in the drum; and also a drum-motor-reset push-button. Access to this panel is gained by unlocking and removing the bottom panel of the CP/IO unit (below the control panel and the Paper Tape Reader/Punch). The indicators and switch on this panel function as follows:

| Control or Indicator | Function |
|---|---|
| ACTUATION PRESSURE LOW indicator | Lights to denote detection of low pressure on the read/write heads. When this pressure is low the pneumatic pump is actuated to restore it to the nominal value. |
| DRUM SPEED LOW indicator | When the drum is energized, this lamp lights until the rotational speed reaches 3300 rpm (2750 rpm on 50-hertz power). On power or motor failure, this lamp lights when rotational speed drops below 3100 rpm (2583 rpm on 50-hertz power). Either of these conditions prevents data operations. When the drum-speed-low condition is detected, the pressure is removed from the read/write heads, retracting them from the surface of the disc. |
| DRUM TEMP HIGH indicator | This indicator lights to denote detection of excessive temperature in the drum shroud (150 F) or in the drive motor winding (300 F). When either of these conditions is detected, the motor is deenergized and the read/write heads are retracted from the surface of the disc. Either of these conditions prevents data operations. |

| Control or Indicator | Function |
|---|---|
| MOTOR RESET switch | Following correction of a high-temperature alarm condition this pushbutton switch is used to restart the drum drive motor and the read/write head pneumatic pump to enable continued operation of the drum. |
| MOTOR POWER ON indicator | This indicator lights to denote the energized condition of the drum drive motor. |

When either the DRUM SPEED LOW indicator or the DRUM TEMP HIGH indicator is lit, or when both are lit, the Acknowledge signal is inhibited in the drum synchronizer. In this condition any drum instruction causes an IO timer violation (IOT) and sets bit 12 of the central processor status register (1 → CPS12).

## PRESSURE SYSTEM

The actual drum read/write heads and storage medium operate in a sealed housing containing an environment of inert gas. A gas supply bottle and regulator, mounted behind the drum circuit card rack, maintain a pressure of 3/4 psi within the housing. This gas pressure should be checked once a week by reading the regulator low-pressure gauge to assure that it is between 3/4 psi and 1 psi. A gas supply bottle should last approximately three months and should be replaced when the pressure drops to 200 psi.

If the gas pressure goes to zero, the system will have to be purged to remove any contamination. Gas losses occur when the drum is turned on and off, heated and cooled, during rapid changes in atmospheric pressure (such as during air shipment), and due to normal leakage in the system.

The drum is shipped with a full (or nearly full) bottle of gas at a pressure of approximately 2500 psi and with the pressure system sealed at

1 psi. The gas system must be purged at installation before the drum is put into operation.

Any problem requiring the removal of the drum housing cover should be brought to the attention of The Foxboro Company. Unauthorized removal of the cover will nullify all warranties.

NOTE

> Do not, for any reason use any type of gas other than that (helium) originally supplied by The Foxboro Company. The pressure system control switches are set to operate only with the type of gas supplied.

Gas used for replacement or purging must be helium gas of highest grade, oil-free type, with a purity greater than or equal to 99.995%, in accordance with Liquid Carbonic Company, Inc. specification L-114 (Atomic Grade) or equivalent.

### Gas Bottle Replacement

Gas supply bottle replacement can be performed with the system in full operation as follows:

1. Close the gas supply bottle valve.

2. Close the regulator by turning the handle counterclockwise until it is free.

NOTE

> Never disconnect the nylon tubing from the output (low-pressure) side of the regulator or the pressure system will require purging.

3. Disconnect the regulator from the gas bottle, unsnap the latch holding the gas supply bottle, and remove the empty bottle.

4. Connect a full gas supply bottle to the regulator and strap the bottle in place.

5. Fully open the gas supply bottle valve.

6. Open the regulator by carefully turning the handle clockwise until the low pressure gauge just begins to show an increase in pressure. Adjust the regulator until the low pressure gauge indicates 3/4 psi (3/4 to 1 psi is acceptable).

7. Wait 10 minutes, then check the low pressure gauge and, if necessary, readjust the regulator to obtain the 3/4 psi pressure.

Purging Procedure

At installation and any time the gas pressure drops to zero, the pressure system must be purged. Purging can be accomplished with the system energized and operating by proceeding as follows:

1. Close the gas supply bottle valve.

2. Close the regulator by turning the handle counterclockwise until it is free.

3. Disconnect the regulator from the gas bottle, unsnap the latch holding the gas supply bottle, and remove the bottle from the gas supply regulator.

4. Connect a full gas supply bottle to the regulator and strap the bottle in place.

5. Fully open the gas supply bottle valve.

CAUTION

Do not exceed 3 psi on Step 6.

6. Open the regulator by carefully turning the handle clockwise and adjust the pressure to 2 psi, or just high enough to actuate the drum pressure relief valve.

7. Allow the contents of the gas supply bottle to flow through the pressure system for approximately 30 minutes.

8. Close the regulator by turning the handle counterclockwise until it is free.

WARNING

> Avoid physical contact with the motor starting capacitors in Step 9. These large capacitors are charged to the full primary line voltage and are capable of administering a relatively large electrical shock.

9. Manually open the pressure relief valve until the low pressure gauge indication falls to just below 3/4 psi. To do this, reach into the opening below the drum housing at the left side of the rack (as viewed from the back) and locate the pressure relief valve. Open the valve by pulling the valve cap downward with the fingernail. Manual opening of the pressure relief valve must be done very gently to prevent damage to the

spring that holds the valve closed. Damage to this spring can affect the adjustment that allows the valve to open if the pressure in the drum housing exceeds 2 psi.

10. Open the regulator by carefully turning the handle clockwise until the low pressure gauge just begins to show an increase in pressure. Adjust the regulator until the low pressure gauge indicates 3/4 psi (3/4 to 1 psi is acceptable).

11. Wait 10 minutes then read the low pressure gauge. Repeat steps 9 and 10, as necessary, to obtain the 3/4 psi pressure.

# 8/REAL TIME CLOCKS

Real time clocks provide the programming system with a means of measuring time in conventional units; that is, microseconds, milliseconds, seconds, etc. The real time clock equipment is completely under program control, except for the clock frequency which remains fixed for each clock. To measure a predetermined period of time, this length of time is divided by the frequency period of the clock. The resulting quotient is loaded into the real time clock under program control and the program continues to the next task. The clock decrements the number (preloaded quotient) once each frequency period, so that when the number becomes zero the predetermined period of time has elapsed since the number was loaded into the clock. When the number becomes zero, the real time clock requests a program interrupt to inform the program that the time has expired.

## FUNCTIONAL DESCRIPTION

The Real Time Clock Module is classified as optional equipment for the FOX 1 Central Processor. However, process control and foreground/background programming require use of a real time clock. The module contains a common interface to the CP PIO facilities and one, two, three, or four clocks, designated CLK0 through CLK3. Each module contains CLK0 which operates at the power line frequency (50 or 60 Hz). Additional clocks (CLK1, CLK2, CLK3) are implemented on an individual basis. These additional clocks are driven by a crystal oscillator and are available in several frequencies from 5 kHz to 100 kHz.

Each clock is similar in that each contains the following major circuits:

a. Clocking signal source. The basic time base for each clock is established by a pulse train. In CLK0 this pulse train is generated by a Schmitt trigger driven at the power line frequency so the output is a 50 Hz or 60 Hz square wave. In CLK1, CLK2, and CLK3 this pulse train is produced by an 8-bit binary counter that is driven by a crystal oscillator. The clocking signal can be taken from the output of any of the last five stages of this counter so that it is a square wave at the frequency of the crystal oscillator divided by 16, 32, 64, 128, or 256. The counter output tap is selected so that the final output signal is in the range of 5 kHz to 100 kHz.

b. Counter (CTR). The 12-bit CTR is loaded under program control, then decremented by one for each clocking signal received. It can also be read, started, and stopped under program control.

c. Device interrupt (DVI). When the CTR counts down to zero, the DVI flip-flop is set to request a program interrupt. The DVI can also be set, cleared, or read under program control.

d. Busy (BSY). This flip-flop controls and indicates the counting/not counting status of the CTR. It is set to enable counting when the CTR is loaded under program control and is cleared to inhibit counting when the contents of the CTR are read under program control. (It is also cleared by a Write Status instruction.)

e. Status register. Gating circuits provide a means of clearing, loading, and reading the BSY, DVI, and CTR simultaneously under program control as if they are one register. Bit assignments for Read Status and Write Status instructions are:

| BSY | DVI | NOT USED | | | | | | | | | | COUNTER (CTR) | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

The time measured by a real time clock is equal to the period of the clocking signal multiplied by the initial contents of CTR. The largest number that can be loaded into the 12-bit CTR is 4095. Therefore, the longest time that can be measured by CLK0 is 68.251 seconds at 60 Hz (16.667 msec × 4095 = 68.251365 sec.) or 81.9 seconds at 50 Hz (20.0 msec × 4095 = 81.9 sec.). Since the clocking signal frequency for CLK1, CLK2, and CLK3 is in the range of 5 kHz (200 μsec period) to 100 kHz (10 μsec period), the maximum time that can be measured on these clocks is 0.819 seconds (200 μsec × 4095 = 0.819 sec.).

## FUNCTIONAL SUMMARY

When the CTR is loaded by a Write Data instruction, BSY is set and the count down of CTR begins with the next clocking signal. When the contents of CTR becomes 0000, DVI is set to request a program interrupt. The BSY remains set and counting continues. The interrupt handler issues a Read Interrupt Level Status instruction to read the status of the DVI flip-flop of all devices at interrupt level 03 (all four real time clocks and any other devices at the same interrupt level). After determining the clock that caused the interrupt, the program issues a Read Data instruction for that clock to:

a. Clear BSY to stop CTR.

b. Clear DVI to remove the program interrupt request.

c. Read the contents of CTR. The CTR now contains the 2's complement of the number of clocking signals (if any) that occurred beyond the required number. This can occur if devices of higher priority require service simultaneously with the clock.

Variations are possible in the operations performed after determining the clock that requested a program interrupt. For example, if the interrupt handlers of all devices of higher priority are faster than the period of the clocking signal source, a Write Status instruction can be issued to clear BSY, DVI, and CTR. Clearing of CTR results in setting it to 7777.

## CLOCK ADDRESSES

Each real time clock is treated as an independent IO device by the CP programming system. Each clock is assigned a separate device address for the least significant eight bits of associated PIO instructions. The standard device address codes for the clocks are:

| Clock Mnemonic | Octal Device Address | IO Privilege Required |
|---|---|---|
| CLK0 | 300 | CIOP |
| CLK1 | 301 | CIOP |
| CLK2 | 100 | NIOP |
| CLK3 | 101 | NIOP |

All real time clocks are assigned to program interrupt priority level 03. Therefore the Read Interrupt Level Status instruction uses a device address of 003 to read the status of all clock DVI flip-flops into the A register. Bit assignments for the A register correspond to the clock number for this instruction:

```
CLK0 DVI → A00
CLK1 DVI → A01
CLK2 DVI → A02
CLK3 DVI → A03
```

CLOCK INSTRUCTION REPERTOIRE

Programmed commands for the real time clocks are microprogrammed (extended mnemonics) PIO instructions of the FOX 1 Central Processor. Instructions for each clock are as described in this text, and differ only in the device address codes; so to assemble a program using a real time clock it is simply necessary to define the mnemonic and device address (given previously under "Clock Addresses"). For example:

```
CLK0    EQU    0:300:
```

All commands for the real time clocks are always accepted; never rejected; regardless of the status of BSY. Therefore, no skip commands are practical (bit 15 is ignored).

Name: Write Data

Mnemonic: WDA CLK0, WDA CLK1, WDA CLK2, WDA CLK3

Op Code: 21

Timing: 4.80 to 52.16 μsec normally, 2.24 μsec if executed without IO privilege, 16.64 μsec when addressing a nonexistent clock

Literal Class: 1

Command: 0, DTI: 0, ACU: 1, SKP: 0

Registers Affected: A12-23, CTRn12-23, DVIn, BSYn

Errors Possible: IO usage violation (IOU), IO timer violation (IOT)

Symbolic Notation: A12-23 → CTRn12-23
0 → DVIn
1 → BSYn

Description: The counter (CTR or the least significant 12 bits of the status register) of clock n is cleared, then loaded from the least significant twelve bits of the A register. The busy (BSY) bit of the status register is cleared, and the clock is enabled to operate normally (by decrementing the new count the next time the oscillator completes a period).

Example: Set the counter of clock 3 to interrupt in 0.5 second; assuming the counting signal source is 8 kHz.

Counting signal period = 1 ÷ 8 kHz = 125 μsec
Decimal counts required = 0.5 sec ÷ 125 μsec = 4000
Octal counts required = 7640

| Mnemonic | M | Y | Response |
|---|---|---|---|
| LDA | 2 | 07640 | 7640 → A |
| WDA CLK3 | 2 | 01101 | A → CTR3 |

Clock 3 counter is set to 7640 and immediately begins counting. In 0.5 second a count of 0000 occurs and a level 03 interrupt is requested.

Name: Read Data
Mnemonic: RDA CLK0, RDA CLK1, RDA CLK2, RDA CLK3
Op Code: 21
Timing: 4.80 to 52.16 μsec normally, 2.24 μsec if executed without IO privilege, 16.64 μsec when addressing a nonexistent clock.
Literal Class: 1
Command: 0, DTI: 1, ACU: 1, SKP: 0
Registers Affected: A12-23, CTRn 12-23, BSYn, DVIn
Errors Possible: IO usage violation (IOU), IO timer violation (IOT)

Symbolic Notation: CTRn12-23 → A12-23
0 → BSYn
0 → DVIn

Description: The counter of the specified clock is inhibited, then the contents of the counter (CTR or the least significant 12 bits of the status register) are read into the least significant 12 bits of the A register. Both the busy bit (BSY) and the device interrupt bit (DVI) are cleared. The clock remains stopped until restarted by a Write Data or Write Status instruction.

Name: Write Status
Mnemonic: WST CLK0, WST CLK1, WST CLK2, WST CLK3
Op Code: 21
Timing: 4.80 to 52.16 μsec normally, 2.24 μsec if executed without IO privilege, 16.64 μsec when addressing a nonexistent clock
Literal Class: 1
Command: 1, DTI: 0, ACU: 1, SKP: 0
Registers Affected: A, CLKn status register
Errors Possible: IO usage violation (IOU), IO timer violation (IOT)
Symbolic Notation: If A23 = 0; then A00 → BSY, A01 → DVI
If A23 = 1; then 0 → BSY, 0 → DVI, 7777 → CTRn

Description: The least significant bit of the A register is sampled and, depending upon the result of this sampling, appropriate operations are performed on the real time clock status register. If bit A23 is clear (A23 = 0), the contents of the two most significant bits of the A register are loaded into the two most significant bits of the status register (A00 → BSY, A01 → DVI) and sensed by the appropriate real time clock (specified by the device address portion of the instruction). If bit A23 is set (A23 = 1), the two most significant bits of the status register are cleared (0 → BSY, 0 → DVI) and a full count is loaded into the twelve least significant bits of the status register (7777 → CTR) for sensing by the program-specified real time clock. This instruction can be used to start or stop a clock, cause or reset an interrupt request, or to reset all status bits.

Examples:

a. Clear clock 1.

| Op | M | Y | Results |
|---|---|---|---|
| LDA | 2 | 00001 | 1 → A; i.e.,<br>A00-23 = 0, A23 = 1 |
| WST CLK 1 | 2 | 05301 | Clear clock 1 |

b. Stop counting of clock 2 and cause clock 2 to request an interrupt. Assume K1 = $20000000_8$.

| Op | M | Address | Results |
|---|---|---|---|
| LDA | X | K1 | 20000000 → A; i.e.,<br>A00 = 0, A01 = 1 |
| WST CLK2 | 2 | 05100 | Set DVI, reset BSY |

c. Start clock 3, then at some time later check elapsed time.

| Op | M | Y | Results |
|---|---|---|---|
| LDA | 2 | 00050 | |
| WDA CLK3 | 2 | 01101 | Load clock 3 with $50_8$ |
| | •<br>•<br>•<br>•<br>• | | |
| RDA CLK3 | 2 | 03101 | Stop clock 3 and read remaining count |
| | •<br>•<br>•<br>•<br>• | | Calculate elapsed time.<br>Elapsed time =<br>[50 - (A)] × clock period |

Name: Read Status

Mnemonic: RST CLK0, RST CLK1, RST CLK2, RST CLK3

Op Code: 21

Timing: 4.80 to 52.16 μsec, 2.24 μsec if executed without IO privilege,
16.64 μsec when addressing a nonexistent clock

Literal Class: 1

Command: 1, DTI: 1, ACU: 1, SKP: 0

Registers Affected: A, CLKn status register

Errors Possible: IO usage violation (IOU), IO timer violation (IOT)

Symbolic Notation: CTRn12-23 → A12-23
DVIn → A01
BSYn → A00

Description: The contents of the status register of the specified clock are loaded into (replace) corresponding bits of the A register. The counter is temporarily frozen to prevent reading during advance of the count. Incoming clock pulses are stored during this command, then when the command is completed the clock is updated automatically. No RST command is ever rejected, so bit 15 (SKP) is not used. An RST command executed immediately following a Write Status (WST) instruction or a master clear operation will read a full count (all 1's) from the counter, since resetting a clock counter establishes a count of 7777.

Name: Read Status and Clear

Mnemonic: RSTC CLK0, RSTC C1K1, RSTC CLK2, RSTC CLK3

Op Code: 21

Timing: 4.80 to 52.16 μsec normally, 2.24 μsec if executed without IO
privilege, 16.64 μsec when addressing a nonexistent clock

Literal Class: 1

Command: 2, DTI: 1, ACU: 1, SKP: 0

Registers Affected: A, CLKn status register

Errors Possible: IO usage violation (IOU), IO timer violation (IOT)

Symbolic Notation: CTRn12-23 → A12-23<br>
DVIn → A01<br>
BSYn → A00<br>
then 0 → DVIn

Description: The contents of the status register of the specified clock are loaded into (replace) the corresponding bits of the A register, then the device interrupt bit of the specified clock is cleared (0 → DVIn). The counter is temporarily frozen to prevent reading of changing data. Incoming clock pulses are stored during this instruction, then the counter is updated when the reading is completed. No RSTC command is ever rejected, so bit 15 (SKP) is not used. An RSTC command executed immediately following a Write Status (WST) instruction or a master clear operation will read a full count (all 1's) from the counter, since resetting a clock counter establishes a count of 7777.

Name: No Operation

Mnemonic: NOPL CLK0, NOPL CLK1, NOPL CLK2, NOPL CLK3

Op Code: 21

Timing: 3.52 to 26.88 μsec normally, 2.24 μsec if executed without IO privilege, 16.64 μsec when addressing a nonexistent clock

Literal Class: 1

Command: 3, DTI: 0, ACU: 1, SKP: 0

Registers Affected: None

Errors Possible: IO usage violation (IOU), IO timer violation (IOT)

Symbolic Notation: None

Description: This instruction performs no operation on data and is used only as a diagnostic command to check the response of the signal interface between the real time clock and the Central Processor. If bits 13 and 14 of this instruction are set (DTI = 1 & ACU = 1), this command clears the A register.

Name: Read Interrupt Level Status

Mnemonic: RILS 03

Op Code: 21

Timing: 9.60 μsec

Literal Class: 1

Command: X, DTI: X, ACU: X, SKP: X, Device Address: Level 03

Registers Affected: A, DVI of all devices on interrupt level 03

Errors Possible: None

Symbolic Notation: DVIn → An

Description: This instruction reads the status of the device interrupt bits of all equipment at interrupt level 03 into preassigned bit positions of the A register. The preassigned positions for the real time clocks are:

| Real Time Clock | A Register Bit Assigned |
|---|---|
| CLK0 | A00 |
| CLK1 | A01 |
| CLK2 | A02 |
| CLK3 | A03 |

All PIO instructions (Op = $21_8$) having device address codes from 00 through $27_8$ are decoded as RILS instructions. In RILS instructions bits 09 through 15 are ignored, but the instructions are executed as if bits 13 and 14 are set (DTI = 1, ACU = 1).

After executing this instruction the program can analyze the data read into the A register to determine the device currently requesting the level 03 interrupt. Upon detecting the device requesting service the program can read the status register of this device and clear the interrupt request by executing a Read Status and Clear (RSTC) instruction. This new status information in the A register can then be analyzed to determine the type of service required by the device (if necessary).

## CLOCK INSTRUCTION SUMMARY

Machine language coding of PIO instructions in the literal mode for all real time clocks is:

| Mnemonic | CLK0 | CLK1 | CLK2 | CLK3 |
|---|---|---|---|---|
| RILS | 21200003 | 21200003 | 21200003 | 21200003 |
| WDA | 21201300 | 21201301 | 21201100 | 21201101 |
| RDA | 21203300 | 21203301 | 21203100 | 21203101 |
| WST | 21205300 | 21205301 | 21205100 | 21205101 |
| RST | 21207300 | 21207301 | 21207100 | 21207101 |
| RSTC | 21213300 | 21213301 | 21213100 | 21213101 |
| NOPL | 21215300 | 21215301 | 21215100 | 21215101 |

# 9/SYSTEM SECURITY

Security features are built into many FOX 1 units, but the security of the entire system is based on the system security module. This logic monitors the central processor, process control devices, and system power to permit graceful degradation of system functions if a malfunction occurs. When a failure is detected by the hardware, the software is notified and takes appropriate action. In cases of major failure, such as loss of primary power, the software is given sufficient advance warning to permit shutdown of the computer system in an orderly fashion and storage of data in all active registers. This allows resumption of computer control from the point of interruption following short-term failures, and allows complete restart and recycling for long-term failures. In cases of less severe failure, such as malfunction of the Analog Input Unit or Digital IO Unit, the software can curtail system functions associated with the defective logic. For example, a failure in the Valve Control Output Module can cause a switching to an operating mode (condition yellow) in which the FOX 1 system monitors and alarms all process inputs but provides no output to the process.

This module consists of seven primary circuit elements:

a. Synchronizer logic
b. Security check logic
c. Stall alarm logic
d. Power failure/restart logic
e. Thermal alarm logic
f. Process control status logic
g. System flunk line

Physically, the system security module consists of:

a. Two printed wiring boards (D3001BE in location 117 and D3001MH in location 119) in the system interface nest of wing 1 of the CP/IO unit (01A1-A1-A1).

b. Two special, screw-mounted printed wiring boards (D3001SJ and D3001SK) behind the processor control panel in the center structure of the CP/IO unit (01A1-A5-A2).

c. A power failure timeout relay near the bottom of the center structure of the CP/IO unit (01A1-A5-A14).

d. Thermostatic switches and indicator lamps in the top of each wing and the center structure of the CP/IO unit (01A1-A1-A6, 01A1-A2-A4, 01A1-A3-A5, 01A1-A4-A6, and 01A1-A5-A13).

SYNCHRONIZER LOGIC

The synchronizer provides an interface to the PIO bus, allowing programmed interrogation and control of the system security logic circuits. The actual interface circuits (bus modifier, drivers, and receivers) are shared with other synchronizers in the system interface nest. A program-accessible 14-bit security control data register in the synchronizer is organized into the following fields:

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SAI | | PFT | STALL ALARM CLOCK | | | | | | | | | | | | | | | | SCOK | EPFT | ESA | RSA |
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

These data fields have the following meanings:

SAI Stall Alarm Interrupt. When set, this bit indicates that the stall alarm clock has timed out due to lack of execution of a Write Data instruction for over 1 second. Upon becoming set, the SAI bit initiates a program interrupt request at level 00, and energizes the stall alarm relay to disable the System Flunk Line. SAI cannot be set if ESA = 0.

PFT Power Failure Timeout. When set (upon restoration of electrical power), this bit indicates that the previous power failure exceeded the preset duration for acceptable continuation of the interrupted program. Therefore, the program must be restarted. This bit is set when the power failure timeout relay is deenergized and has timed out. This bit is cleared when the relay is energized by programmed setting of the EPFT bit.

STALL ALARM CLOCK This field indicates the contents of an 8-bit counter in the stall alarm clock. Each count corresponds to approximately 4.7 milliseconds of elapsed time since the clock was reset. The stall alarm clock is reset by the RSA bit during a Write Data instruction with bit A22 set. The stall alarm clock is read by a Read Data instruction.

SCOK Security Check OK. When set, this bit indicates that the CP exerciser programs have determined that the CP and process IO equipment are satisfactory for operation. When clear, this bit indicates that the CP exerciser programs have detected an error that should inhibit output through the process IO equipment. When set, this bit latches the SCOK relay to energize (enable) the System Flunk Line. When clear, this bit unlatches the SCOK relay to deenergize (disable) the System Flunk Line.

EPFT Enable Power Failure Time. This bit is set and cleared under program control, or is reset by a master clear pulse. Normally, this bit is set by executive and restart programs, and is cleared by a master clear pulse. When set, this bit indicates that the executive and restart programs have determined that electrical power within the FOX 1 System is satisfactory for operation. When clear, this bit indicates that a power failure has been detected. When set, this bit energizes the EPFT relay, which in turn energizes the power failure time delay relay (preventing it from timing out). When clear, this bit deenergized the EPFT relay, which deenergizes the time delay relay (allowing it to begin timing out).

ESA Enable Stall Alarm. When set, this bit enables the stall alarm interrupt bit to be set by timeout of the stall alarm clock. When clear, the ESA bit inhibits setting of the SAI bit. ESA has no affect on operation of the stall alarm clock.

RSA Reset Stall Alarm. Each time RSA is set under program control (by the Write Data instruction), the stall alarm clock is reset.

## SECURITY CHECK LOGIC

The security check logic prevents the system from supplying control signals to the process if the central processor exerciser programs have discovered a failure in the CP/IO unit or in the process IO units (calculation check, analog fail, digital IO fail, etc.). The result of the exerciser programs either sets the SCOK bit (bit 22) of the security control data register when no fault is detected, or clears this bit when a fault is detected.

A relay in the security check logic latches when the SCOK bit is set, and unlatches when this bit is clear. When latched, contacts of this relay energize (enable) the System Flunk Line. When unlatched, contacts of this relay deenergize the System Flunk Line. Contacts of this relay also trigger an alarm at the CP/IO unit and at all consoles to alert the operator. A local or master clear signal also clears the SCOK bit and causes the resulting disabling of the System Flunk Line.

When the System Flunk Line becomes disabled, process controllers switch back to an alternate control mode (analog and/or manual).

## STALL ALARM LOGIC

The stall alarm logic assures that the computer program does not stall or become locked in an inescapable loop due to a hardware or software failure. This logic contains a stall alarm clock composed of a free-running oscillator that drives an 8-bit counter. Three flip-flops and appropriate gating circuits control the stall alarm clock, and allow it to be reset and read under program control, and enable or inhibit a program interrupt request upon overflow. These flip-flops are the stall alarm interrupt (SAI, bit 01), enable stall alarm (ESA, bit 22), and the reset stall alarm (RSA, bit 23) bits of the security control data register. The stall alarm clock serves as bits 04 through 11 of the security control data register.

In normal operation the ESA is set to allow a program interrupt request, however, the stall alarm clock is reset by each 1-second program cycle so the clock does not overflow.

The programming system of each FOX 1 system is based on a one-second cycle, or repetition time. During this time the program checks hardware and software flags and executes all required program assignments. During each one-second period the executive routine executes a command that resets the stall alarm clock. If this command is not executed again within 1.2 seconds, the facility initiates a visual alarm and requests a program interrupt as an indication of program failure. Failure of the program to perform the complete cycle within this time is assumed to indicate a hardware malfunction that prevents the CP from executing instructions, or a program-fault condition such as an inescapable loop (program-stall condition).

The clock count times out approximately 1.0 second after the most recent reset (WDA SSEC instruction). The time-out interval is adjustable over a range of 0.95 to 1.3 seconds, and is normally set at 1.2 seconds. This setting has a long-term drift of less than ±10% and short-term repeatability is better than ±0.5%.

Clock count can be read into the A register at any time (by an RDA SSEC instruction), even if the stall alarm is not enabled. Disabling of the stall alarm (by either a WDA SSEC or WST SSEC instruction) prevents setting of the stall alarm interrupt (SAI) and generation of a program interrupt request, but does not affect the operation of the oscillator or counter. When SAI has been set, it can be cleared by a command (WST SSEC), by a Master Clear pulse, or by pressing the red local clear switch.

If the enable stall alarm (ESA) bit is set and the stall alarm clock overflows, the stall alarm interrupt (SAI) bit is set. Setting of SAI energizes a stall alarm relay and requests a program interrupt at level 00. Energizing the stall alarm relay operates contacts to disable the System Flunk Line and to light visual indicators at all consoles. Disabling the System Flunk Line causes all process controllers to switch to a backup mode of operation (analog or manual control).

The level 00 interrupt handler:

a. Reads and tests the contents of the software interrupt status register (SIS) to determine if a program caused the program interrupt.

b. Reads and tests the contents of the central processor status register (CPS) to determine if CP hardware caused the program interrupt.

c. Executes a Read Interrupt Level Status (RILS) instruction and subsequent test to determine if an IO device caused the program interrupt. (Bit 00 of the A register is set during a RILS instruction if the stall alarm caused the interrupt.)

d. Branches to a subroutine to service the request as soon as the requester is determined.

When the level 00 interrupt handler determines that the SAI bit caused the program interrupt (previous step c), it branches to a subroutine that executes an orderly shutdown of system operations. This routine also clears the SAI bit.

## POWER FAILURE/RESTART LOGIC

During the life of any continuously operating computer system, total power shutdowns will occur due to lightning or equipment failure. The power failure/restart logic monitors secondary power in the CP/IO unit and measures the duration of power interruption to determine the actions to be taken in shutdown and restart of the FOX 1 system. The choice is one of continuing with an interrupted process schedule or beginning over again upon resumption of power. Naturally, this decision depends upon the process being monitored and controlled, and upon the loading and capacity of the FOX 1 system. Therefore, the timing for these hardware decisions is variable and adjusted to suit the application.

Five power supplies in the CP/IO unit are monitored (by the D3001SK printed wiring board) for primary power interruption. An interruption is detected as a significant output voltage drop (i.e., significant for each supply, considering nominal level and supply characteristics). When an interruption is detected, approximately 600 microseconds of operating time remain before logical operations cannot be performed reliably. Upon detection of a power interruption, the main power failure bit (MPF, bit 21) of the central processor status register is set. Setting of the MPF bit initiates a program interrupt request at level 00.

When the level 00 interrupt handler determines that the MPF bit caused the program interrupt (step b as described previously for the stall alarm), it branches to a subroutine that executes an orderly shutdown of system operations. This action is planned to be completed within the alotted 600 microseconds. This routine also clears the MPF bit, then reads it again to see if the power interruption was of a transient nature. If the MPF bit remains clear the subroutine branches back and continues the interrupted program. If the MPF bit becomes set again, the power failure is of sufficient duration to halt operation. In this case, the subroutine ends with a Halt instruction.

When the interrupt handler halts the program, an 800 millisecond master clear signal is generated to initialize all registers and to activate the memory crowbar. The master clear signal clears the security control data register and deenergizes (or unlatches) all relays associated with this register. The crowbar effectively short circuits the memory power supply to prevent power supply transients from affecting the contents of core memory. Both the crowbar and master clear signal are latched and maintained until power is restored and stabilized (these conditions exist for any temporary power surges during the shutdown period).

Clearing of the security control data register by the master clear signal deenergizes all relays so the System Flunk Line is deenergized and the power failure timeout relay begins timing out. Adjustment of the power failure timeout relay is preset so the time delay matches the error time constant (the period for which the process can operate satisfactorily without receiving control signals from the FOX 1 system) of the specific process being controlled. When the time delay expires, this relay operates a set of contacts. If power has been restored by the time the time delay elapses, these contacts set the power failure timeout (PFT) bit of the security check data register.

When the circuits monitoring the power supplies indicate that power is restored, a delay of 100 milliseconds is initiated to assure that recovery is not temporary. If power remains for this 100 millisecond period, the crowbar is released to energize the core memory power supply. A second 100 millisecond delay is initiated to allow the memory power supply to stabilize, before the power clear signal is released. Approximately 800 milliseconds after the initial detection of a power failure, or 200 milliseconds after power is restored (whichever is longer), program control is forced (always) to the start-up routing beginning at address $00070_8$. This routine reads the contents of the security check data register into the A register (with an RDA SSEC instruction) and tests the state of the PFT bit. If the PFT bit is clear, the data environment of the program interrupted by the power failure is reestablished and that program is continued from the point of the interruption. If the PFT bit is set, the duration of the power failure was greater than the error time constant of the process being controlled, so the entire programming system is reinitialized and restarted.

## THERMAL ALARM LOGIC

A thermastatic switch and indicator lamps are provided in each wing and the center structure of the CP/IO unit, and in each major unit of the FOX 1 system. These switches operate at 130 ±3 F, which corresponds to an internal unit temperature of 110 F to 120 F. These switches are interconnected so that any overtemperature condition in the system operates an indicator at the offending unit and at each console. These indicators cause no alarm message or program variation, they simply inform the operator of a physical condition of the system.

## PROCESS CONTROL STATUS LOGIC

Status of the process and control system is indicated and partially controlled by logic circuits within each console. These circuits are discussed in detail in Volume 4, Section 1, Console, of this manual and in other FOX 1 literature. However, they are summarized here to give a more balanced presentation of system security.

Four system security indicators above the CRT light to indicate the following conditions:

| Indicator | Indication |
|---|---|
| CPU STALL OUT | Stall alarm interrupt (SAI) bit of security control data register is set and the stall alarm relay is energized. |
| SYSTEM HI-TEMP | Thermal alarm in any bay of the control system. |
| SECURITY CHECK | Security check OK (SCOK) bit of the security control data register is set and the SCOK relay is latched. |
| MMI HI-TEMP | Thermal alarm in the console. |

The console keyboard contains two indicating pushbutton switches that indicate the three process control states as follows:

| Switch | Color | Indication |
|---|---|---|
| PROCESS CONTROL ON | Green | On Line - The FOX 1 system is on control and on scan - controlling the process |
| PROCESS CONTROL OFF | Yellow | Standby - The FOX 1 system is off control and on scan - capable of controlling the process but not doing so |
| PROCESS CONTROL OFF | Red | Flunk - The FOX 1 system is off control and off scan |

The process operator and the computer software share control of the process control states; both may be overridden by the System Flunk Line. The operator can cause a change from Standby (yellow) to On Line (green) status by pressing the PROCESS CONTROL ON switch. The operator can also cause a change from On Line (green) to Standby (yellow) status by pressing the PROCESS CONTROL OFF switch. Either switch produces a program interrupt to call a program that actually accomplishes the transfer of status.

A power failure, security check, or stall alarm detected by the system security module forces the Flunk (red) status, causing all analog controllers to switch to an alternate (backup) control mode (analog or manual) and inhibiting the function of the PROCESS CONTROL ON switch at all consoles. Exit from the Flunk status is effected by:

a. Execution of a Write Status (WST SSEC with A23 = 1) instruction.

b. Generation of a master clear signal (by manually pressing the CLEAR switch on the processor control panel with the computer in the halted condition).

c. Manually pressing the local clear switch of the system security module.

The program can go to any of three security states (red, green, or yellow) by executing PIO instructions, using fixed alarm addresses reserved for this purpose. This is necessary for automatic restart after a power failure, and for automatic shutdown on alarm conditions or batch completion. When the operator presses the PROCESS CONTROL ON or PROCESS CONTROL OFF switch, unique codes are set up in the function key register and an interrupt is requested. If the operator presses the PROCESS CONTROL ON switch while the red indicator is lit, nothing happens.

A flunk relay located in the analog instrument console is controlled by either the process operator or the program when going to or from the green (On Line) state. The flunk relay, when deenergized, disables all valve signals from the CP/IO unit to the analog controllers. When the valve controllers are in a failed state, a unique bit is set in the function key register to give a true feedback of the status of the instrument console.

## SYSTEM FLUNK LINE

The System Flunk Line prevents the system from outputting to the process if deenergized. It is a series circuit consisting of: a contact of the stall alarm relay, a contact of the security check relay, the PROCESS CONTROL ON and PROCESS CONTROL OFF switches, the coil of the flunk relay, and the 24 vdc analog battery supply. The normal condition of this line, with the system controlling the process, is all contacts closed and the flunk relay energized. If any of the alarm conditions occur or the process operator pushes the PROCESS CONTROL OFF switch, the flunk relay deenergizes. The flunk relay contacts provide the return path for the fail relay in the controllers. When the contact is broken the controllers fail to their backup mode (analog or manual).

The DIO Flunk line consists of the series contacts of the stall alarm and security check relays. This condition is sensed by the Digital I/O Unit Synchronizer which takes the appropriate action.

The system flunk relay is capable of driving up to 250 ma at 28 vdc. The current requirement for a particular system depends on the number of controllers implemented. Since the current requirement is 20 to 30 ma per controller line, an external relay is required to drive this (relatively) high current. This relay will be driven from the system flunk relay. This external relay will be mounted in the Analog Instrumentation Cabinet along with the controllers.

## INSTRUCTION REPERTOIRE

Stored program instructions for the console are presented in Volume 4, Section 1 of this manual.

Programmed commands associated with the system security module are microprogrammed (extended mnemonics) of the PIO instructions of the FOX 1 central processor. In normal operation programs containing instructions to cause some function in the system security module are contained in fixed programs and subroutines, so assembly of these instructions is not necessary. When rewriting or revising these routines, the module instructions can be assembled by defining the mnemonic and device address in the symbol table as:

```
SSEC    EQU    0:242:
```

Because the system security module has no busy flag, all instructions addressing the module are accepted (never rejected), so skip commands are essentially an unconditional skip, and are not used (bit 15 is always clear). All instructions except PIO having a device address of $242_8$ are ignored by this module.

Instructions for the system security module are identified in the following list (literal mode) and in the subsequent detailed descriptions.

| Mnemonic | Octal Code |
|---|---|
| RILS 00 | 21200000 |
| WDA SSEC | 21201242 |
| RDA SSEC | 21203242 |
| WST SSEC | 21205242 |

Name: Read Interrupt Level Status

Mnemonic: RILS 000

Op Code: 21

Timing: 9.60 μsec

Literal Class: 1

Command: X, DTI: X, SKP: X, Device Address: Level 00

Registers Affected: A, DVI of all devices on interrupt level 00

Errors Possible: None

Symbolic Notation: DVIn → An

Description: This instruction reads the status of the device interrupt bits of all equipment at interrupt level 00 into preassigned bit positions of the A register.

All PIO instructions (Op = $21_8$) having device address codes from 00 through $27_8$ are decoded as RILS instructions. In RILS instructions bits 09 through 15 are ignored, but the instructions are executed as if bits 13 and 14 are set (DTI = 1, ACU = 1).

After executing this instruction the program can analyze the data read into the A register to determine the device currently requesting the level 00 interrupt. The preassigned position for the system security module is A00. Therefore, the source of an interrupt request can be determined to be the SSEC by executing a Branch On Negative A Register (BRN) instruction. Upon detecting that the SSEC is requesting service, the program can read the status register of this device by executing a Read Data (RDA SSEC) instruction. The interrupt request can then be cleared by executing a Write Status (WST SSEC) instruction (with A23 = 0). This new status information in the A register can then be analyzed to determine the type of service required by the device.

Name: Write Data

Mnemonic: WDA SSEC

Op Code: 21

Timing: 4.80 to 52.16 μsec normally, 2.24 μsec if executed without IO privilege

Command: 0, DTI: 0, ACU: 1, SKP: 0

Registers Affected: A20-23, SCOK, EPFT, ESA, RSA

Errors Possible: IO usage violation (IOU), IO timer violation (IOT)

Symbolic Notation: A20 → SCOK
A21 → EPFT
A22 → ESA
A23 → RSA

Description: The contents of A register bits 20 through 23 are loaded into corresponding bits of the security control data register.

Example: After a system has been initialized and started, the system security logic requires execution of a WDA SSEC command at least once per second to prevent flunking of the control devices. This command must set the SCOK, ESA, and RSA bits, and may also set EPFT. A typical sequence for this purpose is:

```
LDA (0:17:    SCOK, EPFT, ESA, RSA
WDA SSEC
```

Name: Read Data

Mnemonic: RDA SSEC

Op Code: 21

Timing: 4.80 to 52.16 μsec normally, 2.24 μsec if executed without IO privilege

Literal Class: 1

Command: 0, DTI: 1, ACU: 1, SKP: 0

Registers Affected: A, SAI, PFT, stall alarm clock

Errors Possible: IO usage violation (IOU), IO timer violation (IOT)

Symbolic Notation: 0 → A00
SAI → A01
0 → A02
PFT → A03
Stall alarm clock → A04-11
0 → A12-23

Description: The stall alarm interrupt (SAI), power fail timeout (PFT), and stall alarm clock data is loaded into the A register from the security control data register, and unused portions of the A register (A00, A02, A12-23) are cleared.

Name: Write Status

Mnemonic: WST SSEC

Op Code: 21

Timing: 4.80 to 52.16 μsec normally, 2.24 μsec if executed without IO privilege

Literal Class: 1

Command: I, DTI: 0, ACU: 1, SKP: 0

Registers Affected: A, SAI, SCOK, EPFT, ESA

Errors Possible: IO usage violation (IOU), IO timer violation (IOT)

Symbolic Notation: If A23 = 0; then A01 → SAI
A20 → SCOK
A21 → EPFT

$$\begin{array}{rll} & A22 & \rightarrow \mathrm{ESA} \\ \text{If } A23 = 1;\text{ then} & 0 & \rightarrow \mathrm{SAI} \\ & 0 & \rightarrow \mathrm{SCOK} \\ & 0 & \rightarrow \mathrm{EPFT} \\ & 0 & \rightarrow \mathrm{ESA} \end{array}$$

Description: The least significant bit of the A register is sampled and, depending upon the result of this sampling, appropriate operations are performed on the security control data register. If A23 is clear (A23 = 0), the contents of A register bits A01 and A20 through A22 are loaded into (replace the previous contents of) the corresponding bits of the security control data register. If A23 is set (A23 = 1), the stall alarm interrupt, security check OK, enable power failure timer, enable stall alarm, and reset stall alarm bits of the security control data register are cleared. All other bits (00, 02 through 19, and 23) of the security control data register are not affected by the WST SSEC instruction.

OPERATING PROCEDURES

There are no manual procedures associated with operation of the system security module. Alarm and failure procedures executed through the General Purpose Keyboard or Process Keyboard of the console affect the system security circuits. Refer to the Software System Operation Manual (73001CA) and Console Operator's Guide (73001EA) for these procedures.

A red local clear pushbutton switch at the edge of the System Security Synchronizer card (D3001BE at location 01A1-A1-A1-117) is available as a maintenance facility. Pressing this switch clears the SAI, SCOK, EPFT, and ESA bits of the security control data register in the same manner as by a master clear pulse or by execution of a Write Status instruction with A23 = 1.

# A/FOX 1 SYSTEM CHARACTER SET

| USACII-8 | | Binary | | | | | Hexa- | Card | Holl- |
|---|---|---|---|---|---|---|---|---|---|
| Character | Notes | P | 7 | 654 | 321 | Octal | decimal | Code | erith |
| BLANK | | 1 | 0 | 100 | 000 | 240 | 20 | No Punches | 0000 |
| ! | | 0 | 0 | 100 | 001 | 041 | 21 | 12-8-7 | 4006 |
| " | | 0 | 0 | 100 | 010 | 042 | 22 | 8-7 | 0006 |
| # | | 1 | 0 | 100 | 011 | 243 | 23 | 8-3 | 0102 |
| $ | | 0 | 0 | 100 | 100 | 044 | 24 | 11-8-3 | 2102 |
| % | | 1 | 0 | 100 | 101 | 245 | 25 | 0-8-4 | 1042 |
| & | | 1 | 0 | 100 | 110 | 246 | 26 | 12 | 4000 |
| ' | Apostrophe | 0 | 0 | 100 | 111 | 047 | 27 | 8-5 | 0022 |
| ( | | 0 | 0 | 101 | 000 | 050 | 28 | 12-8-5 | 4022 |
| ) | | 1 | 0 | 101 | 001 | 251 | 29 | 11-8-5 | 2022 |
| * | | 1 | 0 | 101 | 010 | 252 | 2A | 11-8-4 | 2042 |
| + | | 0 | 0 | 101 | 011 | 053 | 2B | 12-8-6 | 4012 |
| , | Comma | 1 | 0 | 101 | 100 | 254 | 2C | 0-8-3 | 1102 |
| - | Hyphen | 0 | 0 | 101 | 101 | 055 | 2D | 11 | 2000 |
| . | Period | 0 | 0 | 101 | 110 | 056 | 2E | 12-8-3 | 4102 |
| / | | 1 | 0 | 101 | 111 | 257 | 2F | 0-1 | 1400 |
| Ø | | 0 | 0 | 110 | 000 | 060 | 30 | 0 | 1000 |
| 1 | | 1 | 0 | 110 | 001 | 261 | 31 | 1 | 0400 |
| 2 | | 1 | 0 | 110 | 010 | 262 | 32 | 2 | 0200 |
| 3 | | 0 | 0 | 110 | 011 | 063 | 33 | 3 | 0100 |
| 4 | | 1 | 0 | 110 | 100 | 264 | 34 | 4 | 0040 |
| 5 | | 0 | 0 | 110 | 101 | 065 | 35 | 5 | 0020 |
| 6 | | 0 | 0 | 110 | 110 | 066 | 36 | 6 | 0010 |
| 7 | | 1 | 0 | 110 | 111 | 267 | 37 | 7 | 0004 |
| 8 | | 1 | 0 | 111 | 000 | 270 | 38 | 8 | 0002 |
| 9 | | 0 | 0 | 111 | 001 | 071 | 39 | 9 | 0001 |
| : | | 0 | 0 | 111 | 010 | 072 | 3A | 8-2 | 0202 |
| ; | | 1 | 0 | 111 | 011 | 273 | 3B | 11-8-6 | 2012 |
| < | | 0 | 0 | 111 | 100 | 074 | 3C | 12-8-4 | 4042 |
| = | | 1 | 0 | 111 | 101 | 275 | 3D | 8-6 | 0012 |
| > | | 1 | 0 | 111 | 110 | 276 | 3E | 0-8-6 | 1012 |
| ? | | 0 | 0 | 111 | 111 | 077 | 3F | 0-8-7 | 1006 |
| @ | | 1 | 1 | 000 | 000 | 300 | 40 | 8-4 | 0042 |
| A | | 0 | 1 | 000 | 001 | 101 | 41 | 12-1 | 4400 |
| B | | 0 | 1 | 000 | 010 | 102 | 42 | 12-2 | 4200 |
| C | | 1 | 1 | 000 | 011 | 303 | 43 | 12-3 | 4100 |
| D | | 0 | 1 | 000 | 100 | 104 | 44 | 12-4 | 4040 |
| E | | 1 | 1 | 000 | 101 | 305 | 45 | 12-5 | 4020 |
| F | | 1 | 1 | 000 | 110 | 306 | 46 | 12-6 | 4010 |
| G | | 0 | 1 | 000 | 111 | 107 | 47 | 12-7 | 4004 |

## FOX 1 SYSTEM CHARACTER SET (contd)

| USACII-8 Character | Notes | Binary | | | | Octal | Hexa-decimal | Card Code | Holl-erith |
|---|---|---|---|---|---|---|---|---|---|
| | | P | 7 | 654 | 321 | | | | |
| H | | 0 | 1 | 001 | 000 | 110 | 48 | 12-8 | 4002 |
| I | | 1 | 1 | 001 | 001 | 311 | 49 | 12-9 | 4001 |
| J | | 1 | 1 | 001 | 010 | 312 | 4A | 11-1 | 2400 |
| K | | 0 | 1 | 001 | 011 | 113 | 4B | 11-2 | 2200 |
| L | | 1 | 1 | 001 | 100 | 314 | 4C | 11-3 | 2100 |
| M | | 0 | 1 | 001 | 101 | 115 | 4D | 11-4 | 2040 |
| N | | 0 | 1 | 001 | 110 | 116 | 4E | 11-5 | 2020 |
| O | | 1 | 1 | 001 | 111 | 317 | 4F | 11-6 | 2010 |
| P | | 0 | 1 | 010 | 000 | 120 | 50 | 11-7 | 2004 |
| Q | | 1 | 1 | 010 | 001 | 321 | 51 | 11-8 | 2002 |
| R | | 1 | 1 | 010 | 010 | 322 | 52 | 11-9 | 2001 |
| S | | 0 | 1 | 010 | 011 | 123 | 53 | 0-2 | 1200 |
| T | | 1 | 1 | 010 | 100 | 324 | 54 | 0-3 | 1100 |
| U | | 0 | 1 | 010 | 101 | 125 | 55 | 0-4 | 1040 |
| V | | 0 | 1 | 010 | 110 | 126 | 56 | 0-5 | 1020 |
| W | | 1 | 1 | 010 | 111 | 327 | 57 | 0-6 | 1010 |
| X | | 1 | 1 | 011 | 000 | 330 | 58 | 0-7 | 1004 |
| Y | | 0 | 1 | 011 | 001 | 131 | 59 | 0-8 | 1002 |
| Z | | 0 | 1 | 011 | 010 | 132 | 5A | 0-9 | 1001 |
| [ | | 1 | 1 | 011 | 011 | 333 | 5B | 12-8-2 | 4202 |
| \ | | 0 | 1 | 011 | 100 | 134 | 5C | 0-8-2 | 1202 |
| ] | | 1 | 1 | 011 | 101 | 335 | 5D | 11-8-2 | 2202 |
| ^ | | 1 | 1 | 011 | 110 | 336 | 5E | 11-8-7 | 2006 |
| _ | Underscore | 0 | 1 | 011 | 111 | 137 | 5F | 0-8-5 | 1022 |
| LF | 1 | 0 | 0 | 001 | 010 | 012 | 0A | 12-2-8-9 | NA |
| FF | 1 | 0 | 0 | 001 | 100 | 014 | 0C | 12-4-8-9 | NA |
| CR | 1 | 1 | 0 | 001 | 101 | 215 | 8D | 12-0-3-8 | NA |

NOTES:

1. Not used by all IO equipment.
2. Some IO devices generate control codes not included in this table.

3. The character set used by the FOX 1 System is similar to ASCII-8. The code uses seven information bits plus an even parity bit.

4. FOX 1 input data can be prepared on off-line keypunch equipment using EBCDIC by observing the following symbol differences:

| FOX 1 | Key Punch | | Card |
|---|---|---|---|
| | EBCDIC | Letter Key (Shifted) | |
| [ | ¢ | R | 12-8-2 |
| \ | Ø-8-2 | T | Ø-8-2 |
| ] | ! | B | 11-8-2 |
| ^ | ¬ | G | 11-8-7 |
| ! | \| | Y | 12-8-7 |

# B/INSTRUCTION SUMMARY

Instructions Listed By Operation Code

| Octal Code | Mnemonic | Name | Notes | Page |
|---|---|---|---|---|
| 00XXXXXX | HLT | Halt | | 4-120 |
| 01XXXXXX | GEA | Generate Effective Address | | 4-118 |
| 02XXXXXX | AND | Logical AND | | 4-11 |
| 03XXXXXX | IOR | Inclusive OR | | 4-12 |
| 04XXXXXX | LLC | Load Logical Complement | | 4-7 |
| 05XXXXXX | XOR | Exclusive OR | | 4-13 |
| 06XXXXXX | BYT | Byte Manipulation | | 4-108 |
| 07XXXXXX<br>XXXXXXXX | BIT | Bit Manipulation | 2 | 4-103 |
| 07XXXXXX<br>00XXXXXX | RBIT | Reset SB And Continue | 1 | 4-107 |
| 07XXXXXX<br>01XXXXXX | BIT | Leave SB Unchanged And Continue (NOP) | | 4-107 |
| 07XXXXXX<br>02XXXXXX | CBIT | Change SB (Alternate) And Continue | | 4-107 |
| 07XXXXXX<br>03XXXXXX | SBIT | Set SB And Continue | | 4-107 |
| 07XXXXXX<br>04XXXXXX | SKSR | Skip If SB Is Set And Reset Bit | | 4-107 |
| 07XXXXXX<br>05XXXXXX | SKS | Skip If SB Is  Set And Leave SB Unchanged | | 4-107 |
| 07XXXXXX<br>06XXXXXX | SKSC | Skip If SB Is Set And Alternate SB | | 4-107 |

Instructions Listed By Operation Code (contd)

| Octal Code | Mnemonic | Name | Notes | Page |
|---|---|---|---|---|
| 07XXXXXX<br>07XXXXXX | SKSS | Skip If SB Is Set And Set SB | | 4-107 |
| 07XXXXXX<br>10XXXXXX | SKRR | Skip If SB Is Reset And Reset SB | | 4-107 |
| 07XXXXXX<br>11XXXXXX | SKR | Skip If SB Is Reset And Leave SB Unchanged | | 4-107 |
| 07XXXXXX<br>12XXXXXX | SKRC | Skip If SB Is Reset And Alternate SB | | 4-107 |
| 07XXXXXX<br>13XXXXXX | SKRS | Skip If SB Is Reset And Set SB | | 4-107 |
| 07XXXXXX<br>14XXXXXX | SKUR | Skip And Reset SB | | 4-107 |
| 07XXXXXX<br>15XXXXXX | SKU | Skip Unconditionally | | 4-107 |
| 07XXXXXX<br>16XXXXXX | SKUC | Alternate SB And Skip Unconditionally | | 4-107 |
| 07XXXXXX<br>17XXXXXX | SKUS | Set SB And Skip Unconditionally | | 4-107 |
| 10XXXXXX | ADD | Fixed Point Add Short | | 4-33 |
| 11XXXXXX | ADL | Fixed Point Add Long | | 4-34 |
| 12XXXXXX | SUB | Fixed Point Subtract Short | | 4-35 |
| 13XXXXXX | SBL | Fixed Point Subtract Long | | 4-36 |
| 14XXXXXX | MPY | Fixed Point Multiply | | 4-40 |
| 15XXXXXX | DIV | Fixed Point Divide | | 4-38 |
| 16XXXXXX | RFI | Return From Interrupt | | 4-95 |

Instructions Listed By Operation Code (contd)

| Octal Code | Mnemonic | Name | Notes | Page |
|---|---|---|---|---|
| 17XXXXXX | SPL | Set Privilege And Level Register | | 4-119 |
| 20XXXXXX<br>XXXXXXXX | CWM | Compare With Memory | 2 | 4-97 |
| 20XXXXXX<br>00XXXXXX | SZSE | Skip If Zero Sign Equals Memory Sign | | 4-101 |
| 20XXXXXX<br>01XXXXXX | BZSE | Branch If Zero Sign Equals Memory Sign | | 4-101 |
| 20XXXXXX<br>02XXXXXX | SZEQ | Skip If Zero Is Equal To Memory | | 4-101 |
| 20XXXXXX<br>03XXXXXX | BZEQ | Branch If Zero Is Equal To Memory | | 4-101 |
| 20XXXXXX<br>04XXXXXX | SZNE | Skip If Zero Is Not Equal To Memory | | 4-101 |
| 20XXXXXX<br>05XXXXXX | BZNE | Branch If Zero Is Not Equal To Memory | | 4-101 |
| 20XXXXXX<br>06XXXXXX | SZLT | Skip If Zero Is Less Than Memory | | 4-101 |
| 20XXXXXX<br>07XXXXXX | BZLT | Branch If Zero Is Less Than Memory | | 4-101 |
| 20XXXXXX<br>10XXXXXX | SZGT | Skip If Zero Is Greater Than Memory | | 4-101 |
| 20XXXXXX<br>11XXXXXX | BZGT | Branch If Zero Is Greater Than Memory | | 4-101 |
| 20XXXXXX<br>12XXXXXX | SZLE | Skip If Zero Is Less Than Or Equal To Memory | | 4-101 |
| 20XXXXXX<br>13XXXXXX | BZLE | Branch If Zero Is Less Than Or Equal To Memory | | 4-101 |

Instructions Listed By Operation Code (contd)

| Octal Code | Mnemonic | Name | Notes | Page |
|---|---|---|---|---|
| 20XXXXXX<br>14XXXXXX | SZGE | Skip If Zero Is Greater Than Or Equal To Memory | | 4-101 |
| 20XXXXXX<br>15XXXXXX | BZGE | Branch If Zero Is Greater Than Or Equal To Memory | | 4-101 |
| 20XXXXXX<br>16XXXXXX | TWBZ | Three-Way Branch On Zero Minus Memory | | 4-101 |
| 20XXXXXX<br>20XXXXXX | SASE | Skip If A Sign Equals Memory Sign | | 4-101 |
| 20XXXXXX<br>21XXXXXX | BASE | Branch If A Sign Equals Memory Sign | | 4-101 |
| 20XXXXXX<br>22XXXXXX | SAEQ | Skip If A Is Equal To Memory | | 4-101 |
| 20XXXXXX<br>23XXXXXX | BAEQ | Branch If A Is Equal To Memory | | 4-101 |
| 20XXXXXX<br>24XXXXXX | SANE | Skip If A Is Not Equal To Memory | | 4-101 |
| 20XXXXXX<br>25XXXXXX | BANE | Branch If A Is Not Equal To Memory | | 4-101 |
| 20XXXXXX<br>26XXXXXX | SALT | Skip If A Is Less Than Memory | | 4-101 |
| 20XXXXXX<br>27XXXXXX | BALT | Branch If A Is Less Than Memory | | 4-101 |
| 20XXXXXX<br>30XXXXXX | SAGT | Skip If A Is Greater Than Memory | | 4-101 |
| 20XXXXXX<br>31XXXXXX | BAGT | Branch If A Is Greater Than Memory | | 4-101 |
| 20XXXXXX<br>32XXXXXX | SALE | Skip If A Is Less Than Or Equal To Memory | | 4-101 |

Instructions LIsted By Operation Code (contd)

| Octal Code | Mnemonic | Name | Notes | Page |
|---|---|---|---|---|
| 20XXXXXX<br>33XXXXXX | BALE | Branch If A Is Less Than Or Equal To Memory | | 4-101 |
| 20XXXXXX<br>34XXXXXX | SAGE | Skip If A Is Greater Than Or Equal To Memory | | 4-101 |
| 20XXXXXX<br>35XXXXXX | BAGE | Brench If A Is Greater Than Or Equal To Memory | | 4-101 |
| 20XXXXXX<br>36XXXXXX | TWBA | Three-Way Branch On A Minus Memory | | 4-101 |
| 20XXXXXX<br>40XXXXXX | SESE | Skip If E Sign Equals Memory Sign | | 4-101 |
| 20XXXXXX<br>41XXXXXX | BESE | Branch If E Sign Equals Memory Sign | | 4-101 |
| 20XXXXXX<br>42XXXXXX | SEEQ | Skip If E Is Equal To Memory | | 4-101 |
| 20XXXXXX<br>43XXXXXX | BEEQ | Branch If E Is Equal To Memory | | 4-101 |
| 20XXXXXX<br>44XXXXXX | SENE | Skip If E Is Not Equal To Memory | | 4-101 |
| 20XXXXXX<br>45XXXXXX | BENE | Branch If E Is Not Equal To Memory | | 4-101 |
| 20XXXXXX<br>46XXXXXX | SELT | Skip If E Is Less Than Memory | | 4-101 |
| 20XXXXXX<br>47XXXXXX | BELT | Branch If E Is Less Than Memory | | 4-101 |
| 20XXXXXX<br>50XXXXXX | SEGT | Skip If E Is Greater Than Memory | | 4-101 |
| 20XXXXXX<br>51XXXXXX | BEGT | Branch If E Is Greater Than Memory | | 4-101 |

Instructions Listed By Operation Code (contd)

| Octal Code | Mnemonic | Name | Notes | Page |
|---|---|---|---|---|
| 20XXXXXX<br>52XXXXXX | SELE | Skip If E Is Less Than Or Equal To Memory | | 4-101 |
| 20XXXXXX<br>53XXXXXX | BELE | Branch If E Is Less Than Or Equal To Memory | | 4-101 |
| 20XXXXXX<br>54XXXXXX | SEGE | Skip If E Is Greater Than Or Equal To Memory | | 4-101 |
| 20XXXXXX<br>55XXXXXX | BEGE | Branch If E Is Greater Than Or Equal To Memory | | 4-101 |
| 20XXXXXX<br>56XXXXXX | TWBE | Three-Way Branch On E Minus Memory | | 4-101 |
| 20XXXXXX<br>60XXXXXX | SLSE | Skip If A, E Sign Equals Memory Sign | | 4-101 |
| 20XXXXXX<br>61XXXXXX | BLSE | Branch If A, E Sign Equals Memory Sign | | 4-101 |
| 20XXXXXX<br>62XXXXXX | SLEQ | Skip If A, E Is Equal To Memory | | 4-101 |
| 20XXXXXX<br>63XXXXXX | BLEQ | Branch If A, E Is Equal To Memory | | 4-101 |
| 20XXXXXX<br>64XXXXXX | SLNE | Skip If A, E Is Not Equal To Memory | | 4-101 |
| 20XXXXXX<br>65XXXXXX | BLNE | Branch If A, E Is Not Equal To Memory | | 4-101 |
| 20XXXXXX<br>66XXXXXX | SLLT | Skip If A, E Is Less Than Memory | | 4-101 |
| 20XXXXXX<br>67XXXXXX | BLLT | Branch If A, E Is Less Than Memory | | 4-101 |

Instructions Listed By Operation Code (contd)

| Octal Code | Mnemonic | Name | Notes | Page |
|---|---|---|---|---|
| 20XXXXXX<br>70XXXXXX | SLGT | Skip If A, E Is Greater Than Memory | | 4-101 |
| 20XXXXXX<br>71XXXXXX | BLGT | Branch If A, E Is Greater Than Memory | | 4-101 |
| 20XXXXXX<br>72XXXXXX | SLLE | Skip If A, E Is Less Than Or Equal To Memory | | 4-101 |
| 20XXXXXX<br>73XXXXXX | BLLE | Branch If A, E Is Less Than Or Equal To Memory | | 4-101 |
| 20XXXXXX<br>74XXXXXX | SLGE | Skip If A, E Is Greater Than Or Equal To Memory | | 4-101 |
| 20XXXXXX<br>75XXXXXX | BLGE | Branch If A, E Is Greater Than Or Equal To Memory | | 4-101 |
| 20XXXXXX<br>76XXXXXX | TWBL | Three-Way Branch On A, E Minus Memory | | 4-101 |
| 21XXXXXX | PIO | Programmed Input Output | 2 | 4-121 |
| 212XXXXX | RILS | Read Interrupt Level Status | 3, 4 | 4-133 |
| 212000XX | CIO | Initiate Channel IO | 4 | 4-132 |
| 212004XX | CIOSK | Initiate Channel IO And Skip | 4 | 4-132 |
| 212010XX | WDA | Write Data | 4 | 4-130 |
| 212014XX | WDASK | Write Data And Skip | 4 | 4-130 |
| 212030XX | RDA | Read Data | 4 | 4-131 |
| 212034XX | RDASK | Read Data And Skip | 4 | 4-131 |
| 212050XX | WST | Write Status | 4 | 4-129 |
| 212070XX | RST | Read Status | 4 | 4-127 |

Instructions Listed By Operation Code (contd)

| Octal Code | Mnemonic | Name | Notes | Page |
|---|---|---|---|---|
| 212130XX | RSTC | Read Status And Clear | 4 | 4-128 |
| 212134XX | RSTCSK | Read Status And Clear And Skip | 4 | 4-128 |
| 22XXXXXX | BRU | Branch Unconditionally | | 4-73 |
| 22200000 | RFS | Return From Subroutine | | 4-91 |
| 22400001 | NOP | No Operation | | 4-120 |
| 23XXXXXX | BRN | Branch On Negative A Register | | 4-70 |
| 24XXXXXX | BRZ | Branch On Zero A Register | | 4-72 |
| 25XXXXXX<br>XXXXXXXX | BSR | Branch And Save Region | | 4-81 |
| 26XXXXXX | BSP | Branch And Save Place | | 4-75 |
| 30XXXXXX | FAS | Floating Point Add Short | | 4-48 |
| 31XXXXXX | FSS | Floating Point Subtract Short | | 4-50 |
| 32XXXXXX | FMS | Floating Point Multiply Short | | 4-51 |
| 33XXXXXX | FDS | Floating Point Divide Short | | 4-53 |
| 34XXXXXX | FAL | Floating Point Add Long | | 4-49 |
| 35XXXXXX | FSL | Floating Point Subtract Long | | 4-50 |
| 36XXXXXX | FML | Floating Point Multiply Long | | 4-52 |
| 37XXXXXX | FDL | Floating Point Divide Long | | 4-53 |
| 40XXXXXX | RLE | Rotate Left E Register | | 4-16 |

Instructions Listed By Operation Code (contd)

| Octal Code | Mnemonic | Name | Notes | Page |
|---|---|---|---|---|
| 41XXXXXX | NMS | Normalize Short | | 4-26 |
| 42XXXXXX | NML | Normalize Long | | 4-29 |
| 43XXXXXX | SHF | Shift And Rotate | 2 | 4-17 |
| 432XX0XX | RRS | Rotate Right Short | 4 | 4-18 |
| 432XX1XX | RLS | Rotate Left Short | 4 | 4-19 |
| 432XX2XX | RRL | Rotate Right Long | 4 | 4-19 |
| 432XX3XX | RLL | Rotate Left Long | 4 | 4-20 |
| 432X04XX | ARS | Shift Right Short-Arithmetic | 4 | 4-21 |
| 432X05XX | ALS | Shift Left Short-Arithmetic | 4 | 4-22 |
| 432X06XX | ARL | Shift Right Long-Arithmetic | 4 | 4-23 |
| 432X07XX | ALL | Shift Left Long-Arithmetic | 4 | 4-25 |
| 432X14XX | LRS | Shift Right Short-Logical | 4 | 4-21 |
| 432X15XX | LLS | Shift Left Short-Logical | 4 | 4-23 |
| 432X16XX | LRL | Shift Right Long-Logical | 4 | 4-24 |
| 432X17XX | LLL | Shift Left Long-Logical | 4 | 4-26 |
| 44XXXXXX | STA | Store A Register | | 4-8 |
| 45XXXXXX | STL | Store Long | | 4-9 |

Instructions Listed By Operation Code (contd)

| Octal Code | Mnemonic | Name | Notes | Page |
|---|---|---|---|---|
| 46XXXXXX | STE | Store E Register | | 4-9 |
| 47XXXXXX | SNR | Store Normalized and Rounded | | 4-54 |
| 50XXXXXX<br>XXXXXXXX | MOV | Multiple Move | | 4-116 |
| 51XXXXXX | LDL | Load Long | | 4-6 |
| 52XXXXXX | LDE | Load E Register | | 4-5 |
| 53XXXXXX | LDA | Load A Register | | 4-4 |
| 54XXXXXX | EAM | Exchange A And Memory | | 4-10 |
| 55XXXXXX | MST | Masked Store | | 4-9 |
| 56XXXXXX | DEM | Decrement Memory | | 4-116 |
| 60XXXXXX | LXA | Load Index A | | 4-59 |
| 61XXXXXX | LXB | Load Index B | | 4-59 |
| 62XXXXXX | SXA | Store Index A | | 4-60 |
| 63XXXXXX | SXB | Store Index B | | 4-61 |
| 64XXXXXX | AXA | Add To Index A | | 4-61 |
| 65XXXXXX | AXB | Add To Index B | | 4-62 |
| 66XXXXXX | CXA | Compare Index A And Memory | | 4-62 |
| 67XXXXXX | CXB | Compare Index B And Memory | | 4-63 |
| 70XXXXXX | TIA | Test And Increment Index A | | 4-63 |
| 71XXXXXX | TIB | Test And Increment Index B | | 4-65 |
| 72XXXXXX | BDA | Branch And Decrement Index A | | 4-67 |
| 73XXXXXX | BDB | Branch And Decrement Index B | | 4-68 |

NOTES:

1. SB = Sensed Bit.
2. Basic Instruction never used in this form; always used in microcoded (extended mnemonic) form.
3. Device Address field contains interrupt level.
4. Octal code listed assumes literal mode.

Instructions Listed By Mnemonic

| Mnemonic | Octal Code | Name | Notes | Page |
|---|---|---|---|---|
| ADD | 10XXXXXX | Fixed Point Add Short | | 4-33 |
| ADL | 11XXXXXX | Fixed Point Add Long | | 4-34 |
| ALL | 432X07XX | Shift Left Long-Arithmetic | 4 | 4-25 |
| ALS | 432X05XX | Shift Left Short-Arithmetic | 4 | 4-22 |
| AND | 02XXXXXX | Logical AND | | 4-11 |
| ARL | 432X06XX | Shift Right Long-Arithmetic | 4 | 4-23 |
| ARS | 432X04XX | Shift Right Short-Arithmetic | 4 | 4-21 |
| AXA | 64XXXXXX | Add To Index A | | 4-61 |
| AXB | 65XXXXXX | Add To Index B | | 4-62 |
| BAEQ | 20XXXXXX<br>23XXXXXX | Branch If A Is Equal To Memory | | 4-101 |
| BAGE | 20XXXXXX<br>35XXXXXX | Branch If A Is Greater Or Equal To Memory | | 4-101 |
| BAGT | 20XXXXXX<br>31XXXXXX | Branch If A Is Greater Than Memory | | 4-101 |
| BALE | 20XXXXXX<br>33XXXXXX | Branch If A Is Less Than Or Equal To Memory | | 4-101 |
| BALT | 20XXXXXX<br>27XXXXXX | Branch If A Is Less Than Memory | | 4-101 |
| BANE | 20XXXXXX<br>25XXXXXX | Branch If A Is Not Equal To Memory | | 4-101 |

Instructions Listed By Mnemonic (contd)

| Mnemonic | Octal Code | Name | Notes | Page |
|---|---|---|---|---|
| BASE | 20XXXXXX<br>21XXXXXX | Branch If A Sign Equals Memory Sign | | 4-101 |
| BDA | 72XXXXXX | Branch And Decrement Index A | | 4-67 |
| BDB | 73XXXXXX | Branch And Decrement Index B | | 4-68 |
| BEEQ | 20XXXXXX<br>43XXXXXX | Branch If E Is Equal To Memory | | 4-101 |
| BEGE | 20XXXXXX<br>55XXXXXX | Branch If E Is Greater Than Or Equal To Memory | | 4-101 |
| BEGT | 20XXXXXX<br>51XXXXXX | Branch If E Is Greater Than Memory | | 4-101 |
| BELE | 20XXXXXX<br>53XXXXXX | Branch If E Is Less Than Or Equal To Memory | | 4-101 |
| BELT | 20XXXXXX<br>47XXXXXX | Branch If E Is Less Than Memory | | 4-101 |
| BENE | 20XXXXXX<br>45XXXXXX | Branch If E Is Not Equal To Memory | | 4-101 |
| BESE | 20XXXXXX<br>41XXXXXX | Branch If E Sign Equals Memory Sign | | 4-101 |
| BIT | 07XXXXXX<br>XXXXXXXX | Bit Manipulation | 2 | 4-103 |
| BIT | 07XXXXXX<br>01XXXXXX | Leave SB Unchanged And Continue (NOP) | 1 | 4-103 |
| BLEQ | 20XXXXXX<br>63XXXXXX | Branch If A, E Is Equal To Memory | | 4-101 |
| BLGE | 20XXXXXX<br>75XXXXXX | Branch If A, E Is Greater Than Or Equal To Memory | | 4-101 |

Instructions Listed By Mnemonic (contd)

| Mnemonic | Octal Code | Name | Notes | Page |
|---|---|---|---|---|
| BLGT | 20XXXXXX<br>71XXXXXX | Branch If A, E Is Greater Than Memory | | 4-101 |
| BLLE | 20XXXXXX<br>73XXXXXX | Branch If A, E Is Less Than Or Equal To Memory | | 4-101 |
| BLLT | 20XXXXXX<br>67XXXXXX | Branch If A, E Is Less Than Memory | | 4-101 |
| BLNE | 20XXXXXX<br>65XXXXXX | Branch If A, E Is Not Equal To Memory | | 4-101 |
| BLSE | 20XXXXXX<br>61XXXXXX | Branch If A, E Sign Equals Memory Sign | | 4-101 |
| BRN | 23XXXXXX | Branch On Negative A Register | | 4-70 |
| BRU | 22XXXXXX | Branch Unconditionally | | 4-73 |
| BRZ | 24XXXXXX | Branch On Zero A Register | | 4-72 |
| BSP | 26XXXXXX | Branch And Save Place | | 4-75 |
| BSR | 25XXXXXX<br>XXXXXXXX | Branch And Save Region | | 4-81 |
| BYT | 06XXXXXX | Byte Manipulation | | 4-108 |
| BZEQ | 20XXXXXX<br>03XXXXXX | Branch If Zero Is Equal To Memory | | 4-101 |
| BZGE | 20XXXXXX<br>15XXXXXX | Branch If Zero Is Greater Than Or Equal To Memory | | 4-101 |
| BZGT | 20XXXXXX<br>11XXXXXX | Branch If Zero Is Greater Than Memory | | 4-101 |
| BZLE | 20XXXXXX<br>13XXXXXX | Branch If Zero Is Less Than Or Equal To Memory | | 4-101 |

Instructions Listed By Mnemonic (contd)

| Mnemonic | Octal Code | Name | Notes | Page |
|---|---|---|---|---|
| BZLT | 20XXXXXX<br>07XXXXXX | Branch If Zero Is Less Than Memory | | 4-101 |
| BZNE | 20XXXXXX<br>05XXXXXX | Branch If Zero Is Not Equal To Memory | | 4-101 |
| BZSE | 20XXXXXX<br>01XXXXXX | Branch If Zero Sign Equals Memory Sign | | 4-101 |
| CBIT | 07XXXXXX<br>02XXXXXX | Change SB (Alternate) And Continue | 1 | 4-107 |
| CIO | 212000XX | Initiate Channel IO | 4 | 4-132 |
| CIOSK | 212004XX | Initiate Channel IO And Skip | 4 | 4-132 |
| CWM | 20XXXXXX<br>XXXXXXXX | Compare With Memory | 2 | 4-97 |
| CXA | 66XXXXXX | Compare Index A And Memory | | 4-62 |
| CXB | 67XXXXXX | Compare Index B And Memory | | 4-62 |
| DEM | 56XXXXXX | Decrement Memory | | 4-116 |
| DIV | 15XXXXXX | Fixed Point Divide | | 4-38 |
| EAM | 54XXXXXX | Exchange A And Memory | | 4-10 |
| FAL | 34XXXXXX | Floating Point Add Long | | 4-49 |
| FAS | 30XXXXXX | Floating Point Add Short | | 4-48 |
| FDL | 37XXXXXX | Floating Point Divide Long | | 4-53 |
| FDS | 33XXXXXX | Floating Point Divide Short | | 4-53 |
| FML | 36XXXXXX | Floating Point Multiply Long | | 4-52 |

Instructions Listed By Mnemonic (contd)

| Mnemonic | Octal Code | Name | Notes | Page |
|---|---|---|---|---|
| FMS | 32XXXXXX | Floating Point Multiply Short | | 4-51 |
| FSL | 35XXXXXX | Floating Point Subtract Long | | 4-50 |
| FSS | 31XXXXXX | Floating Point Subtract Short | | 4-50 |
| GEA | 01XXXXXX | Generate Effective Address | | 4-118 |
| HLT | 00XXXXXX | Halt | | 4-120 |
| IOR | 03XXXXXX | Inclusive OR | | 4-12 |
| LDA | 53XXXXXX | Load A Register | | 4-5 |
| LDE | 52XXXXXX | Load E Register | | 4-5 |
| LDL | 51XXXXXX | Load Long | | 4-6 |
| LLC | 04XXXXXX | Load Logical Complement | | 4-7 |
| LLL | 432X17XX | Shift Left Long-Logical | 4 | 4-26 |
| LLS | 432X15XX | Shift Left Short-Logical | 4 | 4-23 |
| LRL | 432X16XX | Shift Right Long-Logical | 4 | 4-24 |
| LRS | 432X14XX | Shift Right Short-Logical | 4 | 4-21 |
| LXA | 60XXXXXX | Load Index A | | 4-59 |
| LXB | 61XXXXXX | Load Index B | | 4-59 |
| MOV | 50XXXXXX<br>XXXXXXXX | Multiple Move | | 4-116 |
| MPY | 14XXXXXX | Fixed Point Multiply | | 4-40 |
| MST | 55XXXXXX | Masked Store | | 4-9 |

Instructions Listed By Mnemonic (contd)

| Mnemonic | Octal Code | Name | Notes | Page |
|---|---|---|---|---|
| NML | 42XXXXXX | Normalize Long | | 4-29 |
| NMS | 41XXXXXX | Normalize Short | | 4-26 |
| NOP | 22400001 | No Operation | | 4-120 |
| PIO | 21XXXXXX | Programmed Input Output | 2 | 4-121 |
| RBIT | 07XXXXXX<br>00XXXXXX | Reset SB and Continue | 1 | 4-107 |
| RDA | 212030XX | Read Data | 4 | 4-131 |
| RDASK | 212034XX | Read Data And Skip | 4 | 4-131 |
| RFI | 16XXXXXX | Return From Interrupt | | 4-95 |
| RFS | 22200000 | Return From Subroutine | | 4-91 |
| RILS | 212XXXXX | Read Interrupt Level Status | 3, 4 | 4-133 |
| RLE | 40XXXXXX | Rotate Left E Register | | 4-16 |
| RLL | 432XX3XX | Rotate Left Long | 4 | 4-20 |
| RLS | 432XX1XX | Rotate Left Short | 4 | 4-19 |
| RRL | 432XX2XX | Rotate Right Long | 4 | 4-19 |
| RRS | 432XX0XX | Rotate Right Short | 4 | 4-18 |
| RST | 212070XX | Read Status | 4 | 4-127 |
| RSTC | 212130XX | Read Status And Clear | 4 | 4-128 |
| RSTCSK | 212134XX | Read Status And Clear And Skip | 4 | 4-128 |

## Instructions Listed By Mnemonic (contd)

| Mnemonic | Octal Code | Name | Notes | Page |
|---|---|---|---|---|
| SAEQ | 20XXXXXX<br>22XXXXXX | Skip If A Is Equal To Memory | | 4-101 |
| SAGE | 20XXXXXX<br>34XXXXXX | Skip If A Is Greater Than Or Equal To Memory | | 4-101 |
| SAGT | 20XXXXXX<br>30XXXXXX | Skip If A Is Greater Than Memory | | 4-101 |
| SALE | 30XXXXXX<br>32XXXXXX | Skip If A Is Less Than Or Equal To Memory | | 4-101 |
| SALT | 20XXXXXX<br>26XXXXXX | Skip If A Is Less Than Memory | | 4-101 |
| SANE | 20XXXXXX<br>24XXXXXX | Skip If A Is Not Equal To Memory | | 4-101 |
| SASE | 20XXXXXX<br>20XXXXXX | Skip If A Sign Equals Memory Sign | | 4-101 |
| SBIT | 07XXXXXX<br>03XXXXXX | Set SB And Continue | 1 | 4-107 |
| SBL | 13XXXXXX | Fixed Point Subtract Long | | 4-36 |
| SEEQ | 20XXXXXX<br>42XXXXXX | Skip If E Is Equal To Memory | | 4-101 |
| SEGE | 20XXXXXX<br>54XXXXXX | Skip If E Is Greater Than Or Equal To Memory | | 4-101 |
| SEGT | 20XXXXXX<br>50XXXXXX | Skip If E Is Greater Than Memory | | 4-101 |
| SELE | 20XXXXXX<br>52XXXXXX | Skip If E Is Less Than Or Equal To Memory | | 4-101 |
| SELT | 20XXXXXX<br>46XXXXXX | Skip If E Is Less Than Memory | | 4-101 |

Instructions Listed By Mnemonic (contd)

| Mnemonic | Octal Code | Name | Notes | Page |
|---|---|---|---|---|
| SENE | 20XXXXXX<br>44XXXXXX | Skip If E Is Not Equal To Memory | | 4-101 |
| SESE | 20XXXXXX<br>40XXXXXX | Skip If E Sign Is Equal To Memory Sign | | 4-101 |
| SHF | 43XXXXXX | Shift And Rotate | 2 | 4-17 |
| SKR | 07XXXXXX<br>11XXXXXX | Skip If SB Is Reset And Leave SB Unchanged | 1 | 4-107 |
| SKRC | 07XXXXXX<br>12XXXXXX | Skip If SB Is Reset And Alternate SB | 1 | 4-107 |
| SKRR | 07XXXXXX<br>10XXXXXX | Skip If SB Is Reset And Reset SB | 1 | 4-107 |
| SKRS | 07XXXXXX<br>13XXXXXX | Skip If SB Is Reset And Set SB | 1 | 4-107 |
| SKS | 07XXXXXX<br>05XXXXXX | Skip If SB Is Set And Leave SB Unchanged | 1 | 4-107 |
| SKSC | 07XXXXXX<br>06XXXXXX | Skip If SB Is Set And Alternate SB | 1 | 4-107 |
| SKSR | 07XXXXXX<br>04XXXXXX | Skip If SB Is Set Reset Bit | 1 | 4-107 |
| SKSS | 07XXXXXX<br>07XXXXXX | Skip If SB Is Set And Set SB | 1 | 4-107 |
| SKU | 07XXXXXX<br>15XXXXXX | Skip Unconditionally | | 4-107 |
| SKUC | 07XXXXXX<br>16XXXXXX | Alternate SB And Skip Unconditionally | 1 | 4-107 |
| SKUR | 07XXXXXX<br>14XXXXXX | Skip And Reset SB | 1 | 4-107 |

Instructions Listed By Mnemonic (contd)

| Mnemonic | Octal Code | Name | Notes | Page |
|---|---|---|---|---|
| SKUS | 07XXXXXX<br>17 | Set SB And Skip Unconditionally | 1 | 4-107 |
| SLEQ | 20XXXXXX<br>62XXXXXX | Skip If A, E Is Equal To Memory | | 4-101 |
| SLGE | 20XXXXXX<br>74XXXXXX | Skip If A, E Is Greater Than Or Equal To Memory | | 4-101 |
| SLGT | 20XXXXXX<br>70XXXXXX | Skip If A, E, Is Greater Than Memory | | 4-101 |
| SLLE | 20XXXXXX<br>72XXXXXX | Skip If A, E Is Less Than Or Equal To Memory | | 4-101 |
| SLLT | 20XXXXXX<br>66XXXXXX | Skip If A, E, Is Less Than Memory | | 4-101 |
| SLNE | 20XXXXXX<br>64XXXXXX | Skip If A, E, Is Not Equal To Memory | | 4-101 |
| SLSE | 20XXXXXX<br>60XXXXXX | Skip If A, E Sign Equals Memory Sign | | 4-101 |
| SNR | 47XXXXXX | Store Normalized And Rounded | | 4-54 |
| SPL | 17XXXXXX | Set Priviledge And Level Register | | 4-119 |
| STA | 44XXXXXX | Store A Register | | 4-8 |
| STE | 46XXXXXX | Store E Register | | 4-9 |
| STL | 45XXXXXX | Store Long | | 4-9 |
| SUB | 12XXXXXX | Fixed Point Subtract Short | | 4-35 |
| SXA | 62XXXXXX | Store Index A | | 4-60 |
| SXB | 63XXXXXX | Store Index B | | 4-61 |

Instructions Listed By Mnemonic (contd)

| Mnemonic | Octal Code | Name | Notes | Page |
|---|---|---|---|---|
| SZEQ | 20XXXXXX<br>02XXXXXX | Skip If Zero Is Equal To Memory | | 4-101 |
| SZGE | 20XXXXXX<br>14XXXXXX | Skip If Zero Is Greater Than Or Equal To Memory | | 4-101 |
| SZGT | 20XXXXXX<br>10XXXXXX | Skip If Zero Is Greater Than Memory | | 4-101 |
| SZLE | 20XXXXXX<br>12XXXXXX | Skip If Zero Is Less Than Or Equal To Memory | | 4-101 |
| SZLT | 20XXXXXX<br>06XXXXXX | Skip If Zero Is Less Than Memory | | 4-101 |
| SZNE | 20XXXXXX<br>04XXXXXX | Skip If Zero Is Not Equal To Memory | | 4-101 |
| SZSE | 20XXXXXX<br>00XXXXXX | Skip If Zero Sign Equals Memory Sign | | 4-101 |
| TIA | 70XXXXXX | Test And Increment Index A | | 4-63 |
| TIB | 71XXXXXX | Test And Increment Index B | | 4-65 |
| TWBA | 20XXXXXX<br>36XXXXXX | Three-Way Branch On A Minus Memory | | 4-101 |
| TWBE | 20XXXXXX<br>56XXXXXX | Three-Way Branch On E Minus Memory | | 4-101 |
| TWBL | 20XXXXXX<br>76XXXXXX | Three-Way Branch On A, E Minus Memory | | 4-101 |
| TWBZ | 20XXXXXX<br>16XXXXXX | Three-Way Branch On Zero Minus Memory | | 4-101 |
| WDA | 212010XX | Write Data | 4 | 4-130 |
| WDASK | 212014XX | Write Data And Skip | 4 | 4-130 |

Instructions Listed By Mnemonic (contd)

| Mnemonic | Octal Code | Name | Notes | Page |
|---|---|---|---|---|
| WST | 212050XX | Write Status | 4 | 4-129 |
| XOR | 05XXXXXX | Exclusive OR | | 4-13 |

NOTES:

1. SB = Sensed Bit.
2. Basic instruction never used in this form; always used in microcoded (extended mnemonic) form.
3. Device address field contains interrupt level.
4. Octal code listed assumes literal mode.

# C/PROGRAMMING EXAMPLES

## PROGRAMMING FOR CHANNEL IO

Programming to accomplish channel IO operation must take into consideration the requirements of sequence instruction processing and the nature of the program that initiates channel IO.

Most programs are relocatable, and the storage locations of both instructions and data are not known until program loading time. Several absolute addresses are required for channel IO, and in a relocatable program these must be generated during program execution. The Generate Effective Address (GEA) instruction is used for this purpose. Among the addresses which often require generation are:

a. The First Word Address, which must be in word 3 of each Channel Order Triple Word (COTW) when channel IO is activated.

b. The Chaining Link Address, which point to subsequent Channel Order Triple Words if chaining is used.

c. The address of the first word of the first COTW must be stored in a memory location to allow CIO to access COTW.

d. The address in NXAR, which specifies the location containing the address of the first COTW.

A PIO instruction is used to start channel IO. This instruction may be written with SKP = 1, if the programmer wishes to skip if the device is not busy.

The following example shows a sequence of instructions for setting up and initiating channel IO operation.

Example:

A program has its origin at location α. One of its functions is to transmit two blocks of data by channel IO on channel 6. The first COTW is at location α + 200, and the second is at α + 204. The two blocks of data are located in common storage at relative locations XC + $100_8$ and XC + $250_8$. The first word of common is used as temporary storage for the address of the first COTW. Assume a device address of $100_8$.

| Loc. | Op | M | I | XB | XA | Y<br>or<br>DISP | Interpretation |
|---|---|---|---|---|---|---|---|
| | | | | (bits) | | | |
| α<br>•<br>•<br>• | | | | | | | |
| α + 20 | GEA | 5 | 0 | 0 | 0 | 0100 | Generate First Word Address of First Data Block. |
| α + 21 | STE | 4 | 0 | 0 | 0 | 0161<br>α + 202 - (α + 21) | Establish First Word Address in First COTW |
| α + 22 | GEA | 5 | 0 | 0 | 0 | 0250 | Generate First Word Address of Second Data Block |
| α + 23 | STE | 4 | 0 | 0 | 0 | 0163<br>α + 206 - (α + 23) | Establish First Word Address of Second COTW |
| α + 24 | GEA | 4 | 0 | 0 | 0 | 0160<br>α + 204 - (α + 24) | Generate Address of Second COTW |
| α + 25 | STE | 4 | 0 | 0 | 0 | 0156<br>α + 203 - (α + 25) | Store into Link of First COTW |

| Loc. | Op | M | I | XB | XA | Y<br>or<br>DISP | Interpretation |
|---|---|---|---|---|---|---|---|
| | | | (bits) | | | | |
| α + 26 | GEA | 4 | 0 | 0 | 0 | 0152 | Generate Address of First COTW |
| α + 27 | STE | 5 | 0 | 0 | 0 | 0000 | Store Address of First COTW into COMMON |
| α + 30 | GEA | 5 | 0 | 0 | 0 | 0000 | Generate Pointer to Address of First COTW |
| α + 31 | STE | 0 | 0 | 0 | 0 | 0056 | Store Indirect Address in NXAR6 |
| α + 32 | CIO | 2 | 0 | 0 | 0 | 00500 | Output Command, DTI = 0, ACU = 0, SKP = 1, Device Address = 100 |
| α + 33 | BRU | 4 | 0 | 0 | 0 | 7777 | Skipped when PIO Acknowledged. If rejected, branch back to PIO |

•
•
•

# D/ARITHMETIC NOTATION

## TWO'S COMPLEMENT NOTATION

The 2's complement of a binary number is the result of subtracting the binary number, including its sign bit, from the binary quantity equal to $2^n$, where n is the number of binary digits in the number to be complemented. For example, to compute the 2's complement of a 24-bit word, the 24 bits are subtracted from $2^{24}$.

```
        2^24 = 1000000000000000000000000
  Binary No. = -010111000111010011101001
  2's Comp.  = X101000111000101100010111
```

Another method of arriving at the same result is to reverse all bits in the word to be complemented (1's complement) and then add a 1 bit at the least significant bit position (bit 23).

```
  Binary No. = 010111000111010000000000
  1's Comp.  = 101000111000101111111111
  Add 1                              +1
  2's Comp.  = 101000111000110000000000
```

An easier method is to reverse all bits in the word, starting with bit position 00, down to (but not including) the least significant (rightmost) 1 bit. The remaining bits are unchanged.

```
  Binary No. = 111000110101000001000000
                \_______\/_______/ \_\/_/

                     Reverse   Unchanged
  2's Comp.  = 000111001010111111000000
```

The 2's complement of a stored number can be accomplished by two successive instructions:

a. Load Logical Complement (LLC)

b. Add To A Register (ADD) in which the addend is +1; i.e., 10200001.

## FIXED POINT NOTATION

All fixed point operands are treated as signed integers. Positive numbers are represented in true binary notation with the sign bit set to zero. Negative numbers are represented in 2's complement notation with a one in the sign bit. The 2's complement representation of a negative number may be considered the sum of the integer part of the field, taken as a positive number and the maximum negative number.

This type of number representation can be considered the low-order portion of an infinitely long representation of the number. When the number is positive, all bits to the left of the most significant bit of the number, including the sign bit, are zeros. When the number is negative, all these bits, including the sign bit, are ones. Therefore, when an operand must be extended with high-order bits, the expansion is achieved by prefixing a field in which each bit is set equal to the high-order bit of the operand.

Two's complement notation does not include a negative zero. It has a number range in which the set of negative numbers is one larger than the set of positive numbers. The maximum positive number consists of an all-one integer field with a sign bit of zero, whereas the maximum negative number (the negative number with the greatest absolute value) consists of an all-zero integer field with a one-bit for sign.

The CP cannot represent the complement of the maximum negative number. When an operation, such as a subtraction from zero, produces the complement of the maximum negative number, the number remains unchanged, and a fixed point overflow condition is recognized. An overflow does not result, however, when the number is complemented and the final result is within the representable range. An example of this case is a subtraction from minus one. The product of two maximum negative numbers is representable as a double-length positive number.

The sign bit is left-most in a number. In an arithmetic operation, a carry out of the integer field changes the sign. However, in algebraic left-shifting, the sign bit does not change even if significant high-order bits are shifted out of the integer field.

Fixed point operands may be recorded in single-word (24 bits) or double-word (48 bits) lengths. In both lengths, the first bit position (00) holds the sign of the number, with the remaining bit positions (01-23 for single words and 01-47 for double words) used to designate the magnitude of the number.

Positive fixed point numbers are represented in true binary form with a zero sign bit. Negative fixed point numbers are represented in 2's complement notation with a one bit in the sign position. In all cases, the bits between the sign bit and the left-most significant bit of the integer are the same as the sign bit (i.e., all zeros for positive numbers and all ones for negative numbers).

This notation is called fixed point because the programmer determines the fixed positioning of the binary point.

## FLOATING POINT NOTATION

Floating point arithmetic simplifies programming by automatically maintaining binary point placement (scaling) during computations in which the ranges of values used vary widely or are unpredictable. In floating point notation, each number is represented in two parts: one indicating the significant digits of the number, and the second indicating the size (scale) of the number. Thus, the number is expressed as a fraction times a power of 2.

A floating point number has two associated sets of values. One set represents the significant digits of the number and is called the fraction. The second set specifies the power (exponent) to which 2 is raised and indicates the location of the binary point of the number.

These two numbers (the fraction and exponent) are recorded in single-word or double-word format.

Since each of these two numbers is signed, some method must be employed to express two signs in an area that provides for a single sign. This is accomplished by having the fraction sign use the sign associated with the word (or double word) and expressing the exponent in excess 32 (single-word) or excess 2048 (double-word) arithmetic; that is, the exponent is added as a signed number to 32 or 2048. The resulting number is called the characteristic.

Both long format and short format use a sign bit for the fraction in bit position 00, followed by a signed 6-bit or 12-bit characteristic, followed by a 16-bit or 35-bit fraction.

Features of floating-point numbers as used in the FOX 1 System are:

a.1 True zero is a positive floating point number with a fraction of zero and a zero characteristic; i.e., floating point zero looks like 0.0.

a.2 Abnormal zero is a floating point number with a fraction of zero and a non-zero characteristic.

Use of abnormal zero may erroneously cause FPO on multiply or divide and may cause loss of accuracy on add or subtract.

b. A positive floating point number is normalized if, and only if, the fraction is in the interval $1/2 \leq F < 1$. This indicates that the high-order bit of the fraction portion will always be a 1 bit if the number is normalized.

c. Negative floating point numbers are the 2's complement of the positive representation of the number. This 2's complement involves both the fraction and characteristic.

d. Negative floating point numbers are normalized only if the positive representation of the number is normalized.

e. The characteristic of a floating point number (excess 32 in the short form; excess 2048 in the long form) is a function of the magnitude of the fraction and of the sign of the fraction.

f. The most significant advantage achieved by the use of 2's complement notation and excess characteristics in representing floating point numbers is that the numbers may be compared using fixed point instructions such as CWM (Compare With Memory). This only applies, however, if the numbers are normalized and zero is represented as true zero. This feature can be seen by examining the values in Table D-1.

g. All floating point arithmetic is done in the long (double-precision) format. Short floating point numbers are stretched to the long form during execution of arithmetic instructions and the result is in the long form. The sole advantage of using the short (single-precision) format is that less memory is required. Using the double-precision (long) format allows more precision, a larger exponent range, and less instruction execution time.

## Table D-1. Floating Point Number Ranges

| BINARY VALUE | APPROX.[1] DECIMAL VALUE | OCTAL REPRESENTATION | NOTES | BINARY VALUE | APPROX. DECIMAL VALUE | OCTAL REPRESENTATION | NOTES |
|---|---|---|---|---|---|---|---|
| $+(1-2^{-35})2^{2047}$ | 1.76D628<br>1.76E628 | 3777777777777777<br>Overflow | LPD[2] | $-2^{-2049}$ | -1.40D-629<br>-1.40E-629 | 7777600000000000<br>Underflow | SND[6] |
| $+(1-2^{-35})2^{2046}$ | 8.83D627<br>8.83E627 | 3777377777777777<br>Overflow | | $-2^{-2048}$ | -2.79D-629<br>-2.79E-629 | 7777200000000000<br>Underflow | |
| $+(1-2^{-17})2^{31}$ | 2.15D9<br>2.15E9 | 2017777777000000<br>37777777 | LPS[3] | $-2^{-34}$ | -5.82D-11<br>-5.82E-11 | 6020200000000000<br>Underflow | |
| $+(1-2^{-17})2^{30}$ | 1.07D9<br>1.07E9 | 2017377777000000<br>37377777 | | $-2^{-33}$ | -1.16D-10<br>-1.16E-10 | 6017600000000000<br>77600000 | SNS[7] |
| +13/4 | 3.25D0<br>3.25E0 | 2001320000000000<br>21320000 | | -5/64 | -7.81D-2<br>-7.81E-2 | 6001140000000000<br>61140000 | |
| +3/2 | 1.5D0<br>1.5E0 | 2000700000000000<br>20700000 | | -5/16 | -3.125D-1<br>-3.125E-1 | 6000140000000000<br>60140000 | |
| +1/2 | 0.5D0<br>0.5E0 | 2000200000000000<br>20200000 | | -1/2 | -0.5D0<br>-0.5E0 | 5777600000000000<br>57600000 | |
| +5/16 | 3.125D-1<br>3.125E-1 | 1777640000000000<br>17640000 | | -3/2 | -1.5D0<br>-1.5E0 | 5777100000000000<br>57100000 | |
| +5/64 | 7.81D-2<br>7.81E-2 | 1776640000000000<br>16640000 | | -13/4 | -3.25D0<br>-3.25E0 | 5776460000000000<br>56460000 | |
| $+2^{-33}$ | 1.16D-10<br>1.16E-10 | 1760200000000000<br>00200000 | SPS[4] | $-(1-2^{-17})2^{30}$ | -1.07D9<br>-1.07E9 | 5760400001000000<br>40400001 | |
| $+2^{-34}$ | 5.82D-11<br>5.82E-11 | 1757600000000000<br>Underflow | | $-(1-2^{-17})2^{31}$ | -2.15D9<br>-2.15E9 | 5760000001000000<br>40000001 | LNS[8] |
| $+2^{-2048}$ | 2.79D-629<br>2.79E-629 | 0000600000000000<br>Underflow | | $-(1-2^{-35})2^{2046}$ | -8.83D627<br>-8.83E627 | 4000400000000001<br>Overflow | |
| $+2^{-2049}$ | 1.40D-629<br>1.40E-629 | 0000200000000000<br>Underflow | SPD[5] | $-(1-2^{-35})2^{2047}$ | -1.76D628<br>-1.76E628 | 4000000000000001<br>Overflow | LND[9] |
| 0 | 0.D0<br>0.E0 | 0000000000000000<br>00000000 | | | | | |

[1] Fortran Notation.
[2] Largest Positive Double-Precision Floating-Point Number.
[3] Largest Positive Single-Precision Floating-Point Number.
[4] Smallest Positive Single-Precision Floating-Point Number.
[5] Smallest Positive Double-Precision Floating-Point Number.
[6] Smallest Negative Double-Precision Floating-Point Number.
[7] Smallest Negative Single-Precision Floating-Point Number.
[8] Largest Negative Single-Precision Floating-Point Number.
[9] Largest Negative Double-Precision Floating-Point Number.

# Index

INDEX (contd)

# INDEX (contd)

## INDEX (contd)

INDEX (contd)

# INDEX (contd)

INDEX (contd)

## INDEX (contd)

# INDEX (contd)

INDEX (contd)

# READER'S COMMENT FORM

DOCUMENT TITLE HARDWARE SYS OVERVIEW & REF MAN; VOL. II, COMPUTER REF. MANUAL
NUMBER 73001BA-2

*Please give specific page and line references with your comments when appropriate.*
*If you wish a reply, be sure to include your name and address.*

Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

fold DO NOT STAPLE fold

THE FOXBORO COMPANY
FOXBORO, MASS. 02035

ATTENTION: TECHNICAL WRITING SERVICES, DEPT. 989

fold DO NOT STAPLE fold

73001BA-2
3-72/800

**Digital Systems Division**
**THE FOXBORO COMPANY** Foxboro, Massachusetts, U.S.A. 02035
Printed in U.S.A.