

17329120



**MP-60 EMULATION
REFERENCE MANUAL**

**CONTROL DATA®
MP-32
COMPUTER SYSTEMS**

REVISION RECORD

| REVISION | DESCRIPTION |
|--------------------------|--|
| A (02/01/78) | Original release. |
| B (08/15/80) | Incorporate new instructions. |
| C (03/11/83) | Correct typographical errors for customer release. |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| Document No. 17329120 | |

Revision letters I, O, Q, and X are not used.

Please address comments concerning this manual to:

©COPYRIGHT CONTROL DATA CORPORATION
 1978, 1980, 1983
 All Rights Reserved
 Printed in the United States of America

CONTROL DATA CORPORATION
 SYSTEMS TECHNOLOGY DIVISION
 215 Moffett Park Drive
 Sunnyvale, California 94086

Or use the Comment Sheet in the back of this manual.

LIST OF EFFECTIVE PAGES

| Page | Revision |
|-------------------|----------|
| Front Cover | - |
| ii/iii | C |
| v | B |
| vii | B |
| viii | C |
| xi | B |
| 1-1 through 1-7 | A |
| 2-1 through 2-15 | A |
| 3-1 | A |
| 3-2 | B |
| 3-3 | C |
| 3-4 through 3-14 | A |
| 4-1 through 4-11 | A |
| 4-12 | B |
| 5-1 through 12 | A |
| 6-1 | C |
| 6-2 | A |
| 6-3 | C |
| 6-4 through 6-8 | A |
| 6-9 | B |
| 6-10 through 6-13 | A |
| 6-14 through 6-45 | B |
| 6-46 | C |
| 6-47 | B |
| 6-48 | C |
| 6-49 through 6-57 | B |
| 6-58 | C |
| 6-59 through 6-79 | B |
| 6-80 | C |
| 6-81/6-82 | B |
| 6-83 | C |
| 6-84 through 6-86 | B |
| 6-87 | C |
| 6-88 | B |
| 6-89 | C |
| 6-90 through 6-96 | B |
| A-1 | A |
| B-1 through B-7 | A |
| C-1 | A |

| Page | Revision |
|------------------|----------|
| D-1 through D-4 | A |
| E-1/E-2 | A |
| E-3 through E-10 | C |
| Comment Sheet | C |



PREFACE

This manual provides information for the machine language use of the MP-60 Emulated computer system. Its intention is to describe the capabilities and programming restraints of the hardware.

COMPASS mnemonics are used to abbreviate titles of instructions; however, no software systems are used in describing instructions. Detailed descriptions for those systems in operation are available in the appropriate software reference manuals.



CONTENTS

| <u>Section</u> | <u>Title</u> | <u>Page</u> |
|----------------|---------------------------------|-------------|
| 1 | SYSTEM DESCRIPTION | 1-1 |
| | System Modularity | 1-2 |
| | CPU Module | 1-2 |
| | I/O Module | 1-2 |
| | Memory Module | 1-2 |
| | Console | 1-5 |
| | Computer Organization | 1-5 |
| | Computer Word Format | 1-5 |
| | Register Files | 1-5 |
| | P Register | 1-6 |
| | I Register | 1-6 |
| | M Registers | 1-6 |
| | Bit Register | 1-6 |
| | Relocation Register | 1-6 |
| | Real-Time Clock | 1-7 |
| | Parity | 1-7 |
| 2 | INPUT/OUTPUT CHARACTERISTICS | 2-1 |
| | Internal Input/Output Interface | 2-1 |
| | Register Input/Output | 2-2 |
| | Addressing - In/Out Protocol | 2-3 |
| | Data Transfer - In/Out | 2-4 |
| | Director Functions - In/Out | 2-4 |
| | Director Status - In/Out | 2-7 |
| | Automatic Data Transfer | 2-12 |
| | Direct Memory Access | 2-15 |

CONTENTS (Cont'd)

| <u>Section</u> | <u>Title</u> | <u>Page</u> |
|----------------|-----------------------------------|-------------|
| 3 | EXCHANGE PACKAGE/INTERRUPT SYSTEM | 3-1 |
| | Exchange Package Area | 3-1 |
| | Interrupt System | 3-4 |
| | Internal Interrupts | 3-4 |
| | Arithmetic Overflow Fault | 3-4 |
| | Function Fault | 3-5 |
| | Exponent Fault | 3-5 |
| | Divide Fault | 3-5 |
| | Illegal Instruction | 3-6 |
| | Page Fault 1 | 3-6 |
| | Page Fault 2 | 3-6 |
| | Page Fault 3 | 3-6 |
| | Memory Parity Error | 3-6 |
| | Memory Reject | 3-6 |
| | Power Failure | 3-7 |
| | I/O Interrupts | 3-7 |
| | Real-Time Interrupts | 3-7 |
| | Inter-Processing Interrupts | 3-7 |
| | Interrupt Mask Register | 3-8 |
| | Interrupt Control | 3-8 |
| | Interrupt Recognition | 3-8 |
| | Interrupt Priority | 3-9 |
| 4 | DISPLAY CONSOLE | 4-1 |
| | Panel Interface | 4-2 |
| | Functional Operation | 4-5 |
| | Control Functions | 4-5 |
| | H Control Function | 4-5 |
| | I Control Function | 4-5 |

CONTENTS (Cont'd)

| <u>Section</u> | <u>Title</u> | <u>Page</u> |
|----------------|-----------------------------------|-------------|
| | J Control Function | 4-6 |
| | K Control Function | 4-6 |
| | L Control Function | 4-7 |
| | Stop/Go Control | 4-7 |
| | Master Clear | 4-7 |
| | Breakpoint (BP) | 4-8 |
| | Auto Display | 4-9 |
| | Special Considerations | 4-9 |
| | Micro Mode | 4-10 |
| | Macro Mode | 4-10 |
| | MP-60/Panel Interface Displays | 4-12 |
| 5 | MEMORY SYSTEM | 5-1 |
| | Multi-Port Memory | 5-1 |
| | CPU Memory Management Interface | 5-2 |
| | Lookahead Registers | 5-2 |
| | CPU Data Formatting | 5-3 |
| | Paging | 5-3 |
| | Page Index File | 5-4 |
| | Page Index | 5-4 |
| | Program State Register | 5-5 |
| | Memory Address Generation | 5-5 |
| | Memory Errors and Protection | 5-5 |
| | DMA Data Formatting | 5-9 |
| | DMA State Register | 5-9 |
| 6 | INSTRUCTION SET DESCRIPTION | 6-1 |
| | Instruction Formats | 6-1 |
| | Symbol Definitions | 6-4 |
| | Indexing and Address Modification | 6-5 |

CONTENTS (Cont'd)

| <u>Section</u> | <u>Title</u> | <u>Page</u> |
|----------------|---------------------------|-------------|
| | Use of Registers | 6-7 |
| | Number Representation | 6-7 |
| | No Operation | 6-8 |
| | Load | 6-9 |
| | Store | 6-15 |
| | Fixed Point Arithmetic | 6-19 |
| | Floating Point Arithmetic | 6-23 |
| | Shift | 6-28 |
| | Logical | 6-29 |
| | Test | 6-32 |
| | Register Bit Operations | 6-39 |
| | Bit Skips | 6-41 |
| | File Skips | 6-44 |
| | Jumps | 6-46 |
| | Register Operations | 6-47 |
| | Function | 6-63 |
| | Block Transfers | 6-67 |
| | Special Functions | 6-78 |
| | External Functions | 6-84 |
| | Supplementary Functions | 6-93 |

FIGURES

| | | |
|-----|---------------------------------|------|
| 1-1 | Typical MP-60 System | 1-3 |
| 1-2 | MP-60 Emulated CPU Module | 1-4 |
| 2-1 | ADT Table Entry Format | 2-14 |
| 2-2 | Current Memory Address Format | 2-14 |
| 3-1 | Exchange Package Description | 3-2 |
| 4-1 | Function Control Register (FCR) | 4-3 |
| 5-1 | Memory Page Structure | 5-4 |
| 5-2 | Page Index File Addressing | 5-6 |
| 5-3 | Page Index Register | 5-6 |
| 5-4 | Memory Address Generation | 5-7 |

CONTENTS (Cont'd)

| <u>Section</u> | <u>Title</u> | <u>Page</u> |
|----------------|--|-------------|
| 5-5 | Memory Error Table | 5-8 |
| 5-6 | CPU State Register | 5-11 |
| 5-7 | DMA State Register | 5-11 |
| 5-8 | Typical Address Snapshot | 5-12 |
| 6-1 | Word Addressed Instruction Format | 6-1 |
| 6-2 | Half Word Addressed Instruction Format | 6-2 |
| 6-3 | Byte Addressed Instruction Format | 6-2 |
| 6-4 | Bit Addressed Instruction Format | 6-3 |
| 6-5 | Relative Displacement Instruction Format | 6-5 |
| 6-6 | Three Address Instruction Format | 6-4 |

TABLES

| | | |
|-----|---|------|
| 3-1 | Real-Time Interrupt Mask Bit Assignment | 3-10 |
| 3-2 | Interrupt Mask Bit Assignments | 3-11 |
| 3-3 | Low Memory Assignments | 3-12 |
| 3-4 | Interrupt Priority | 3-14 |
| 4-1 | Display Code Definitions | 4-4 |
| 4-2 | MP-60 Register Utilization | 4-13 |

APPENDIXES

| | | |
|---|--|-----|
| A | Powers of Two Table | A-1 |
| B | I/O Programming Formats | B-1 |
| C | Hexadecimal Conversion Tables | C-1 |
| D | IOC Interface | D-1 |
| E | MP-60 Machine Language Instruction Formats | E-1 |



SYSTEM DESCRIPTION

1

The CONTROL DATA® MP-32 is a medium scale, solid state, general purpose digital computer system. Advanced design techniques in the field of microprogrammable architecture are used throughout the system to provide a flexible, extendable, compact and high-performance system for use in scientific, real-time, data management applications. Modular packaging of the basic MP-32 facilitates expansion of the system components to accommodate increased customer needs.

The MP-32 provides a general-purpose computer capability with a basic firmware emulation package. This package utilizes the wealth of hardware resources to present a state-of-the-art 32-bit architecture, termed the MP-60. As such, the MP-60 represents an optimum emulator design using the CDC MP-32 series of architectural components.

The advantages of the MP-60 for the system designer are an instruction set which can be extended to include user-defined functions, large, fast local store and flexible input/output (I/O) structure, interrupt structure and memory address range. Briefly, the MP-60 system provides the following features:

- 32-bit, two's complement single precision arithmetic and 64-bit, two's complement double precision arithmetic.
- Eight sets of register files, 32 registers per set.
- Fixed point integer and floating point arithmetic.
- Double word, full word, half word, byte and bit addressing modes.
- Read/write control store containing the MP-60 microprogram (firmware).
- Extendable instruction set with the ability to define unique instruction sets for new applications.

- Features to allow expansion to multiprocessor, multiprogramming systems.
- Addressing capability to 16 million bytes.

SYSTEM MODULARITY

A basic MP-60 system consists of a central processor unit (CPU) with I/O controllers (IOC), one 65K bank of memory with two ports and an operators console. Figure 1-1 shows a typical MP-60 system with a possible expansion in dotted lines. Memory may be increased to a maximum of sixteen (16) chassis, each chassis containing 256K words. In addition, each of these banks may be equipped with up to eight ports. It is also possible to create other configurations by the addition of one or more CPUs in a multi-computer environment.

CPU MODULE

The MP-60 emulated CPU module, Figure 1-2, consists of the arithmetic, control, control memory, I/O and memory interface hardware. The CPU is a 32-bit, file organized, two's complement, single precision integer processor.

I/O MODULE

The I/O module consists of the available peripheral controllers and interface cards. These cards are normally inserted in open slots, located in the CPU chassis.

MEMORY MODULE

The Multi-Port Memory (MPM) is a modular semiconductor memory capable of expanding the number of banks, access ports and memory chassis. A bank of memory consists of either 32K, or 65K words, and a memory chassis may contain from 1 to 4 banks. The number of access ports per chassis may vary between two and eight in increments of two. The maximum number of memory chassis a single CPU may be connected to is sixteen (16); therefore, the maximum amount of memory a single CPU may reference is four million words.

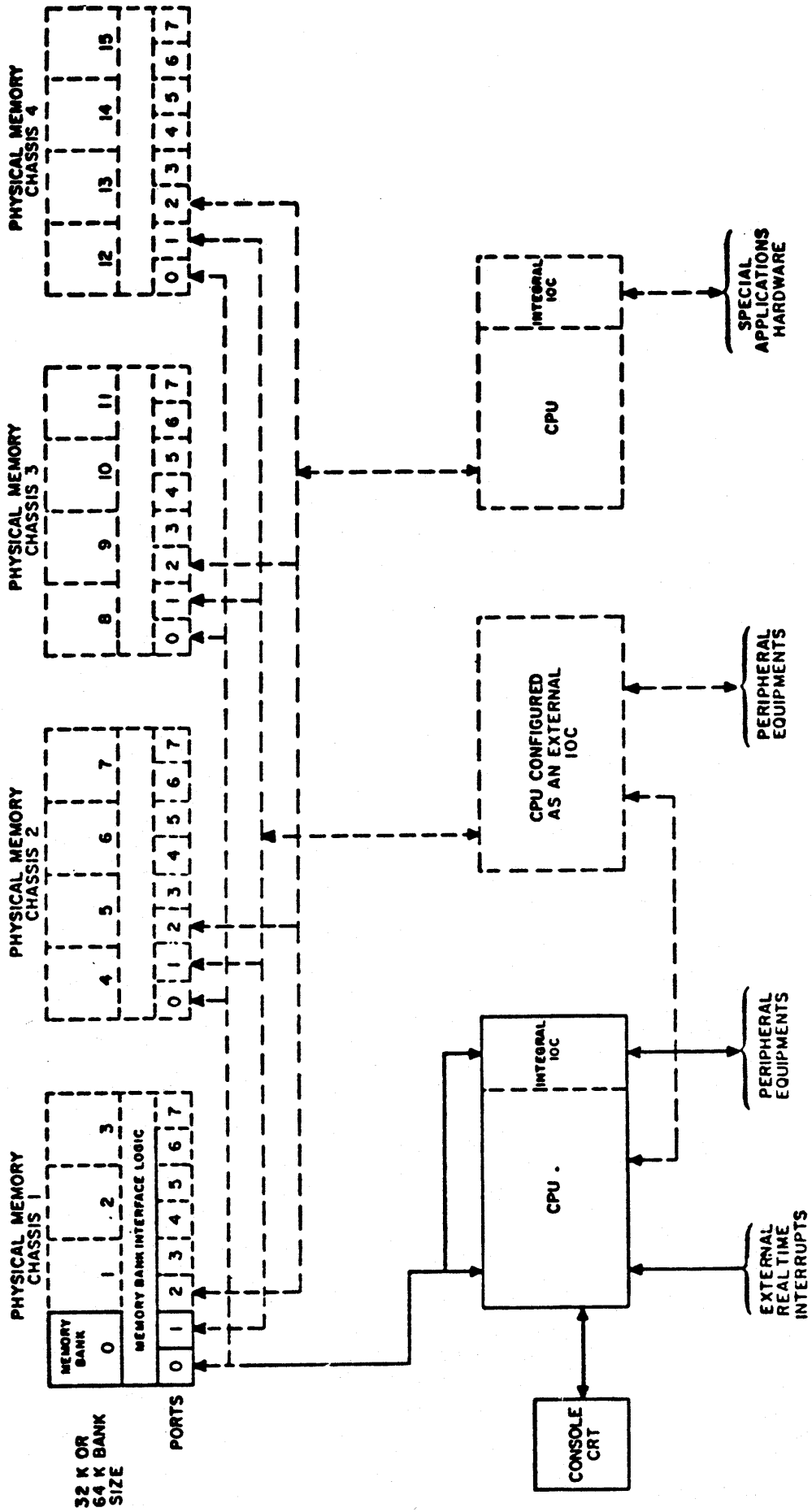


Figure 1-1. Typical MP-60 System

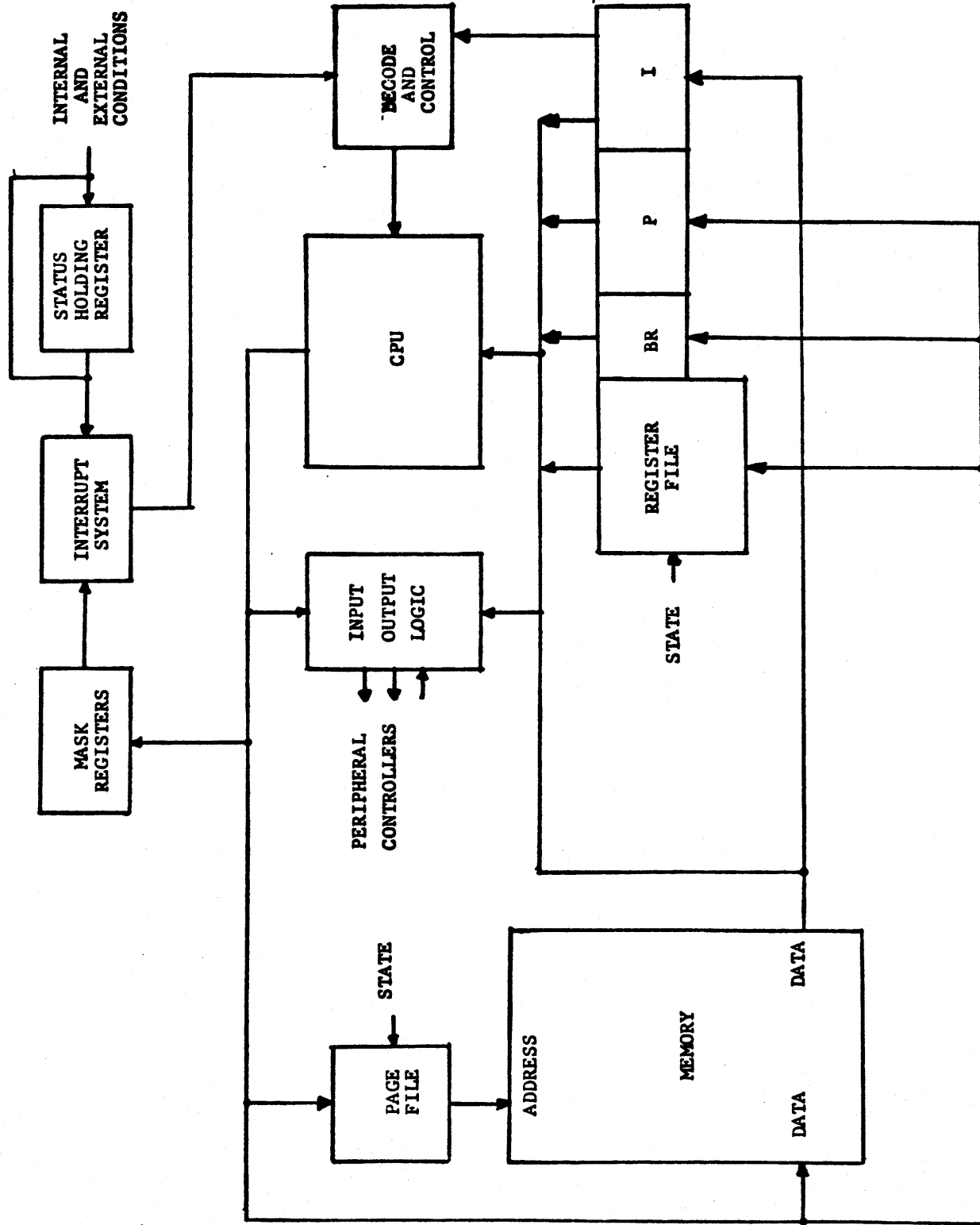


Figure 1-2. MP-60 Emulated CPU Module

CONSOLE

The MP-60 is equipped with an operator console. The operator console is a CDC[®] CRT display, which includes a keyboard, a CRT and a controller, all self-contained.

COMPUTER ORGANIZATION

COMPUTER WORD FORMAT

The MP-60 computer word consists of 32 binary digits. Each word is divided into four 8-bit bytes. An odd parity bit is generated in storage and checked for each of the four bytes, lengthening the storage word to 36 bits. Figure 1-3 illustrates the bit assignments of a computer word in storage.

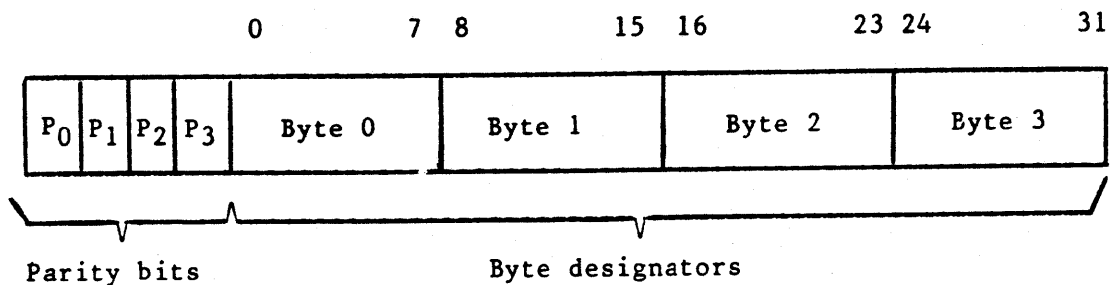


Figure 1-3. Computer Word Format

REGISTER FILES

The MP-60 contains eight sets of register files. Each file consists of 32 general-purpose registers which may be used as accumulators or indexes. The program state setting determines which file is addressed by an instruction. A register within a register file can only be accessed while executing in its associated program state.

The registers in any given register file are numbered sequentially from 0 through 31 and are specified by various fields in the instruction formats. For some operations, two adjacent registers are coupled together, providing a

double register capacity. In this type of instruction, the referenced register contains the most significant portion of the operand and the next higher addressed register contains the least significant portion. Any register, with the exception of register 31, may be referenced as the first register in a double register instruction.

P REGISTER

The 32-bit P register is the program address counter. Only the lower order 16 bits of P are used for program addressing (bits 16 through 31).

I REGISTER

The 32-bit I register holds the instructions read from memory by the read next instruction (RNI) cycle.

M REGISTERS

The 16-bit or 32-bit M registers hold the interrupt mask bits and control the recognition of interrupts by the interrupt system. A bit set in M enables its corresponding interrupt, while a clear bit disables the interrupt.

BIT REGISTER

A single register is used to perform single bit operations. The bit register is not part of the general register files.

RELOCATION REGISTER

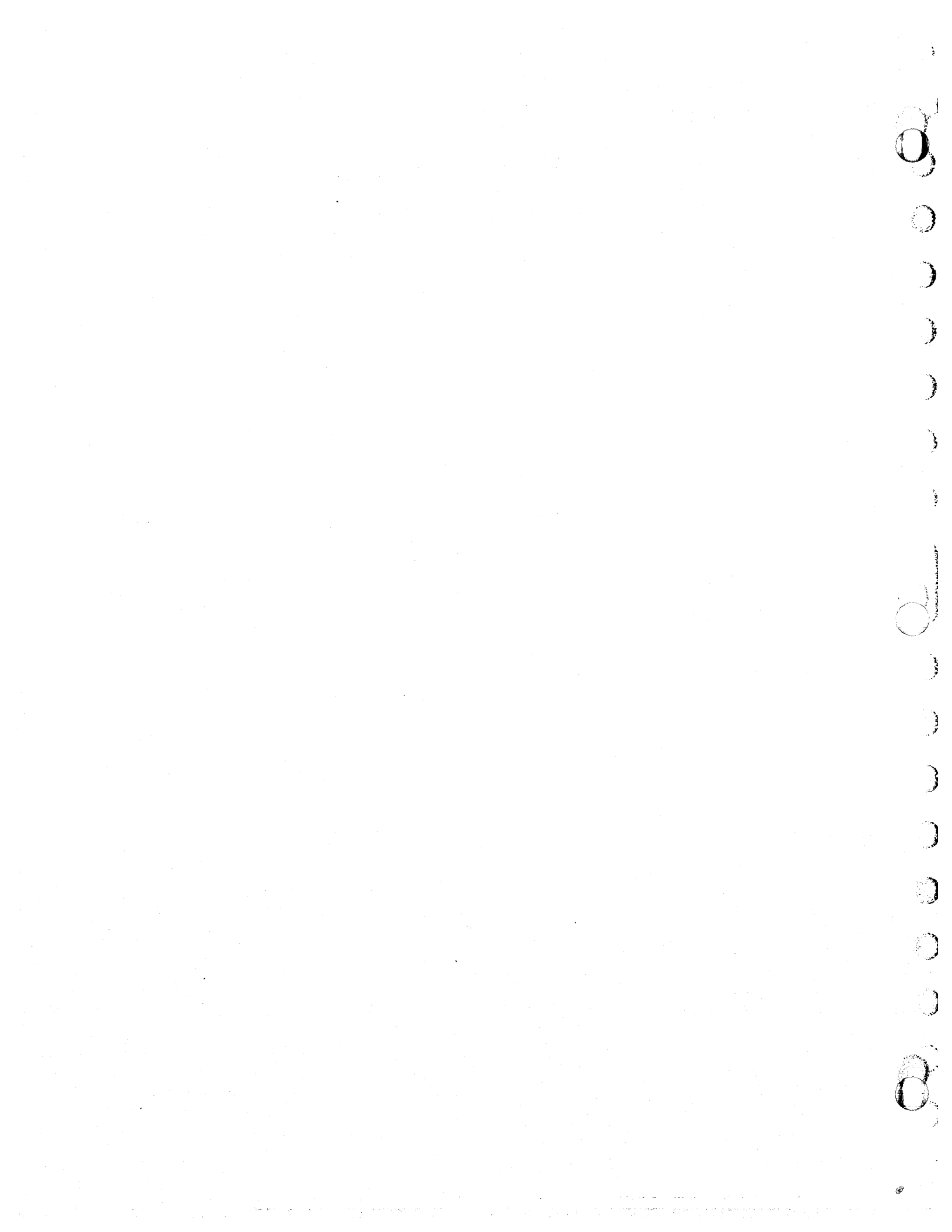
A single register is used by the system to hold the relocation state for the LDP and STP instructions. The relocation register is not included in the general register files.

REAL-TIME CLOCK

The real-time clock is a 32-bit counter that is incremented each millisecond to a maximum period of 4,294,967,296 milliseconds. The clock, which is controlled by a 1 KHz signal, starts as soon as power is applied to the computer. Instructions are available to set the real-time clock count (time-of-day) and to set an interrupt interval. If the interrupt system is enabled and the real-time clock mask bit is set in the interrupt mask register, an interrupt will be generated each time the specified interval time elapses. The counter is always updated and can be read as a source of time-of-day regardless of the status of the interrupt system. An interrupt interval must be specified in milliseconds.

PARITY

During each write cycle, a parity bit is stored along with each byte. When part or all of a word is read from storage by the central processor, parity is checked for a loss or gain of bits. Failure to produce the correct parity during read operations results in a memory parity error.



The minimum MP-60 system consists of a CPU, an IOC and one bank of memory. The IOC subsystem contains the logic necessary to route I/O requests to the peripheral equipment. The CPU initiates I/O operations by establishing an I/O request and invoking the IOC subsystem. The IOC subsystem responds by accepting the request, performing the operation, returning status and generating an interrupt. An IOC may be internal or external to the requesting CPU. When the IOC is external, it may be in an MP-60 emulator or a specialized I/O handler.

The CPU invokes the IOC subsystem with the following five instructions: CFO, CTO, SEL, SIO and CIO. The IOC subsystem will route the request to the proper processor. This processor will then perform the required I/O operation, return status and interrupt the requesting processor when the operation is complete.

INTERNAL INPUT/OUTPUT INTERFACE

The MP-32 microprogrammable processor is composed of many components used in the CDC CYBER 18 line of computers. Since the internal I/O interface is among those borrowed components, the MP-32 can, therefore, be connected to most CYBER 18 peripheral controllers. The MP-60 emulator on the MP-32 provides enhanced instructions which communicate to these peripheral controllers via a register I/O bus. The controllers are then cabled to their associated peripheral devices. If a controller has a CYBER 18 compatible direct memory access (DMA) channel, then the channel is connected to one of four DMA assembly/disassembly ports of the MP-32 memory system. The peripheral controllers may use three techniques to transfer data to and from the MP-60 system: Register Input/Output, Automatic Data Transfer (ADT) and Direct Memory Access (DMA).

The Register I/O method uses two registers to communicate commands and data to a peripheral controller. The "Y" register is used to designate the equipment

to be used, while the "D" register is used to hold function codes, to accept status information or to transfer data in and out of the MP-60 emulator. The MP-60 emulator allows the programmer to specify which of his 32 registers is the "D" and "Y" registers. During a data transfer, either a 16-bit half word or 8-bit character is transmitted to/from the "D" register, depending upon the peripheral controller. The data transfer rate of this technique depends on input/output instruction execution time and the peripheral hardware characteristics.

The Automatic Data Transfer (ADT) technique enables data transfers to and from memory in a buffered-type operation. At the MP-60 macro level, the transfer appears as a direct memory access (DMA). However, at the micro level, the MP-60 emulator processes each data interrupt and inputs/outputs the next element of data. Thus, the micro interrupt is fully transparent to the macro program and the macro program continues to execute while the data transfer is occurring at the micro level. The maximum data transfer rate is dependent upon the following:

- Interrupt sense time
- Micro data transfer time

The MP-60 emulator can process a maximum of 80K ADT interrupts per second.

The MP-32 memory interface has four (4) direct memory access (DMA) ports. The purpose of the DMA ports is to allow a controller to transfer data to/from main memory independent of processor intervention. This feature allows faster data transfer rates than is possible using the register I/O and ADT techniques. The maximum aggregate data transfer rate is approximately 1.6M transfers per second.

REGISTER INPUT/OUTPUT

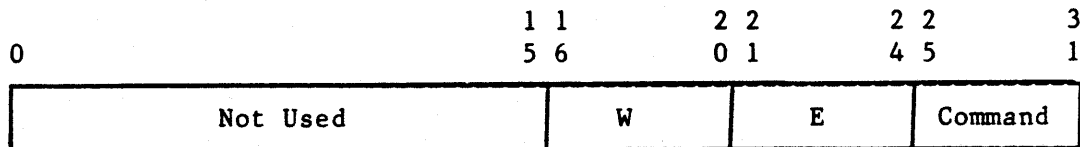
All control information for a peripheral controller is transmitted via the register I/O technique and is, therefore, the basis for all data transfers. Automatic Data Transfer (ADT) and direct memory access (DMA) controllers receive control information on the register I/O bus which initiates their buffered operations.

There exists two different register I/O protocols, in/out and set/sample, in the MP-32 peripheral controllers. These two types of I/O protocols use the same data and control paths. The difference between the two internal I/O busses is basically resolved in the MP-60 emulator, which controls the appropriate I/O signals and timings.

The MP-60 emulator on the MP-32 offers four macro register I/O instructions, two for each protocol (IN/OUT and NIO/SPS). The usage and format of the "Y" (address) and "D" (data) registers is supplied in the following paragraphs.

ADDRESSING — IN/OUT PROTOCOL

The "Y" register in the MP-60 emulator is used to send addressing codes to peripheral equipments. The format of the "Y" register is shown below. Each level of peripheral equipment (converter, equipment, station and unit), except a unit, is addressed by a unique section of the "Y" register.



The W field, bits 16 through 20, are reserved for addressing the converter. The W field must be zero for lower level peripheral devices and standard peripheral controllers.

The equipment field (E), bits 21 through 24, are used to contain the equipment number of the peripheral equipments on the I/O bus. Each device responds when the equipment number of the peripheral device, which is selectable, matches the code specified in bits 21 through 24.

The command code, bits 25 through 31, are not specifically used by the equipment and are, therefore, available to meet specific requirements of the station and unit within the equipment. These bits control and direct information on the I/O bus in the following ways:

- Specify the data transfer.
- Direct the control functions and function level.
- Direct the status and status level.
- Address the data bus to specific stations under one equipment having multiplexing capabilities.

The command code is divided into two sections: "S" contains the station code and "D" contains the director. The station code is located in bit 25 and adjacent lower order bits as required. The director is located in bit 31 and adjacent higher order bits as required. The S and D sections obviously cannot overlap and not all bits in the command code will necessarily be used.

Units are controlled by a higher-level peripheral controller and respond only to the controller. Units on the controller are selected by a function code which directs the data bus to select the unit.

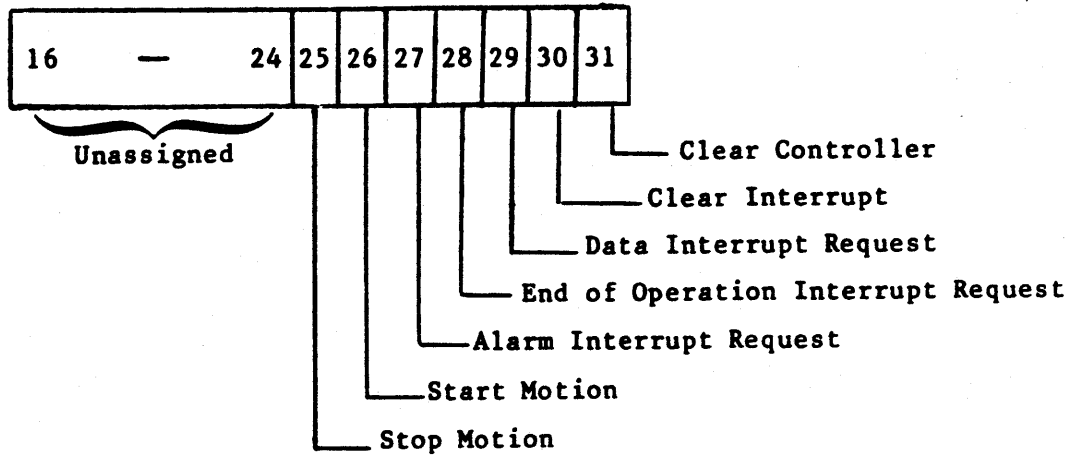
DATA TRANSFER — IN/OUT

To transfer data, the appropriate command must be coded in the director portion of the "Y" register. An IN or OUT instruction specifies the direction of data flow. If the peripheral equipment can receive data from or send data to the I/O bus, it will send a reply. If the peripheral device is not able to receive or send data, a reject will be returned. The data for the transfer originates or finishes in the "D" register.

DIRECTOR FUNCTIONS — IN/OUT

Equipment functions must also be coded in the director bits. An OUT instruction is used with the "D" register containing the information necessary to control the functions of the equipment.

FUNCTION BIT DEFINITIONS (CLASSICAL EXAMPLE)



CLEAR CONTROLLER

Bit 31 clears all interrupt requests and responses, motion requests, errors and other logic. A function code in which bit 31 is set and any of bits 25 through 29 are set will first clear all previous functions and then immediately set the function conditions indicated by bits 25 through 29.

CLEAR INTERRUPT

Bit 30 clears all interrupt requests and responses. A function code in which bit 30 is set and any of bits 25 through 29 are set will first clear all previous interrupt functions and then immediately set the function conditions indicated by bits 25 through 29.

DATA INTERRUPT REQUEST

Bit 29 sets a Data Interrupt Request, which enables the generation of an interrupt response by the peripheral equipment whenever a data transfer is possible. Both the interrupt request and response are cleared by either the Clear Controller or Clear Interrupt Director Functions, or by the Master Reset Control signal.

END OF OPERATION INTERRUPT REQUEST

Bit 28 selects the End of Operation Interrupt Request. An End of Operation results any time the continuous data transfer is interrupted; e.g., End of Record. Both the interrupt request and response are cleared by either the Clear Controller, Clear Interrupt or Master Reset signals.

ALARM INTERRUPT REQUEST

Bit 27 selects the Alarm Interrupt Request. An alarm may indicate a change of status (Ready to Not Ready) or it may be an indication of an error (Lost Data) or a warning (End of Tape). Each equipment must specify the manner in which the alarm is used and must provide a status indication for each condition causing an alarm. Both the interrupt request and response are cleared by either the Clear Controller, Clear Interrupt or Master Reset signals.

START MOTION

Bit 26 directs the device to start motion in its storage medium. If Start Motion is received while a block of data is being transferred, the remaining data within the block is not transferred. Data transfer resumes at the start of the next block. On machines which can halt between characters (non-interrupt on Data Mode), Start Motion initiates data transfer from the Hold register. Motion occurs only after the computer honors the process. If Start Motion does not apply to the particular device, this bit may be optionally redefined and used in some other manner.

STOP MOTION

Bit 25 halts the operation started by Start Motion. Stop Motion takes precedence over Start Motion. If Stop Motion is received while a block of data is being transferred, the remaining data within the block is not transferred. Motion, however, does not cease until the end of the block is reached.

UNASSIGNED

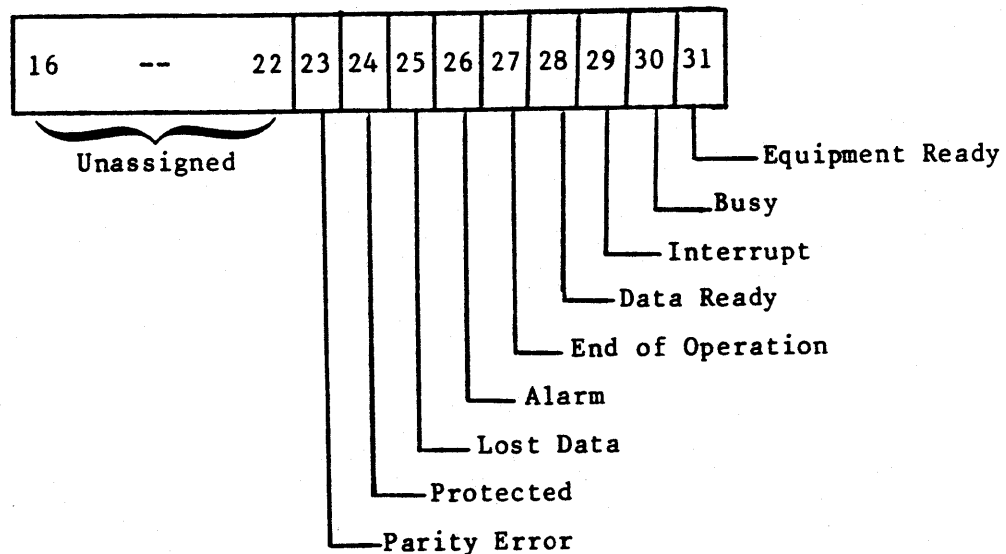
Bits 16 through 24 are unassigned and may be used at the discretion of the controller.

NOTE: Motion control may be received at any time from various parts of the program. The last function received is the one honored.

DIRECTOR STATUS — IN/OUT

Equipment status is obtained by the appropriate command in the director field of the "Y" register, and an IN instruction directs the data lines containing the status of the equipment into the MP-60 emulator.

STATUS BIT DEFINITIONS (CLASSICAL EXAMPLE)



EQUIPMENT READY

Bit 31 indicates that an equipment is Ready and an operation can be performed when requested by a Start request. Once Ready, an equipment remains so until operation is no longer possible. An equipment cannot become Not Ready while information transfer is actually in progress. Those equipments which require manual intervention must be made Ready manually.

BUSY

Bit 30 indicates that an equipment is Busy, or in operation. The equipment becomes Busy immediately upon initiation of the Start operation if the operation can be performed. Normally, an equipment remains Busy until it has finished all activity and is able to perform another operation.

INTERRUPT

Bit 29 indicates an interrupt response has been sent from this controller. Other bits must be monitored to determine the cause of the interrupt.

DATA READY

Bit 28 indicates that the controller is ready to perform a data transfer. If a data interrupt has been selected, this bit also indicates the type of interrupt which has occurred.

A director function sets the interrupt request. Another director function clears both the interrupt request and response.

During a Read operation, the interrupt response occurs when data has been loaded into the Data Hold register and is ready for transfer to the CPU. The interrupt response is cleared by the reply to data transfer.

During a Write operation, the interrupt occurs when data from the CPU can be loaded into the Data Hold register of the output device. The interrupt response is cleared by the reply to data transfer.

END OF OPERATION

Bit 27 indicates an End of Operation which means continuous transfers of data can no longer occur. It may also indicate the source of the interrupt response if the request had been selected. Each equipment specifies the particular conditions which constitute an End of Operation.

A director function sets the interrupt request. Another director function clears both the interrupt request and response. The operation may or may not be in progress at the time of the selection.

An EOP Interrupt cannot occur from an operation which has ended before the selection was made.

An operation and an End of Operation must be defined for each peripheral device.

ALARM INTERRUPT

Bit 26 indicates an alarm which may be caused by one of several conditions.

A director function sets the interrupt request and enables the device to generate the interrupt. Another director function clears both the interrupt request and response. An alarm condition that exists at the time of the interrupt request provides an immediate response.

The alarm conditions must be defined for each peripheral device. A status bit should indicate the state of each possible alarm condition. See reference information for the specific conditions on each device.

LOST DATA

Bit 25 indicates that data may have been lost. This occurs when the computer does not service the controller within the prescribed time for the device. This loss should be detected and displayed as Lost Data. This may be a condition for an Alarm Interrupt.

PROTECTED

Bit 24 indicates that the Program Protect switch for an equipment has manually been placed in the Protected position.

PARITY ERROR

Bit 23 indicates that a Parity Error has occurred in those storage devices that do incorporate parity as part of their format.

UNASSIGNED

Bits 22-16 are unassigned and may be used at the discretion of the controller. Where more practical, it may be desirable to assign another status level in the address and repeat use of the lower bits.

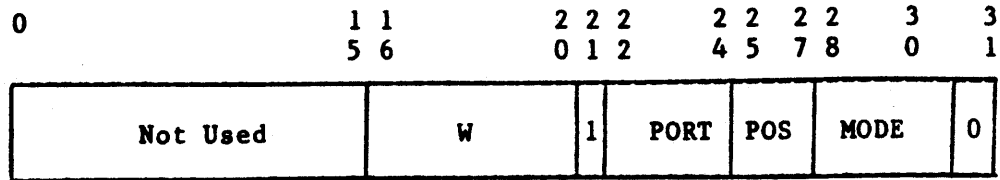
SET/SAMPLE PROTOCOL

The Set/Sample I/O structure provides eight port addressing and an eight-way priority scheme. Each port is numbered from 0 to 7. Port 7 has the lowest priority. Each port can communicate with a peripheral directly, or each port may optionally be further multiplexed to communicate with up to eight peripherals. Thus, up to 64 peripherals can be controlled. The second level of eight operates on a priority or scanning scheme supplied by external means.

Information is transferred to or from the device via the NIO instruction. The NIO instruction causes 16 bits to be input to the MP-60 processor or 16 bits to be transferred to the device called out in the instruction. The direction of the transfer is implied to the interface by the address bit 28, called "SMB9".

When address bit 28 is a "1", the 16 bits of information contained in the "D" register are placed on the output bus. Hence, during a sample condition, "Y" register bit 28 is a "0" and the 16 bits of information on the input bus are transferred to the "D" register.

The "Y" register is defined as follows:



Bits 16 through 20 and 31 must be zero.

The code in bits 22 through 24 specify the port number of the device.

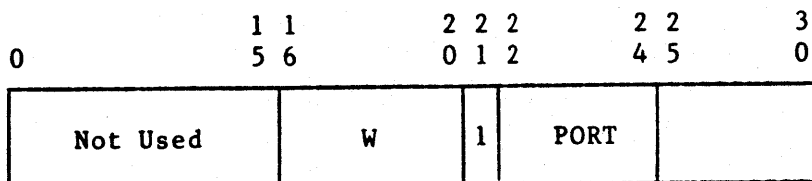
Bit 21 must be a "1". The code in bits 21 through 24 are directly analogous to the IN/OUT protocol's E field and must not conflict with any of these equipment codes.

The position of any multiple Set/Sample devices that are on the same port is specified by the code in bits 25 through 27.

Bits 28 through 30 specify the mode in which the selected device is to operate. Bit 28 is defined as the set/sample condition bit. When bit 28 is a "1", one 16-bit value will be set (output) from the "D" register. When bit 28 is a "0", one 16-bit value will be sampled (input) into the "D" register. Bits 29 and 30 are device dependent.

The SPS instruction (inputs) the position and status of a set/sample device. It is used whenever a set/sample device causes a macro interrupt to determine the position of the device generating the interrupt.

The "Y" register is defined as follows:

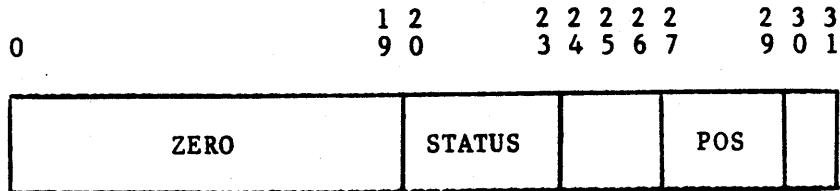


Bits 10 through 20, and 25 through 31, must be zero.

Bits 22 through 24 specify the port number of the device.

Bit 10 must be a "1".

Upon completion of the SPS instruction, the "D" register contains the following:



Bits 0 through 19, 24 through 26 and 30 through 31, will be zero.

The position of the device that generated the macro interrupt on the port will be in bits 27 through 29.

Bits 20 through 23 will contain four bits of status information. If these four bits of status are insufficient, the controller may provide additional status bits via the NIO instruction.

AUTOMATIC DATA TRANSFER

The MP-60 emulator supplies an I/O technique which transfers data to and from memory in a buffered-type operation. At the MP-60 macro level, the Automatic Data Transfer (ADT) appears as a direct memory access. However, at the micro level, the MP-60 emulator processes each data interrupt and transfers the next data element in a method similar to the register I/O technique. Thus, the micro interrupt is fully transparent to the macro program and the macro program continues to execute while the data transfer is occurring.

The MP-32 Processor is configured with eight ADT interrupt lines. These eight interrupt lines are connected via backpanel wiring to a particular controller.

The controller may be either protocol, In/Out or Set/Sample. The MP-60 emulator logically ties these eight interrupt lines to an ADT table entry.

The ADT table entry is set by the MP-60 instruction SMIO, and is used by the emulator to control the data transfer. The following information is contained in the ADT table:

- "Y" register for addressing controller
- number of transfers
- mode of transfer (halfword or character)
- current memory address

The MP-60 emulator uses two 32-bit registers to contain the above ADT table information. The formats of this table entry are given in Figure 2-1.

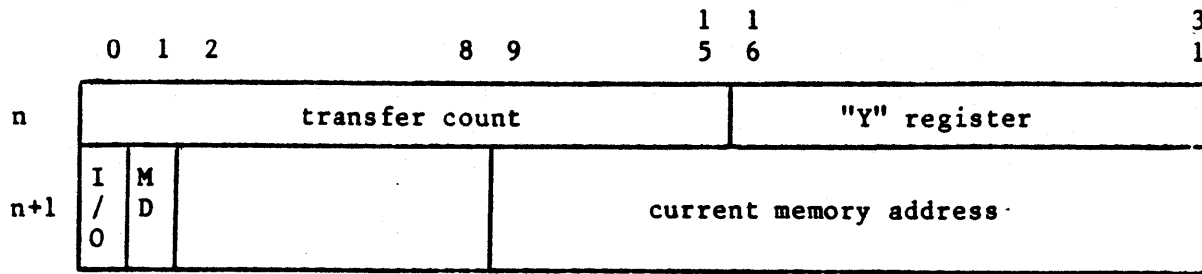
The MP-60 programmer performs the following operations to initiate the ADT micro-sequence.

1. Status controller and wait for busy status to drop.
2. Set up two registers with ADT table entry.
3. Execute SMIO instruction.
4. Condition controller for ADT and initiate ADT transfer.

When the MP-60 ADT micro sequence detects a transfer count of zero, or the controller detects an error or end of record, an end-of-operation interrupt is generated. The MP-60 programmer then performs the following sequence:

1. Status controller and clear interrupt.
2. Read ADT table using (RMIO) to determine record length.
3. Process interrupt status.

Since the MP-60 emulator is performing In/Out or Set/Sample protocols to the controller, extreme care should be taken when performing I/O instructions to a controller conditioned for an ADT micro sequence.



register n bits 0-15 transfer count, decremented by 1 each transfer

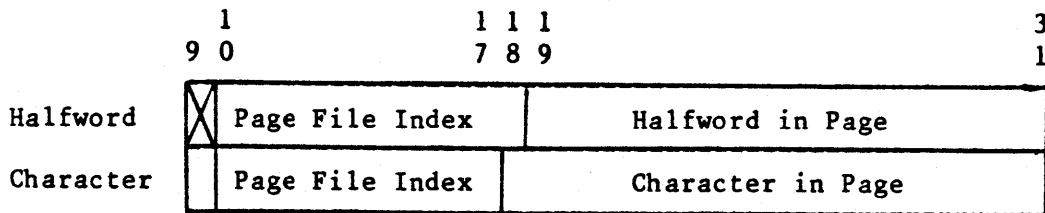
bits 16-31 "Y" address register for associated controller

register n+1 bits 0 Input/Output (In/Out Protocol only)
0 = input, 1 = output

bit 1 Transfer mode (In/Out Protocol only)
0 = halfword, 1 = character

bits 9-31 current memory address, incremented by 1 each transfer. See Figure 2-2.

Figure 2-1. ADT Table Entry Format



Page File Index -- address of page file entry applicable for this transfer (2 - if transfer crosses page boundary). See Section 5.

Figure 2-2. Current Memory Address Format

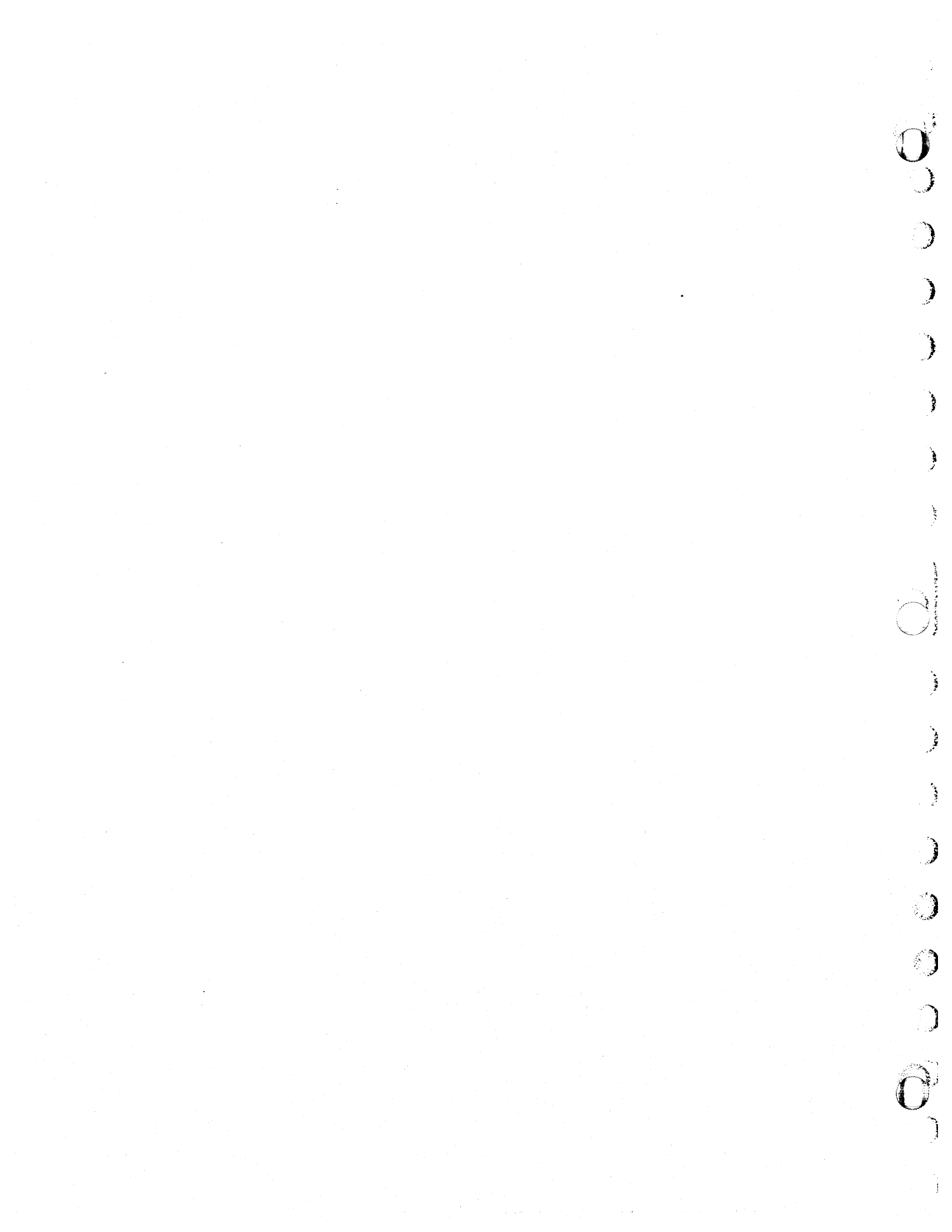
DIRECT MEMORY ACCESS

The four Direct Memory Access (DMA) ports, contained in the MP-32, supply medium and high-speed peripherals a direct data path to the external main memory. Since the DMA ports interface with the main memory interface, they gain use of its paging and lookahead capabilities.

Each DMA port contains separate request and accept control lines, but share the address and data busses. The address bus is 17 bits wide and specifies a halfword address associated with the 16-bit data value contained on the data bus. The DMA port creates a 16-bit DMA bus to 32-bit memory interface which includes the operating modes, full word and half word.

In the half-word mode, each DMA memory request results in a memory cycle while the full-word mode makes only one memory request to the DMA memory requests. The half-word mode is intended to be used on DMA devices which perform memory accesses randomly or terminate on uneven transfer counts. The full-word mode requires transfer in multiples of two and start/terminate on 32-bit word boundaries. These modes of data transfer are controlled by the DMA state register explained in the memory section.

The MP-60 programmer supplies all needed information to the DMA controller and includes the starting address in either bytes or halfwords and transfer count. This information is sent to the controller using either the register or set/sample protocols and once the transfer begins, no additional intervention is required. When the data transfer is complete, the controller may send an end-of-operation interrupt to the MP-32 processor.



The MP-60 emulator has been designed to support real-time applications. The primary criteria is the ability to respond quickly to time-critical events. The emulator provides this response by:

1. Reducing information saved during a state change.
2. Vectoring the priority interrupt system.

The information saved during a state change is minimized by having multiple register sets and page files. Only the information needed to place the interrupted state back into execution is collected and saved in the exchange package area. This reduction in saved information increases the rate at which state changes can be made and improves response time.

The vectored, priority interrupt system saves a software priority scheduling and vectoring system, thus reducing processing overhead.

EXCHANGE PACKAGE AREA

The exchange package area is a main memory buffer used to save the definition of environmental conditions associated with a machine state between executions in that state. Execution in a state begins by reading and establishing the environment's conditions from the exchange package area. Termination of execution in a state is complete only after the environment's conditions have been restored to the exchange package area. Execution is initiated with a CONT instruction and terminates at the occurrence of an interrupt or as a result of executing a MON instruction. Several instructions and the interrupt processing description refer to the exchange package area. The area layout and field definitions are provided in Figures 3-1 and 3-2.

| WORD | NON-REAL TIME MASK | | | | | | | | | | CPU | STATE | FAULTS | | | | M | D | | | | | | | | | | | | | | | | | | | | | | |
|------|--------------------|---|---|---|---|---|---|---|---|---|------|-------|--------|---|------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|--|--|--|--|--|--|
| | 0 | 1 | 2 | 3 | 4 | 5 | 7 | 8 | 9 | 0 | | | B | R | F | E | | | D | A | | | | | | | | | | | | | | | | | | | | |
| 1 | 0 | 1 | 2 | 3 | 4 | 5 | 7 | 8 | 9 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 3 | 6 | 7 | 8 | 9 | 0 | 1 | | | | | | |
| 2 | | | | | | | | | | | CW | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | PXPA | | | | SXPA | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | P1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | P2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | | | | | | | | | | | P3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | | | | | | | | | | | P4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Figure 3-1. Exchange Package Description

| | |
|-------------|--|
| A | Arithmetic overflow (mask or condition) |
| BR | Contents of bit register |
| CPU | Central processor number |
| CW | Current instruction word pointer |
| D | Divide fault (mask or condition) |
| E | Exponent fault (mask or condition) |
| F | Function fault (mask or condition) |
| FAULTS | Four fault mask bits or condition occurrence |
| IO | I/O interrupt enable |
| IP | Inter-processor enable |
| MD | Operating mode (program or monitor) |
| P1,P2,P3,P4 | Parameter values passed to executive |
| PXPA | Predecessor exchange package address |
| RTC | Millisecond lock interrupt mask bit |
| STATE | Executive state |
| SXPA | Successor exchange package area |

Figure 3-2. Exchange Package Area Field Descriptions

INTERRUPT SYSTEM

The interrupt control section of the MP-60 computer is capable of testing for the existence of certain internal and external conditions and, upon recognition of a condition, interrupting the main program flow. At the start of each RNI cycle, a test is made for interruptable conditions. If an interrupt exists, execution of the main program terminates, the current exchange package is updated, the interrupt system is disabled and an interrupt routine is initiated. After the interrupt routine has completed its processing, main program execution is resumed by executing a CONT instruction. Each interrupt in the MP-60 is assigned a unique memory address containing the entry point of the interrupt processing routine.

INTERNAL INTERRUPTS

The internal interrupts represent operational fault conditions that may be detected by the programmer on a conditional or unconditional basis. Four internal interrupts are in the conditional category and are controlled through the use of the interrupt mask register (refer to Table 3-1). The other interrupts are always enabled regardless of the state of the interrupt system or interrupt mask register.

ARITHMETIC OVERFLOW FAULT

The arithmetic overflow fault is set when the capacity of the adder is exceeded. The adder capacity, including sign, is 32 and 64 bits for single and double precision, respectively. The arithmetic overflow fault is a maskable (conditional) interrupt. The following instructions can cause an arithmetic fault.

Mnemonics:

| | | | |
|-----|-----|------|------|
| AD | SB | ADD | SBD |
| R,+ | R,- | RD,+ | RD,- |

FUNCTION FAULT

The function fault is set when a function instruction encounters a fault condition. The function fault is a maskable (conditional) interrupt. The following instructions can cause a function fault.

| <u>Mnemonic</u> | <u>Cause</u> |
|-----------------|------------------------------------|
| F,SQ | Negative value |
| F,UF | Integer value greater than 32 bits |
| FD,UF | Integer value greater than 64 bits |

EXPONENT FAULT

During all floating point operations, overflow occurs if the resultant exponent exceeds $+7F_{16}$ and underflow occurs if the resultant exponent exceeds -80_{16} . The exponent fault is a maskable (conditional) interrupt. The following instructions can cause an exponent fault:

Mnemonics:

| | | | |
|-----|------|------|-------|
| FAD | FADD | RF,+ | RFD,+ |
| FSB | FSBD | RF,- | RFD,- |
| FMP | FMPD | RF,* | RFD,* |
| FDV | FDVD | RF,/ | RFD,/ |

DIVIDE FAULT

The divide fault is set if a quotient, including sign, exceeds the capacity of the adder (DV and R,/ instructions). A divide fault also occurs when an attempt is made to divide by zero, fixed or floating point. The divide fault is a maskable (conditional) interrupt.

ILLEGAL INSTRUCTION

When a privileged instruction is encountered in program mode, an illegal instruction interrupt is generated. This interrupt is also generated upon detection of an illegal operation code. Illegal operation codes are those reserved for instruction set expansion. The illegal instruction is always enabled.

PAGE FAULT 1

The page fault 1 interrupt is generated when an attempt is made to store an operand into a read only page. The page fault 1 interrupt is always enabled.

PAGE FAULT 2

The page fault 2 interrupt is generated when a protected memory location is referenced as either an instruction or operand. The page fault 2 interrupt is always enabled.

PAGE FAULT 3

The page fault 3 interrupt is generated when a nonresident memory location is referenced as either an instruction or operand. The page fault 3 interrupt is always enabled.

MEMORY PARITY ERROR

When a memory parity error is sensed on either an instruction or operand reference, a parity error interrupt is generated. The memory parity error is always enabled.

MEMORY REJECT

The memory reject interrupt signifies that an attempt was made to address beyond the range of available system memory, either as an instruction or operand. A memory reject may indicate loss of power in a memory module. The memory reject interrupt is always enabled (unmaskable).

POWER FAILURE

A power failure interrupt is generated upon detection of loss of power. The power failure interrupt overrides all other interrupts regardless of the state of interrupt control. This interrupt causes the disabling of the normal interrupt system. The power failure interrupt is nonmaskable.

I/O INTERRUPTS

The MP-60 may be configured with a number of peripheral controllers. These controllers may generate interrupts which inform the peripheral drivers of external conditions. These I/O interrupts are maskable.

REAL-TIME INTERRUPTS

The MP-60 has been designed to facilitate its use in a wide range of real-time applications. To accomplish this, a group of sixteen priority-ordered interrupts is available for use in each unique system. All real-time interrupts are maskable.

Real-time interrupts are external signals to the MP-60 System. A real-time source must generate an interrupt by transmitting a pulse from 500-1000 nanoseconds. This pulse is captured in an internal register, which generates the interrupt condition. When the interrupt is recognized by the MP-60, the appropriate bit in the holding register is cleared, thus removing the interrupt condition.

INTER-PROCESSOR INTERRUPTS

The emulator has been designed to simplify multiprocessor communications. When a processor executes a DST instruction, then the appropriate processor will sense the inter-processor interrupt. The inter-processor interrupts are maskable, as a group.

INTERRUPT MASK REGISTER

The programmer may choose to honor or ignore an interrupt by means of the interrupt mask register. These mask registers can be selectively set and selectively cleared, thus controlling interrupt recognition. Tables 3-1 and 3-2 show the assignment of these interrupt mask registers.

INTERRUPT CONTROL

The enable interrupt system (EINT) and continue (CONT) instructions enable the maskable interrupts. The nonmaskable interrupts are enabled at all times. Receipt of any interrupt results in the disabling of the maskable interrupts. The interrupt system will also be disabled by executing either a disable interrupt system (DINT) or monitor call (MON) instructions.

INTERRUPT RECOGNITION

Those interrupts which are not under mask control are always recognized during the RNI sequence. Maskable interrupts are recognized if the following conditions exist:

1. The interrupt system is enabled.
2. The appropriate bit in the interrupt mask register is set.

Recognition of an interrupt causes execution to be discontinued for the current state and initiates execution in the privileged monitor mode (state 0).

Discontinuation of execution proceeds as follows:

1. The execution environment is collected and stored into word 1 of the current exchange package area (refer to Figure 3-1).
2. The instruction address pointer is stored into word 2 of the current exchange package area.

3. The SXPA (successor exchange package address) field of word 3 of the current exchange package is read and defines the address of the new monitor mode exchange package area.
4. The interrupt system is disabled.
5. Word 3 of the new exchange package is redefined by writing the address of the old exchange package area into the PXPA (predecessor exchange package address) field.
6. A location defined for the interrupt is read and establishes the initial execution address for execution in the monitor.

These unique locations for each interrupt are given in Table 3-3.

INTERRUPT PRIORITY

Table 3-4 shows the priority of interrupt on the MP-60 System. Within a group, the order specifies the priority with highest given first.

TABLE 3-1. REAL-TIME INTERRUPT MASK BIT ASSIGNMENT

| MASK BIT | INTERRUPT CONDITION |
|-------------|------------------------|
| 00 | REAL TIME 15 |
| 01 | REAL TIME 14 |
| 02 | REAL TIME 13 |
| 03 | REAL TIME 12 |
| 04 | REAL TIME 11 |
| 05 | REAL TIME 10 |
| 06 | REAL TIME 9 |
| 07 | REAL TIME 8 |
| 08 | REAL TIME 7 |
| 09 | REAL TIME 6 |
| 10 | REAL TIME 5 |
| 11 | REAL TIME 4 |
| 12 | REAL TIME 3 |
| 13 | REAL TIME 2 |
| 14 | REAL TIME 1 |
| 15 | REAL TIME 0 |

TABLE 3-2. INTERRUPT MASK BIT ASSIGNMENTS

| MASK BIT | INTERRUPT CONDITION | | | |
|----------|---------------------|-----|--------|----|
| 00 | Arithmetic Fault | | | |
| 01 | Function Fault | | | |
| 02 | Exponent Fault | | | |
| 03 | Divide Fault | | | |
| 04 | | | | |
| 05 | | | | |
| 06 | | | | |
| 07 | | | | |
| 08 | Clock Interval | | | |
| 09 | | | | |
| 10 | | | | |
| 11 | | | | |
| 12 | | | | |
| 13 | | | | |
| 14 | | | | |
| 15 | | | | |
| 16 | Macro | I/O | Number | 0 |
| 17 | Macro | I/O | Number | 1 |
| 18 | Macro | I/O | Number | 2 |
| 19 | Macro | I/O | Number | 3 |
| 20 | Macro | I/O | Number | 4 |
| 21 | Macro | I/O | Number | 5 |
| 22 | Macro | I/O | Number | 6 |
| 23 | Macro | I/O | Number | 7 |
| 24 | Macro | I/O | Number | 8 |
| 25 | Macro | I/O | Number | 9 |
| 26 | Macro | I/O | Number | 10 |
| 27 | Macro | I/O | Number | 11 |
| 28 | Macro | I/O | Number | 12 |
| 29 | Macro | I/O | Number | 13 |
| 30 | Macro | I/O | Number | 14 |
| 31 | Macro | I/O | Number | 15 |

TABLE 3-3. LOW MEMORY ASSIGNMENTS

| | | | |
|-------------------------|----|--------------------------|----|
| 10 - Inter-Processor | 15 | 20 - Inter-Processor I/O | 15 |
| 1 - Inter-Processor | 14 | 1 - Inter-Processor I/O | 14 |
| 2 - Inter-Processor | 13 | 2 - Inter-Processor I/O | 13 |
| 3 - Inter-Processor | 12 | 3 - Inter-Processor I/O | 12 |
| 4 - Inter-Processor | 11 | 4 - Inter-Processor I/O | 11 |
| 5 - Inter-Processor | 10 | 5 - Inter-Processor I/O | 10 |
| 6 - Inter-Processor | 9 | 6 - Inter-Processor I/O | 9 |
| 7 - Inter-Processor | 8 | 7 - Inter-Processor I/O | 8 |
| 8 - Inter-Processor | 7 | 8 - Inter-Processor I/O | 7 |
| 9 - Inter-Processor | 6 | 9 - Inter-Processor I/O | 6 |
| A - Inter-Processor | 5 | A - Inter-Processor I/O | 5 |
| B - Inter-Processor | 4 | B - Inter-Processor I/O | 4 |
| C - Inter-Processor | 3 | C - Inter-Processor I/O | 3 |
| D - Inter-Processor | 2 | D - Inter-Processor I/O | 2 |
| E - Inter-Processor | 1 | E - Inter-Processor I/O | 1 |
| F - Inter-Processor | 0 | F - Inter-Processor I/O | 0 |
| | | | |
| 30 - CFO Table Pointer | | 40 - Arithmetic Fault | |
| 1 - CTO Table Pointer | | 1 - Function Fault | |
| 2 - SEL Table Pointer | | 2 - Exponent Fault | |
| 3 - | | 3 - Divide Fault | |
| 4 - | | 4 - | |
| 5 - | | 5 - | |
| 6 - | | 6 - | |
| 7 - | | 7 - | |
| 8 - | | 8 - Clock Interval - lms | |
| 9 - | | 9 - | |
| A - SIO Table Pointer | | A - | |
| B - CIO Table Pointer | | B - | |
| C - | | C - | |
| D - | | D - | |
| E - | | E - | |
| F - Illegal Instruction | | F - | |

TABLE 3-3. LOW MEMORY ASSIGNMENTS (Cont'd)

| | | | |
|------------------------------------|----|-------------------|----|
| 50 - REAL TIME | 15 | 60 - MACRO I/O | 15 |
| 1 - REAL TIME | 14 | 1 - MACRO I/O | 14 |
| 2 - REAL TIME | 13 | 2 - MACRO I/O | 13 |
| 3 - REAL TIME | 12 | 3 - MACRO I/O | 12 |
| 4 - REAL TIME | 11 | 4 - MACRO I/O | 11 |
| 5 - REAL TIME | 10 | 5 - MACRO I/O | 10 |
| 6 - REAL TIME | 9 | 6 - MACRO I/O | 9 |
| 7 - REAL TIME | 8 | 7 - MACRO I/O | 8 |
| 8 - REAL TIME | 7 | 8 - MACRO I/O | 7 |
| 9 - REAL TIME | 6 | 9 - MACRO I/O | 6 |
| A - REAL TIME | 5 | A - MACRO I/O | 5 |
| B - REAL TIME | 4 | B - MACRO I/O | 4 |
| C - REAL TIME | 3 | C - MACRO I/O | 3 |
| D - REAL TIME | 2 | D - MACRO I/O | 2 |
| E - REAL TIME | 1 | E - MACRO I/O | 1 |
| F - REAL TIME | 0 | F - MACRO I/O | 0 |
| 70 - MICRO I/O | 7 | 80 - MONITOR CALL | |
| 1 - MICRO I/O | 6 | | |
| 2 - MICRO I/O | 5 | | |
| 3 - MICRO I/O | 4 | | |
| 4 - MICRO I/O | 3 | | |
| 5 - MICRO I/O | 2 | | |
| 6 - MICRO I/O | 1 | | |
| 7 - MICRO I/O | 0 | | |
| 8 - | | | |
| 9 - | | | |
| A - | | | |
| B - | | | |
| C - | | | |
| D - DMA Memory Error Table Pointer | | | |
| E - CPU Memory Error Table Pointer | | | |
| F - POWER FAILURE | | | |

TABLE 3-4. INTERRUPT PRIORITY

| LEVEL | INTERRUPT GROUPS |
|-------|--------------------------|
| 1 | Power Failure |
| 2 | CPU Memory Errors |
| 3 | DMA Memory Errors |
| 4 | Illegal Instruction |
| 5-12 | Micro I/O 0-8 |
| 13-28 | Macro I/O 0-15 |
| 29-44 | Real-Time 0-15 |
| 45-50 | Open |
| 51 | Open |
| 52 | Clock Interval |
| 53-55 | Open |
| 56 | Inter-Processor |
| 57-60 | Faults Divide-Arithmetic |

1 - highest, 60 - lowest

The MP-32 is equipped with a CDC 1811-1 CRT display console. The CDC 1811-1 CRT display console consists of a display with a detachable keyboard. The MP-32 display console is a multi-functional unit performing as the processor control panel, besides providing to the software an interface with the operator.

The CDC 1811-1 uses a 12-inch rectangular CRT to display alphanumeric data and special symbols. The display area is made up of 24 lines; each line may contain 80 symbols. The repertoire that may be displayed is the USASCII (ANSI) X3.4-1968 character set. A cursor or entry marker (a blinking dash on the screen) indicates where a symbol will display on the screen.

The operator enters data onto the display and controls routing of data to the processor via the display keyboard. The keyboard contains a character set of 128 characters. In addition, the keyboard contains all the necessary controls to regulate cursor positioning, display presentation and data transmission to the processor.

The display console is connected to the MP-32 via backpanel connection and uses asynchronous RS-232-C/CCITT V.25 compatible line protocols. The CRT display console's normal modes are even parity, 9600 baud, and full duplex, all switch selectable on the display console.

Software communications with the MP-32 display console uses the standard MP-60 register IN/OUT and Automatic Data Transfer (ADT) protocols. Appendix B gives the complete details of the "Y" address and "D" data register commands and formats.

PANEL INTERFACE

The MP-32 display console combines the functions of a software display and processor control panel. This console sharing is accomplished by processing particular special character codes as control characters.

These special codes (ESCAPE, $1B_{16}$; @, 40_{16} ; BELL, 07_{16}) are reserved as control codes and cannot be accepted as message data. The ESCAPE code causes the display console to be transferred from software control to processor control. The @ code transfers the display console back to software control from processor control. The final code, BELL, is used to send a manual interrupt to the software.

Basic to the operation of the processor control console is the Function Control Register (FCR). The FCR is a 32-bit register that has access to the MP-32 similar to the way switches on a conventional panel have access to the CPU.

The FCR register, as depicted in Figure 4-1, is grouped into eight hexadecimal digits (0 through 7). These hexadecimal digits are divided into four types as follows:

- Display -- Digits 0 and 1
- Control -- Digits 2 and 3
- Modes -- Digits 4 and 5
- Status -- Digits 6 and 7

The display digits determine which registers of two groups, shown in Table 4-1, may be displayed and/or modified. The control digits are used to set such conditions as step and run. The mode bits control display modes and micromemory modifications. The two least significant digits (6 and 7) of the FCR indicate the status of the processor, such as monitor/program mode and CPU/DMA memory error.

| | | | | | | | | |
|---------|---------|---------|------------|------------|------------|------------|------------|--------|
| 0 | 3 4 | 7 8 | 1 1 1 2 | 1 1 5 6 | 1 2 9 0 | 2 2 3 4 | 2 2 7 8 | 3 1 |
| DISPLAY | DISPLAY | CONTROL | CONTROL | MODE | MODE | STATUS | STATUS | |

| BIT | DIGIT | BIT DEFINITION | |
|----------------------------------|---------|--|---------------------|
| 31 1F 30 1E 29 1D 28 1C | 7 (LSB) | CPU State Reg 31 CPU State Reg 30 CPU State Reg 29 Monitor/Program | } Status Only |
| 27 1B 26 1A 25 19 24 18 | 6 | CPU MEM I/F Error DMA MEM I/F Error Micro Running Macro Running | |
| 23 17 22 16 21 15 20 14 | 5 | Enable Auto Display Enable Console Echo | } Modes |
| 19 13 18 12 17 11 16 10 | 4 | Enable Micromemory Write Suppress Console Transmit | |
| 15 0F 14 0E 13 0D 12 0C | 3 | Refer to BP Description in Control Functions Section BP Int. (BP Int. (BP Stop if Clr) Micro BP, Step, Go, Stop (Macro if Clr) | |
| 11 0B 10 0A 09 09 08 08 | 2 | Step Debug | |
| 07 07 06 06 05 05 04 04 | 1 | DISPLAY 1 | } Display Selection |
| 03 03 02 02 01 01 00 00 | 0 (MSB) | DISPLAY 0 | |

Figure 4-1. Function Control Register (FCR)

TABLE 4-1. DISPLAY CODE DEFINITIONS

| CODE | SELECT CODE | DISPLAY 1 "K FUNCTION" | SELECT CODE | DISPLAY 0 "L FUNCTION" |
|-----------|-------------|---------------------------|-------------|--|
| 0 0 0 0 0 | J10: | FCR | J00: | F2 |
| 1 0 0 0 1 | J11: | P | J01: | N |
| 2 0 0 1 0 | J12: | I | J02: | K |
| 3 0 0 1 1 | | - | J03: | X |
| 4 0 1 0 0 | J14: | A | J04: | Q |
| 5 0 1 0 1 | J15: | MIR | J05: | F |
| 6 0 1 1 0 | J16: | BP/P-MA | J06: | F1 (F3, providing setup of the ad- dressing method is fulfilled.) |
| 7 0 1 1 1 | J17: | BP/P-MA (Display only) | | - |
| 8 1 0 0 1 | J18: | SM1 | | - |
| 9 1 0 0 1 | J19: | M1 | J09: | RTJ |
| A 1 0 1 0 | J1A: | SM2 | | - |
| B 1 0 1 1 | J1B: | M2 | | - |
| C 1 1 0 0 | | - | J0C: | MM |
| D 1 1 0 1 | | - | | - |
| E 1 1 1 0 | | - | | - |
| F 1 1 1 1 | | - | | - (dash specifies an undetermined result). |

FUNCTIONAL OPERATION

The interface accepts nine different control characters: H, I, J, K, L, @, G, : and ?. G and @ are functionally identical to a colon; however, in this manual all operator functions use the colon (:). H through L identify the type of data or operation entered or returned. The character colon (:) terminates all entries except Master Clear.

A question mark will generate a Master Clear to the computer, memory and peripherals. There is no response to this entry.

A normal entry consists of one control character - H through L; zero, two, four or eight hexadecimal digits 0 through F; and the colon (:), in that order. If a transmission or operator error occurs on the entry, the control character is preceded by an asterisk (*) and the Function Control Register is displayed. All entries except ? cause a response unless bit 10₁₆ of the FCR is set (suppress console transmit).

CONTROL FUNCTIONS

The control functions are used as follows:

H CONTROL FUNCTION

This function is used to clear a specific bit in the FCR.

Example: H14:

This would clear bit 14₁₆ in the FCR and the response would be a display of the updated FCR.

I CONTROL FUNCTION

This function is identical to H except it sets a bit in the FCR. H and I are also used for Stop/Run control.

J CONTROL FUNCTION

The J control function is used to replace the contents of the function control register in a digit mode. While it may be used to change the value of any FCR digit, it is generally used to change digits 0 and 1. The value of display 0 and display 1 specifies which MP register is displayed on display requests or entered on enter requests. J functions always consist of J followed by two hexadecimal digits and the :. The first hexadecimal digit specifies the FCR digit 0 through 5 and the second hexadecimal digit specifies the value is to assume, 0 through F.

Example: J14:

This would set FCR digit 1 to 4 (select the A register) and the response would be a display of the updated FCR.

The J code is also used to alternately display the upper and lower 16 bits of a 32-bit register on the maintenance panel.

Example: J:

This would cause display of the other 16 bits and would complement the U/L indicator on the maintenance panel.

K CONTROL FUNCTION

The K control function is used to display or enter data onto the parameter specified by Display 1. The K functions use two formats. The first format is a request to display the parameter specified by Display 1.

Example: K:

The second format is an enter data request. The data is entered into the register specified by Display 1. It consists of K followed by four or eight hexadecimal digits, terminated by the :. The hexadecimal digits are the data to be entered. Some examples are:

To display the P register, perform the following:

J11: Set Display 1 to P register (1)

K: Display register selected in Display 1.

- To enter $14E_{16}$ into the breakpoint register, perform the following:

J16: Set Display 1 to BP register (6).

K16FE: Enter into register selected in Display 1.

L CONTROL FUNCTION

Operationally, the L function is the same as the K function, except it is associated with Display 0.

When micromemory is displayed or entered, the K register is the least significant eight bits of the address, and the N register provides the remaining bits. The K register is incremented by 1 after the display.

STOP/GO CONTROL

The following entry will cause a go:

Entry: I:

This is a micro go when bit 12 of FCR is set. It is both a micro and macro go when bit 12 of FCR is clear.

The following entry will cause a stop:

Entry: H:

This is a micro stop when bit 12 of FCR is set. It is a macro stop when bit 12 of FCR is Clear. The response to a start or stop entry is a display of the FCR.

MASTER CLEAR

A master clear can be generated in several ways:

- A question mark from remote console.
- A signal from a peripheral controller.
- A power on Master Clear.

BREAKPOINT (BP)

There are two types of breakpoint: micro and macro. When bit 12 of the FCR is set, micro BP is selected. Use of the micro and macro BP is described below:

- Micro BP - FCR bits 14 and 15 are used to select two types of micro breakpoint.

| <u>Bit 14</u> | <u>Bit 15</u> | |
|---------------|---------------|----------------------------------|
| 0 | 0 | BP not selected |
| 0 | 1 | Upper/lower micro instruction BP |
| 1 | 0 | Micro-word BP |
| 1 | 1 | Micro-word BP |

The upper/lower micro instruction breakpoint requires that the micro memory address P/MA and upper/lower micro instruction selections are equal to the lower 13 bits of the breakpoint register to cause a micro stop. The micro-word breakpoint only requires that the 12 bits of micro-memory address register P/MA are equal to the lower 12 bits of the breakpoint register.

- Macro BP - FCR bits 14 and 15 are used to select three types of macro breakpoint.

| <u>Bit 14</u> | <u>Bit 15</u> | |
|---------------|---------------|--|
| 0 | 0 | BP not selected |
| 0 | 1 | BP on Lookahead Register 3 |
| 1 | 0 | BP on Lookahead Register 2 |
| 1 | 1 | BP on all Lookahead Registers (LA1, LA2, LA3) |

A macro breakpoint stop occurs when the breakpoint register is equal to the least significant 16 bits of the main memory address and the select conditions are met. If FCR bit 13 is set, an interrupt, rather than a stop, occurs when the breakpoint conditions are met.

AUTO DISPLAY

The auto-display feature is selected by setting bit 15₁₆ in the FCR. It is removed by clearing this bit. When auto-display is selected, the panel interface continuously displays the register determined by the last control code and the display 0 or display 1 function. No line feeds are issued in this mode, so the display stays at the same link on the screen. Note that this precludes using auto-display with a teletypewriter.

If a terminator (colon, G or @) is pushed with no characters preceding it, a Go signal is generated. This feature is convenient for stepping through either macro or microprograms.

SPECIAL CONSIDERATIONS

- The Enable Auto Display and Enable Console Echo bits of the FCR (15₁₆ and 14₁₆) are mutually exclusive; that is, the user may select either one or the other, but not both at the same time.
- Display 1 selection of BP, P/MA permits changing contents of BP only. P/MA cannot be modified by the FCR.
- Display 0 for N or K will cause both N and K to be displayed. However, for code 0001, only N can be modified and for code 0010, only K can be modified.
- Selecting invalid display codes (undefined) may result in abnormal MP-32 operations.
- Transmit operation - During this operation, in response to the control command, the contents of the selected register are transmitted to the display console.

The input and transmit operations are performed entirely by the panel interface module. To perform the change/fetch operation, the panel interface

must generate micro-instructions acceptable to the MP-32 processor. The panel interface must determine if it needs the MP-32 processor; the processor is normally required except when the MIR, FCR or BP/MA are the scheduled registers. If the processor is required, the sequence of operations depends upon the setting of the FCR mode bit 12_{16} .

MICRO MODE

When the panel interface is in micro mode (FCR bit $12_{16}=1$), the panel interface may only gain control of the MP-32 processor when the processor is halted. When the panel interface gains control of the processor, the following is performed:

- Set the status mode register bit SM214.
- Clear the microinstruction register.
- Disable the micromemory and enable the Panel I/F microinstructions to the micromemory 3-state bus.
- Disable the macromemory address buffer register except for Read/Write Macromemory operation.
- Save the micromemory address.
- Save test bit TB.

The panel interface now causes the MP-32 to begin running and be in micro mode. The NOP microinstruction, MIR = 0, is executed. The panel interface generates a microinstruction sequence with M field = 01, which is executed by the processor to change/fetch the selected register. Prior to execution of the last panel interface microinstruction, the panel interface clears status mode bit SM214. The panel interface now generates the last microinstruction with M field = 00 and T field = 000 to exit the processor. The M field = 00 causes the re-enabling of micromemory and the disabling of the panel interface to the microinstruction register. Now the processor reads up the microinstruction awaiting in MIR prior to the panel interface sequence. The panel interface also stops the processor with a micro halt.

MACRO MODE

When the MP-32 is in macromode, the MP-60 emulator is required to prepare the processor before giving up the control to panel interface. When the panel interface determines that it needs the CPU, it generates micro interrupt INT05 which indicates to the emulator the panel interface request. Once the interrupt 05 is recognized by the MP-60 emulator during the read next instruction (RNI) loop, the emulator performs the following:

- Sets the Status Mode Interrupt bit SM214 to pre-enable panel I/F to the microinstruction register (MIR).
- Set RTJ register to re-entrance address for firmware execution after the panel interface sequence.
- Execute a GO return (M field = 00) microinstruction which disables microinstruction fetching from micromemory.

This disabling of micromemory enables the panel interface module to the 3-state bus. The panel interface now has the control of the processor. The panel interface generates a microinstruction sequence with M = 01 to be executed by the MP-32. After the MP-32 completes the required operation, the panel interface exits clearing INT05 and SM214 in the processor. This action re-enables micromemory to the micromemory bus. The last panel interface microinstruction is executed with M field = 00 and T field = 001. The M = 00 causes enabling the micromemory and disabling the panel interface to the micromemory 3-state bus. The M field = 00 also selects the content of RTJ register as the address of the next microinstruction pair. Note that the RTJ contained the re-entrance address for the emulator. The T field = 001 selects the upper microinstruction at the address specified by RTJ register. The emulator now has control back and continues at the point that it left off prior to panel interface sequence.

This interaction between the panel interface and the MP-60 emulator allows the emulator to set up certain display registers with pertinent data. The interaction also allows the panel interface MP-60 emulator to operate in a special debug mode.

MP-60/PANEL INTERFACE DISPLAYS

Prior to allowing the panel interface to assume control of the processor, the MP-60 emulator forces control information into predetermined registers. Table 4-2 defines the register allocation which the MP-60 emulator utilizes.

TABLE 4-2. MP-60 REGISTER UTILIZATION

| REGISTER | REGISTER CONTENTS |
|-----------|---------------------------------|
| P J11G KG | Program Counter |
| I J12G KG | Next Instruction to be executed |

The MP-32 memory system is composed of two basic modules: the Multi-Port Memory (MPM) Chassis and the CPU Memory Management Interface (MMI). The multi-port memory chassis may support as many as eight (8) different CPU memory management interface modules. The CPU memory management unit may be daisy-chained to sixteen (16) multi-port memory chassis.

MULTI-PORT MEMORY

The MPM chassis may be configured with two to eight CPU MMI ports. These ports are added in increments of two and each port may be disabled to facilitate the support of an odd number of CPU memory management interfaces. The MMI ports provide the interface and buffering between the CPU MMI and the independent memory banks. The MPM chassis may contain from one to four independent memory banks, each bank containing either 32K or 64K words of MOS semiconductor memory. Each of the eight ports has access to all four memory banks and all four banks may be active simultaneously if they are accessed by different ports.

The independent memory bank control logic resolves memory request conflicts from the eight ports and refresh. If a bank is not busy, the first request presented is accepted. When a bank is about to go not busy, the eight ports and refresh are scanned in order, starting one higher than the current active port. If a request is active, it is accepted; thus, a port will never wait longer than eight cycles to be accepted.

The bank control logic provides a method of reading and altering memory locations with the assurance that no other port has read or modified those locations between the read and write. Processors may use this method to communicate without the danger of logic discontinuities. This method is done by locking a bank to a port, which locks all other ports (but not refresh) from accessing the "locked" memory bank. This locking technique is used by the MP-60 emulator in the destructive load (DLD) instruction sequence. The

MP-60 emulator locks the bank on the load and unlocks the bank after destroying the memory location.

CPU MEMORY MANAGEMENT INTERFACE

The CPU memory management interface consists of logic cards housed in the MP-32 chassis. The main functions of the memory interface include the following:

- Provide logic and control necessary to interface to a large multi-port, multi-bank external main memory.
- Provide addressing capability to 4 million words of memory.
- Decouple memory references from CPU operations with random access, firmware-supported lookahead logic.
- Four-port interface supporting CYBER 18 DMA compatible devices.
- Provide paging, parity, fault detection and error reporting logic.
- Support full-word, half-word and character operations.

The memory management interface has four major components: lookahead holding registers, CPU data formatting, DMA data formatting and Page File. The general flow of a memory request is as follows:

- Memory address through Page File into one of the four lookahead address holding registers.
- Memory data through data formatter and into a write data lookahead register associated with an address holding register.
- Memory address and data are then sent to memory.
- Memory data is returned and formatted before being saved in a read data holding register until requested.

LOOKAHEAD REGISTERS

The purpose of lookahead is to decouple CPU and memory operations and to aid in producing minimum average memory cycle times. This is not a classical P+1 lookahead, but a random access scheme with addresses supplied by the firmware.

The lookahead consists of four pairs of read/write data and address output registers, four data input registers and associated control logic to manage register and data availability. These four sets of lookahead registers are divided between the CPU and DMA channels as follows: three for the CPU and one for the DMA.

The MP-60 emulator attempts to optimize memory references by using the three lookahead registers as follows:

- Operand fetching (reading memory);
- Operand storing (writing memory);
- Instruction prefetching (read memory).

It is important to note the MP-60 emulator overlaps instruction fetching with instruction execution and, therefore, instruction modifications at P+1 are not supported.

CPU DATA FORMATTING

The CPU data formatting logic supports the MP-60 emulator in performing full-word, half-word and character memory references. The data formatter positions the half-word (character), supplied by/to the CPU, into the proper place in the word.

PAGING

The MP-60 memory system is logically separated into pages, with each page containing 4096 consecutive memory locations. A fully expanded memory configuration contains 1024 pages or 4 million words. To facilitate memory management, a page index file is referenced by a program during every memory request to map the logical pages into physical pages and provide memory protection features. Figure 5-1 illustrates the relationship between physical memory pages and physical memory addresses. Thus, physical page 0 contains low-order addresses (0 through 4095), while physical page 1023 contains the highest possible address, 4,193,303.

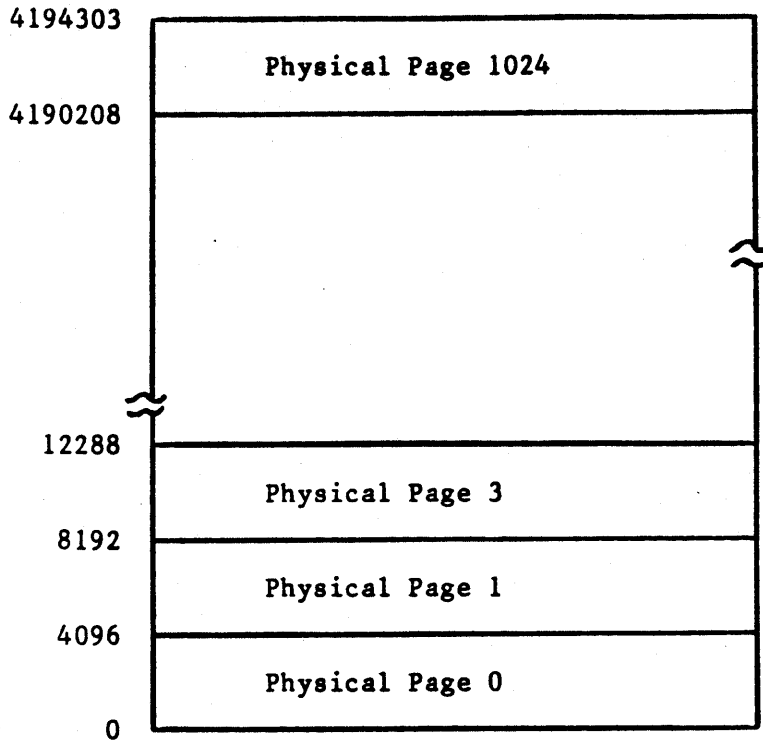


Figure 5-1. Memory Page Structure

PAGE INDEX FILE

The page index file is functionally divided into 32 distinct reference areas. One area is associated with each possible program state. Each reference area within the page index file consists of sixteen 16-bit page index registers. This provides each program state with exclusive use of 16 of these registers. The upper four bits of a program address are concatenated with the program state register to specify which of the 512 page index registers contains the physical page map. Figure 5-2 gives the page index file addressing format.

PAGE INDEX

Each page index register has the same basic format. Three of the 16 bits are used for memory protection indicators. Two bits are used for page status. One of these bits, modified, is set when an operand write is performed into the page. The other status bit, accessed, is set whenever an operand read or

write is performed on the page. The most significant bit is a parity bit for the 10 low-order bits. The 10 low-order bits contain the page address to be used in creating the physical address. Figure 5-3 shows the format of a page index register.

PROGRAM STATE REGISTER

This 5-bit register is used to define the portion of the page index file and operand register file which are currently valid. The contents of this register are modified by the emulator during interrupt recognition, CONT and MON instruction processing.

MEMORY ADDRESS GENERATION

Figure 5-4 shows how a physical memory address is generated using the paging system. The program state register addresses one of the thirty-two sections of the page index file. The upper four bits of the operand address specify one of the 16 page indexes within the section. The contents of this page index register contain the address of one of the 1024 possible pages. The lower 12 bits of the operand address specifies a location within the 4096 word page.

MEMORY ERRORS AND PROTECTION

Any physical page may be assigned as non-resident, fully protected or read only for a given program state. This is done by setting the proper bit within the page index which addresses the physical page.

An attempt to write into a read only page will result in a page fault 1 interrupt. A page fault 2 interrupt is generated when a memory location within a protected page is referenced as either an instruction or operand. A reference to a non-resident page results in a page fault 3 interrupt. An attempt to reference a non-existent physical page results in a memory reject interrupt, while reading a memory word with a parity error results in generating a parity error interrupt. Memory locations 6D and 6E point to a set of memory locations used as depicted in Figure 5-5, Memory Error Table.

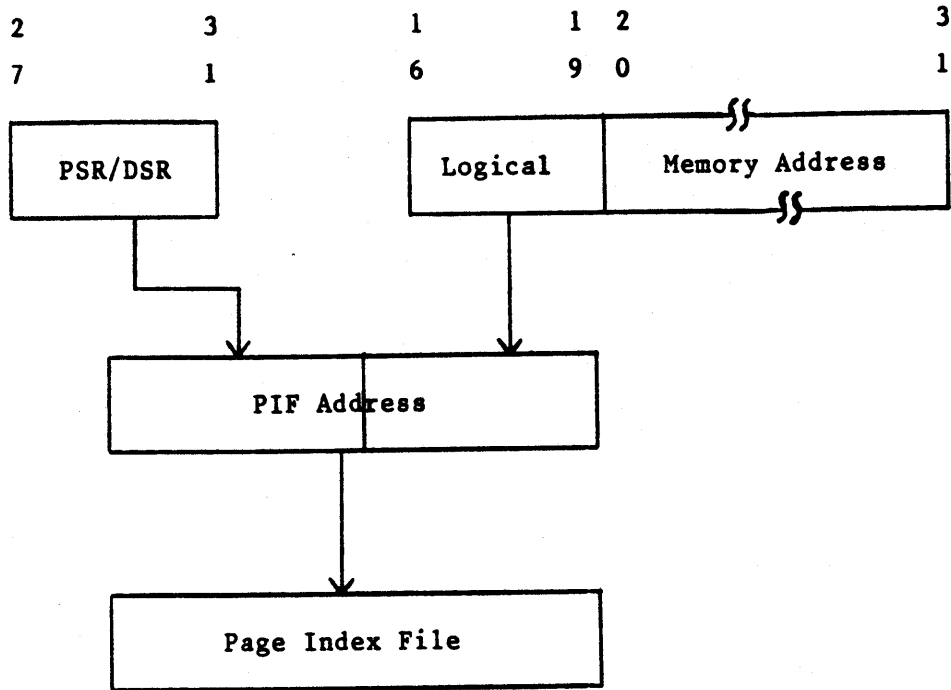


Figure 5-2. Page Index File Addressing

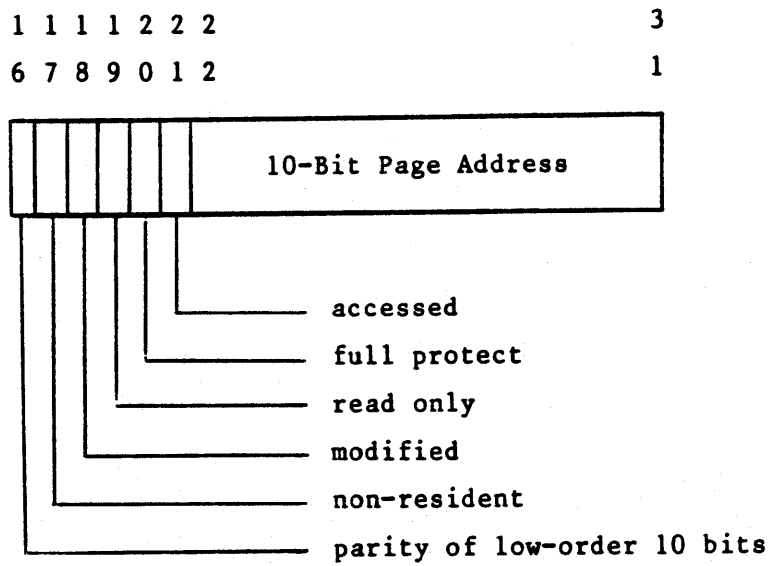


Figure 5-3. Page Index Register

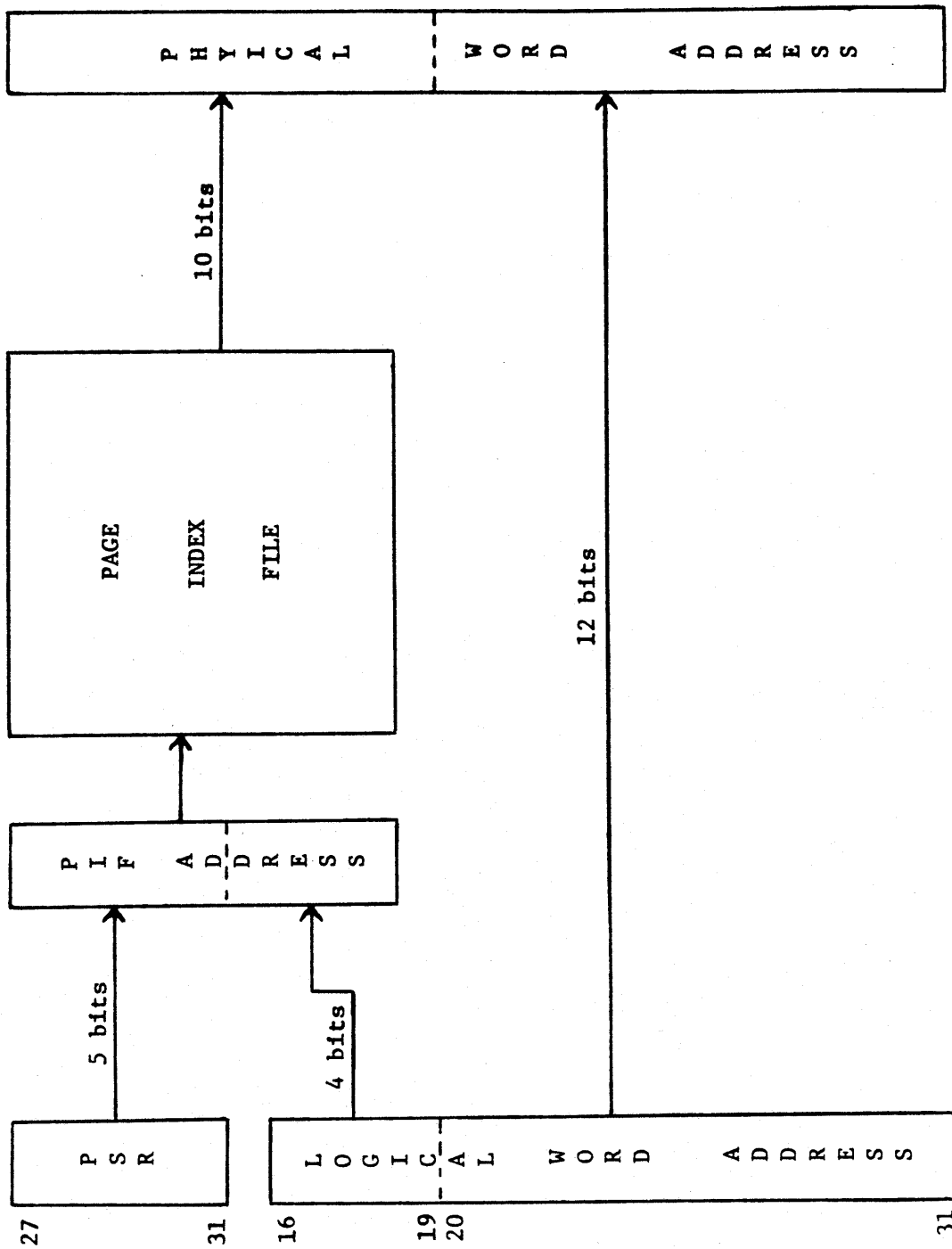


Figure 5-4. Memory Address Generation

| | | |
|----|--------------------|------------------|
| 0 | CPU STATE REGISTER | |
| 1 | DMA 0 STATE REG. | DMA 1 STATE REG. |
| 2 | DMA 2 STATE REG. | DMA 3 STATE REG. |
| 3 | DMA SNAPSHOT | |
| 4 | LA 1 SNAPSHOT | |
| 5 | LA 2 SNAPSHOT | |
| 6 | LA 3 SNAPSHOT | |
| 7 | PAGE FAULT 3 TRAP | |
| 8 | PAGE FAULT 2 TRAP | |
| 9 | PAGE FAULT 1 TRAP | |
| 10 | PARITY ERROR TRAP | |
| 11 | REJECT TRAP | |

Figure 5-5. Memory Error Table

See Figures 5-6 through 5-8 for CPU State Register, DMA State Register and Typical Address Snapshot.

DMA DATA FORMATTING

The DMA data formatting logic performs the function of assembling/disassembling the 16-bit DMA channel words into 32-bit memory data words. The DMA channels have two modes of operation, half-word and full-word. In the half-word mode, each DMA memory request results in a memory cycle, while the full-word mode makes only one memory request for two DMA memory requests. The half-word mode is intended to be used on DMA devices which perform random memory access or could terminate on an uneven transfer count. Mode selection is on a port and transfer basis under program control, making each port flexible in its operation and independent of all other ports.

In the half-word mode, the DMA port acts as if a 16-bit memory is being used. One memory cycle is initiated for each request from a DMA controller. To place any port in this mode, the half-word mode bit is set by the programmer in the DMA state register.

The full-word mode assembles or disassembles a 32-bit word on all read/write operations. The programmer must force all write transfers to an even number of 16-bit half words. All transfers must begin with the upper half word. This mode is selected by clearing the half-word mode bit in the DMA state register.

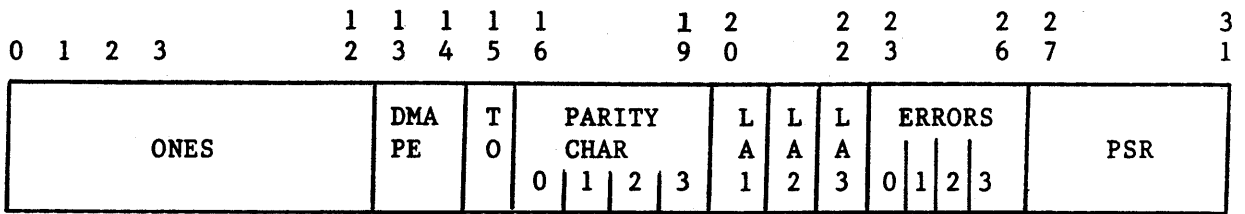
DMA STATE REGISTER

The four DMA ports have a corresponding DMA state register. These DMA state registers perform two functions, selecting the transfer mode and selecting the page file section. The format of a DMA state register is depicted in Figure 5-7. The mode control bits 23 through 26 are shared among the four DMA ports.

The mode control bits 23 through 26 are used to select the transfer mode of the DMA port. Bit 23 corresponds to port 0 and bit 26 to port 3. When the mode bit is a 1, the corresponding port is placed in half-word mode.

The DSR portion of the DMA state register is used instead of the PSR, when a DMA port has control of the paging logic, to select one of the 32 page index file sections. Each DMA port has unique DSR so that the DMA ports may access different sections of the page index file.

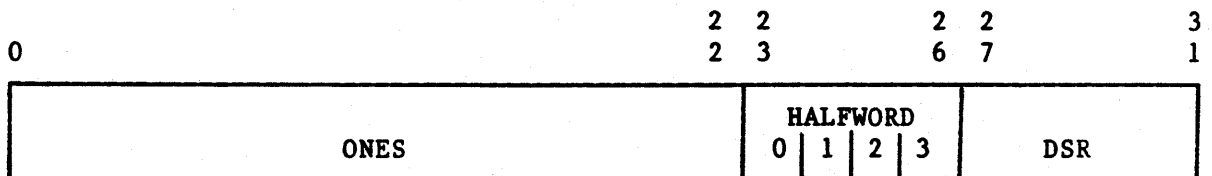
The DMA state registers may be read and written using the privileged RSR and WSR instruction.



BITS

- 0 - 12 Ones
- 13 - 14 DMA PORT IN ERROR
- 15 Time Out Error (Reject)
- 16 - 19 Character Parity Error Flags
- 20 - 22 010
- 23 - 26 Discrete Error Flags
 LA 0 - 3
- 27 - 31 Program State Register

Figure 5-6. CPU State Register



BITS

- 0 - 22 Ones
- 23 - 26 Halfword/fullword mode selections
 One per port
- 27 - 31 DMA State Register used like PSR during
 DMA page file accesses

Figure 5-7. DMA State Register

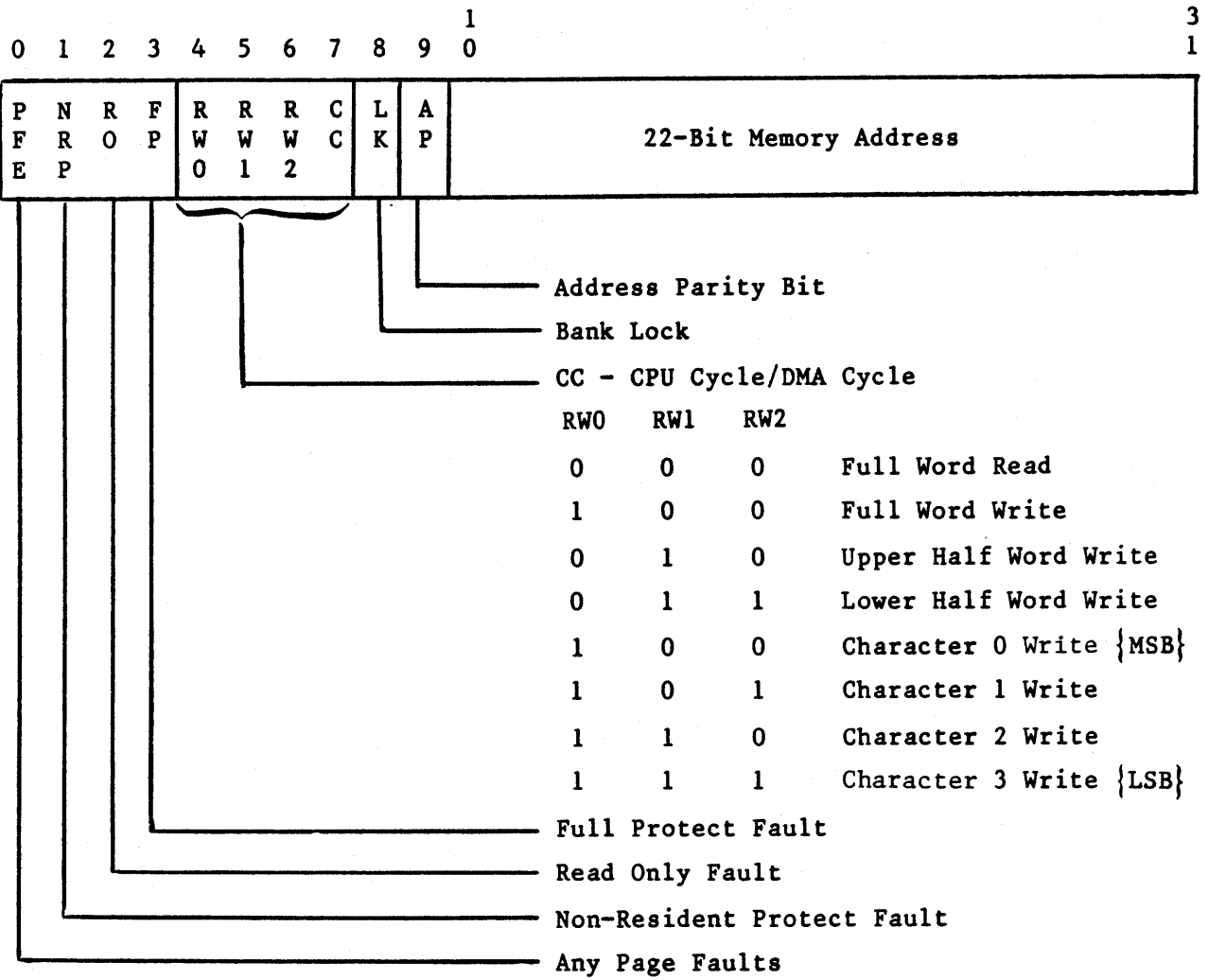


Figure 5-8. Typical Address Snapshot

INSTRUCTION FORMATS

The MP-60 machine coded instruction is 32 bits in length and organized in various formats. An instruction occupies one memory word. Instructions are classified as full word, half-word, byte, or bit oriented, as having a relative displacement address, or as being three-address-file oriented.

Word address instructions consist of 16 bits allocated for an unmodified storage address, immediate operand, or shift count. Figure 6-1 illustrates a word address instruction.

Bit Position

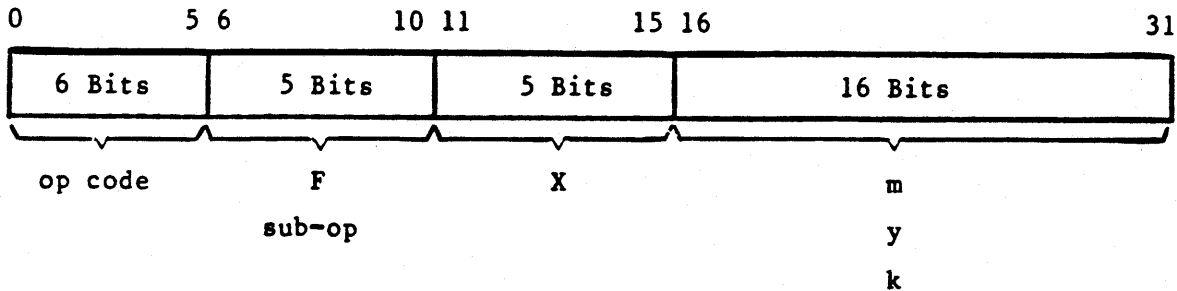


Figure 6-1. Word Addressed Instruction Format

Half-word address instructions consist of 17 bits allocated for an unmodified storage address or immediate operand. Address modification is available for these instructions; however, the range of permissible indexes is limited to specific registers in the file. Figure 6-2 illustrates the format of the half-word oriented instruction word.

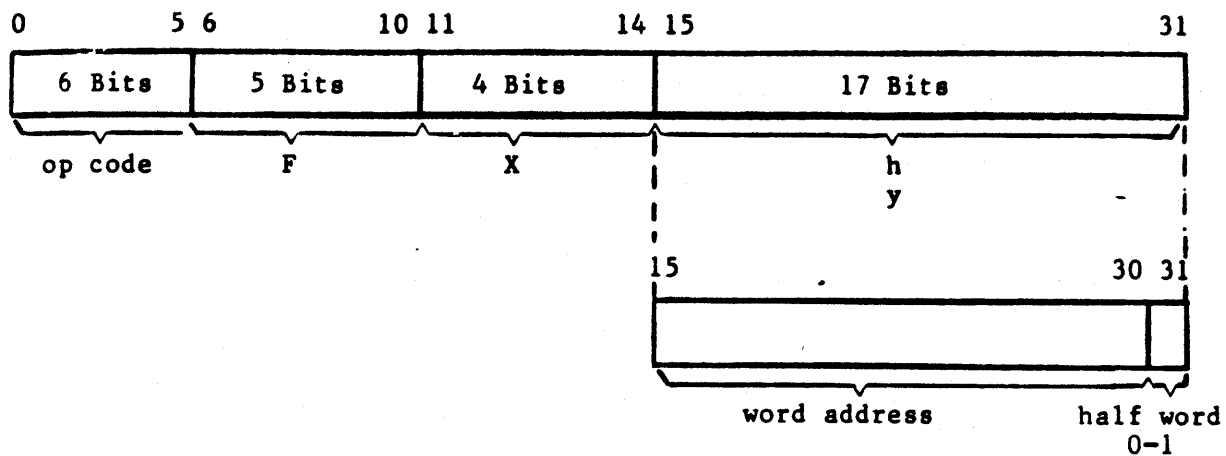
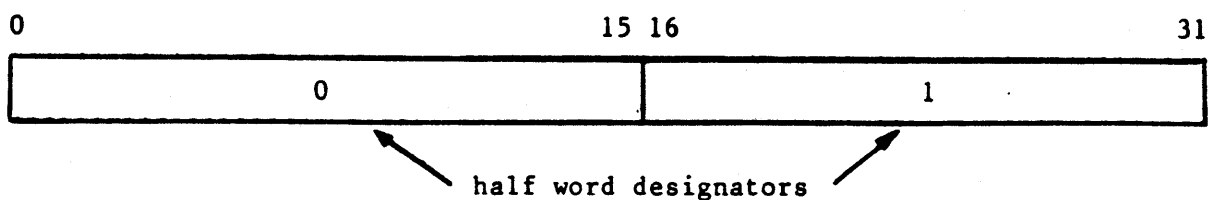


Figure 6-2. Half Word Addressed Instruction Format

Half words in a data word are always specified in the following manner:



Byte address instructions consist of 18 bits allocated for an unmodified storage address or immediate operand. Address modification is available for these instructions; however, the range of permissible indexes is limited to specific registers in the file. Figure 6-3 illustrates the format of the byte oriented instruction word. A byte is referred to as a character.

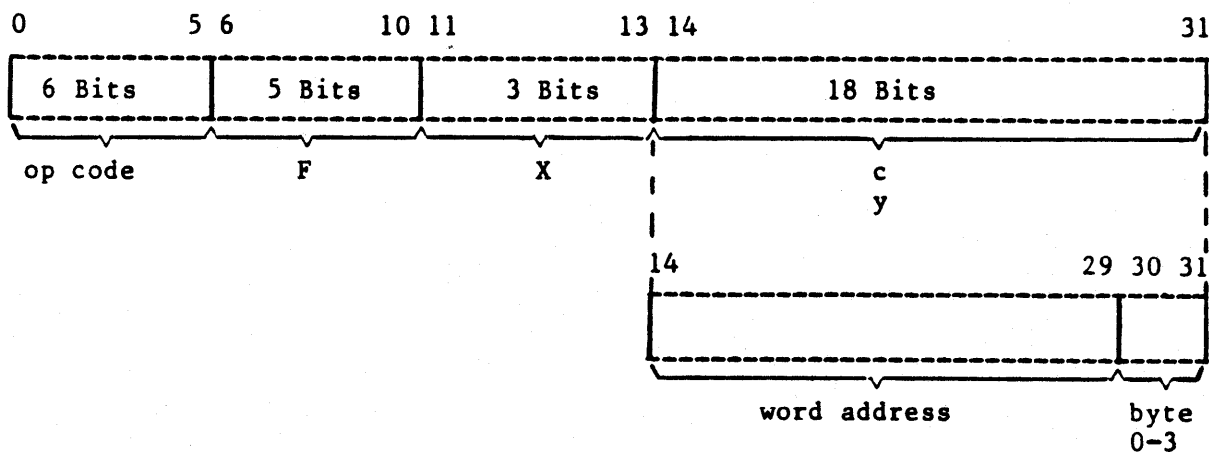
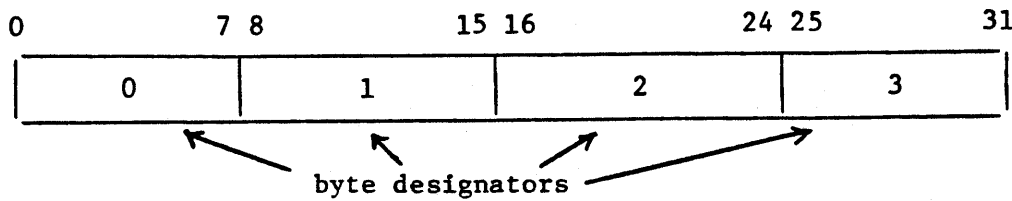


Figure 6-3. Byte Addressed Instruction Format

Byte or characters in a data word are always specified in the following manner:



Bit address instructions consist of 21 bits allocated for an unmodified storage address or immediate operand. Address modification is available for bit address instructions with the data being stored in the bit register; otherwise, an unmodified 21-bit immediate operand is available. Figure 6-4 illustrates the format of these instructions.

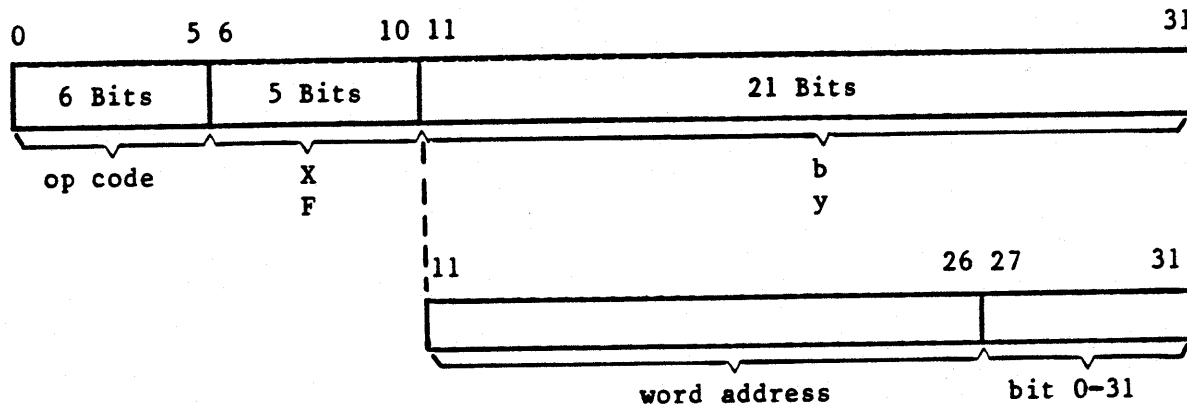
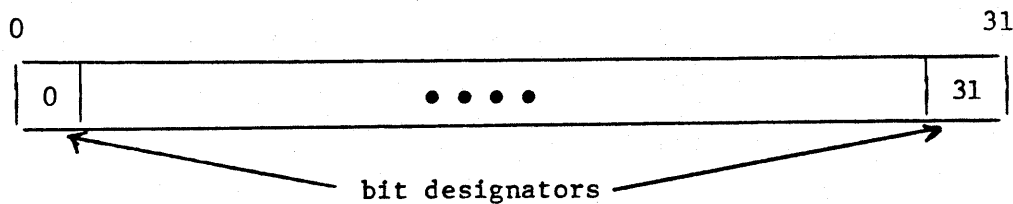


Figure 6-4. Bit Addressed Instruction Format

Bits in a data word are always specified in the following manner:



Relative displacement instructions consist of 11 bits allocated for an unmodified relative storage address. The displacement is sign extended and added to the current contents of the P register to generate the referenced address. Thus, an addressing range of +1023 to -1024 locations can be referenced by this type of instruction. Figure 6-5 illustrates the format of these instructions.

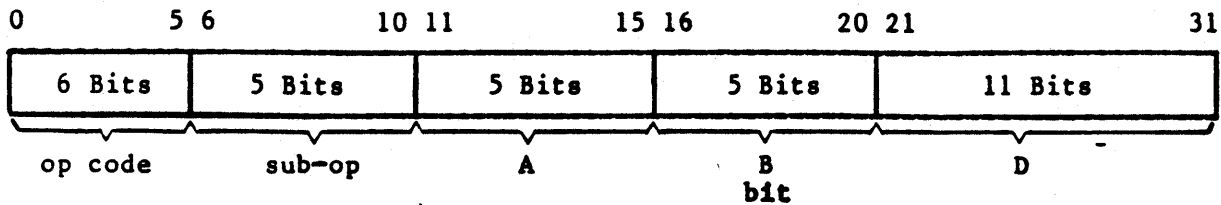


Figure 6-5. Relative Displacement Instruction Format

Three address instructions consist of three operand designators of five bits each. Two of the designators (A and B) are operand sources and the third (C) is the result destination. All operations are to and from the register file. Figure 6-6 illustrates the format of these instructions.

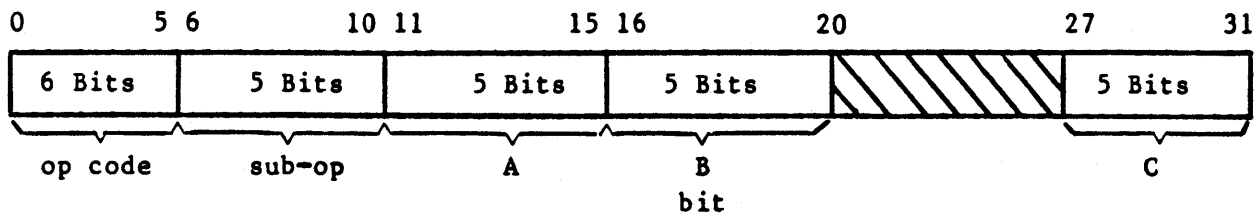


Figure 6-6. Three Address Instruction Format

SYMBOL DEFINITIONS

The following designators are used throughout the list of instructions:

- A = file register A, specifying one of the 32 registers in the file.
- b = base bit address.
- B = effective bit address, after indexing; or file register B, specifying one of the 32 registers in the file.
- BIT = bit number 0-31.
- BR = bit register.
- c = base byte or character address.
- C = effective byte or character address, after indexing; or file register C, specifying one of the 32 registers in the file.

D = displacement; relative address class instructions
 F = file register F, specifying the operand register
 h = base half word address
 H = effective half word address, after indexing
 k = unmodified shift count
 K = effective shift count, after indexing
 m = base word address
 M = effective word address, after indexing
 P = P register, program address counter
 y = base immediate operand
 Y = effective immediate operand, after indexing
 X = index designator, specifying one of the 32 registers in the file

INDEXING AND ADDRESS MODIFICATION

In some instructions, the operand address m , h , c or b , the immediate operand, y , or the shift count, k , is modified by adding to the base, or unmodified value, the contents of an index register, X . For these types of instructions, the contents of one of the 32 registers in the file is always added to form the operand address or immediate operand. Thus, file register 0 should always contain the value zero, since an X field of zero points to file register 0. Operationally, the programmer may find it convenient to clear both file registers 0 and 1 to zero and use one to provide a source of constant zero in both single and double precision modes. Symbols representing the respective modified quantities are as follows:

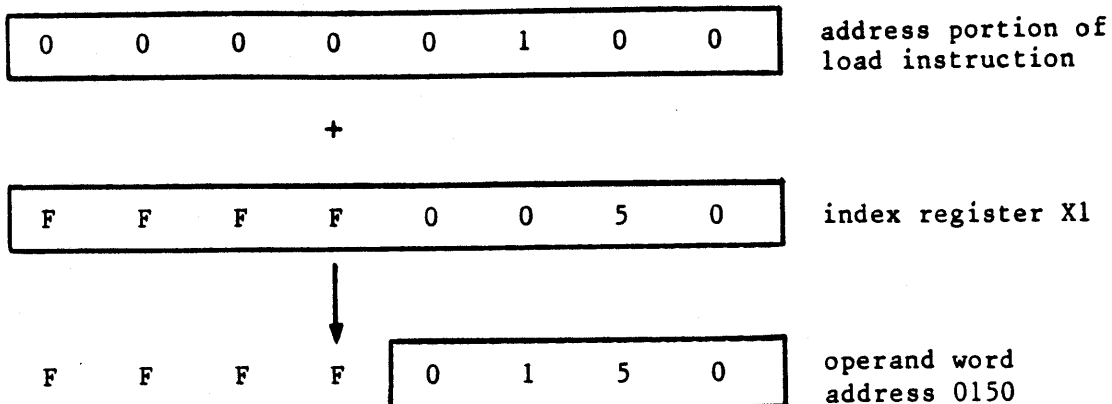
$M = m + (X)$, 16-bit result
 $H = h + (X)$, 17-bit result
 $C = c + (X)$, 18-bit result
 $B = b + (X)$, 21-bit result
 $Y = y + (X)$, 16, 17, 18 or 21-bit result
 $K = k + (X)$, 8-bit result

In each case, the contents of X is a full 32-bit value and the base value is the length of the result.

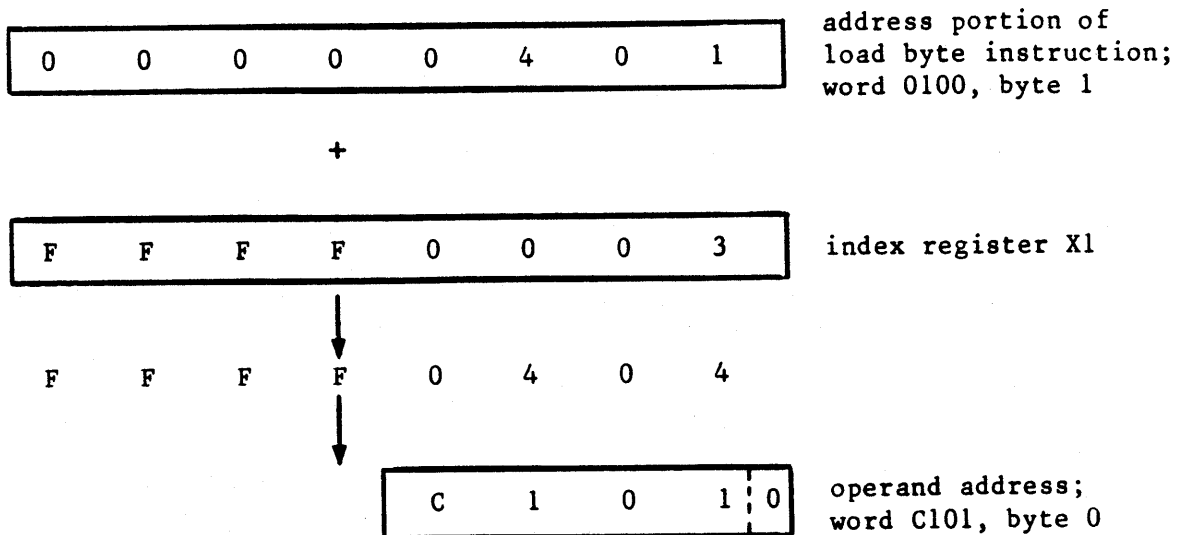
All index registers are 32 bits in length; however, only the lower order bits, consistent with the addressing mode, are significant. Thus, the effective address of an instruction, base address plus index, is always in the range of 0000 through 65,535 words of memory.

Examples:

LD, R0 0100, X1 (X1) = FFFF0050



LDC, R0 00401, X1 (X1) = FFFF0003



In this case, certain upper order bits of the index register were significant because byte addressing consists of 18 bits (refer to Figure 6-3). An index value of FFFC003, for this example, would have effectively added 3 to the operand address or word 0101, byte 0.

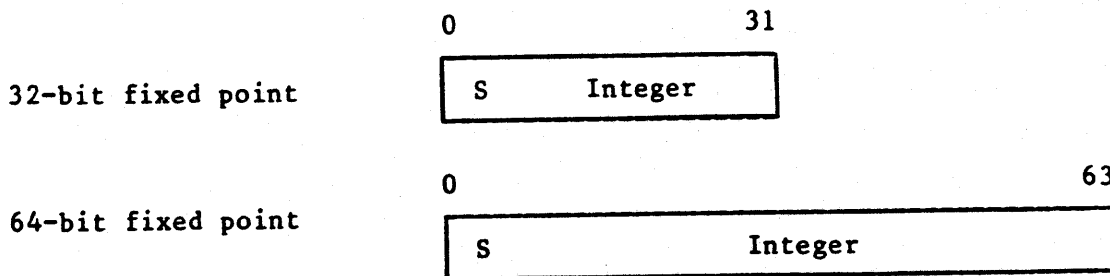
All addressing operations yield a 16-bit word address and possibly a 1, 2 or 5-bit half-word, byte or bit designator. The same comment applies to jump-type instructions; while indexing on a jump produces a 32-bit result and all 32 bits are gated to the P register, only the lower 16 bits of the P register are significant as instruction addresses.

USE OF REGISTERS

All 32 registers in a machine state may be used to contain operands or indexes, except as noted for specific addressing modes. Register 0 of a state must contain zeros in at least the lower 21 bits because an index field of zero references register 0. However, this register may be used as one source of a test or register operation instruction. Any register may be used to specify the most significant part of a double precision operation. If a double precision operand destination specifies register 31 as the most significant portion, register 0 of the adjacent program state is destroyed. However, when a state change occurs in the MP-60, register 0 of the new state is unconditionally cleared. Thus, programs in the various program states are protected from programming errors.

NUMBER REPRESENTATION

Fixed point numbers in the MP-60 occupy 32 or 64 bits of memory (one or two words). All fixed point operands are expressed as signed integers. Positive numbers are represented in true binary notation with the sign bit set to 0. Negative numbers are represented in two's complement notation with the sign bit set to 1.



Two's complement notation includes one more negative number than positive numbers; 80 - 0. However, this number has no positive complement. Thus, subtracting this number from zero yields the number itself and an overflow fault.

NO OPERATION

| Operation Field | Address Field | Interpretation |
|-----------------|---------------|----------------|
| NOP,F | m,X | No Operation |

NOP No Operation

0 5 6 10 11 15 16 31

| | | | |
|----|---|---|---|
| 00 | F | X | m |
|----|---|---|---|

F = operand register

X = index register

m = base word address; M = m + (X)

Description: No operation is performed.

LOAD

| Operation Field | Address Field | Interpretation |
|-----------------|---------------|--------------------------------|
| LDB | b,X | load bit register |
| LDC,F | c,X | load byte or character |
| LDH,F | h,X | load half word |
| LDI,F | y,X | load immediate |
| LDHA,F | h,X | load half word address |
| LDCA,F | c,X | load byte or character address |
| LDBA,F | b | load bit address |
| LDA,F | m,X | load word address |
| LD,F | m,X | load word |
| LDD,F | m,X | load double word |
| DLDF,F | m,X | destructive load |
| LDP,F | m,X | load paged |
| LDM,F | n,D | load multiple |
| LDF | A,B,C | load field |

LDB Load Bit 0 5 6 10 11 31



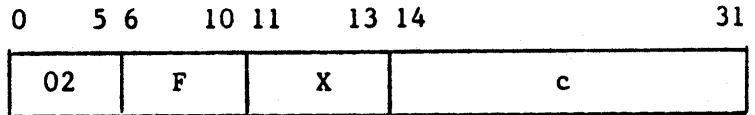
X = index register

b = base bit address; B = b + (X)

Description: Load the bit register BR with the value of the bit from memory specified by B.

BR = (B)

LDC Load Byte



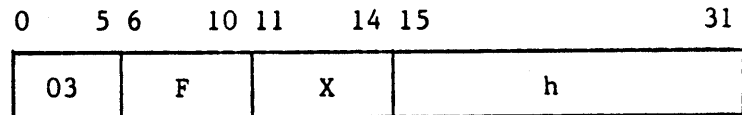
F = operand register
X = index register (0-7)
c = base byte address; C = c + (X)

Description: Load the operand register F with the byte from memory specified by C. The byte is loaded into the lower eight bits of F (24-31) with zero fill.

$$F = (C)$$

Note: Only registers 0-7 can be used for indexing in an LDC instruction.

LDH Load Half Word



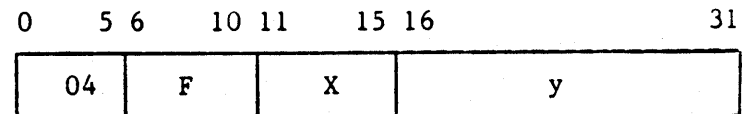
F = operand register
X = index register (0-F)
h = base half word address; H = h + (X)

Description: Load the operand register F with the half word from memory specified by H. The half word is loaded into the lower 16 bits of F (16-31) with zero fill.

$$F = (H)$$

Note: Only registers 0-F can be used for indexing in an LDH instruction.

LDI Load Immediate



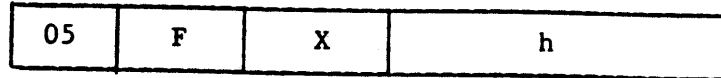
F = operand register
X = index register
y = immediate operand; Y = y + (X)

Description: Load the operand register F with the operand Y; y is sign extended to 32 bits before generating Y. Bit 16 of y is the sign bit.

$$F = Y$$

LDHA Load Half Word Address

0 5 6 10 11 14 15 31



F = operand register

X = index register (0-F)

h = base half word address; H = h + (X)

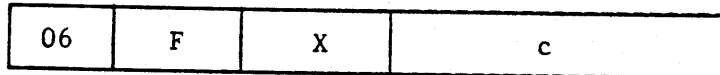
Description: Load the operand register F with the 17-bit operand H; the base operand h is zero extended before the index is added. However, the 32-bit result is loaded into F.

$$F = H$$

Note: Only registers 0-F can be used for indexing in an LDHA instruction.

LDCA Load Byte Address

0 5 6 10 11 13 14 31



F = operand register

X = index register (0-7)

c = base byte address; C = c + (X)

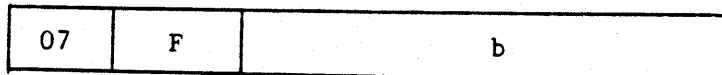
Description: Load the operand register F with the 18-bit operand C; the base operand c is zero extended before the index is added. However, the 32-bit result is loaded into F.

$$F = C$$

Note: Only registers 0-7 can be used for indexing in an LDCA instruction.

LDBA Load Bit Address

0 5 6 10 11 31

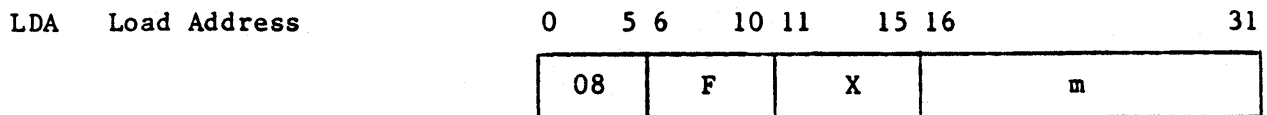


F = operand register

b = base bit address

Description: Load the operand register F with the 21-bit operand b. The operand is loaded into the lower 21 bits of F (11-31) with zero fill. Indexing is not allowed.

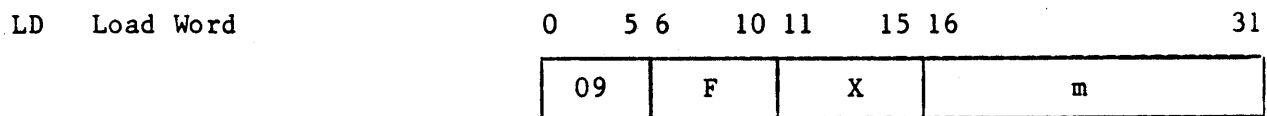
$$F = b$$



F = operand register
 X = index register
 m = base word address; M = m + (X)

Description: Load the operand register F with the 16-bit operand m; the base operand is zero extended after the index is added and the 32-bit result is loaded into F.

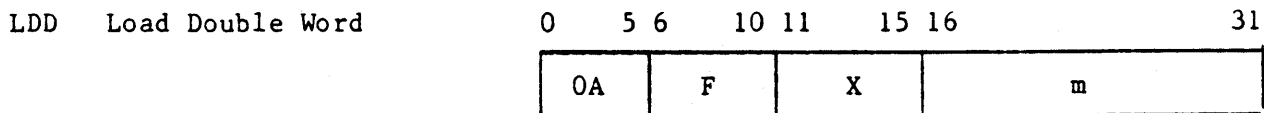
$$F = M$$



F = operand register
 X = index register
 m = base word address; M = m + (X)

Description: Load the operand register F with the word from memory specified by M.

$$F = (M)$$

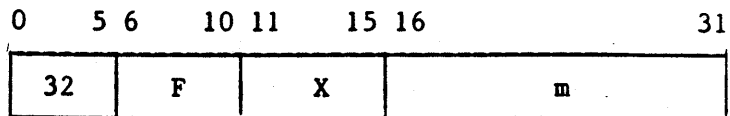


F = operand register
 X = index register
 m = base word address; M = m + (X)

Description: Load the operand register F with the word from memory specified by M; load F+1 with the word from memory specified by M+1. Location M represents the most significant portion of the 64-bit operand; M+1 represents the least significant portion.

$$F, F+1 = (M, M+1)$$

DLD Destructive Load

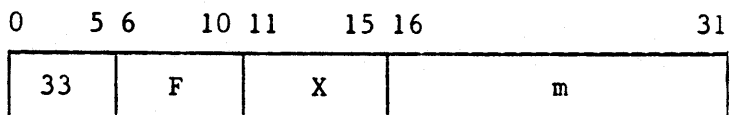


F = operand register
X = index register
m = base word address; $M = m + (X)$

Description: Load the operand register F with the word from memory specified by M. The contents of M are set to ones.

$$F = (M), M = \text{FFFFFFFF}$$

LDP Load Paged

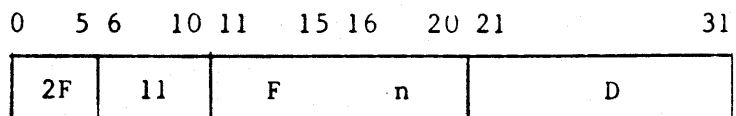


F = operand register
X = index register
m = base word address; $M = m + (X)$

Description: Load the operand register F with the word from memory specified by M, relocated to the state specified by the contents of the relocation register. The relocation register must have previously been initialized with an SST instruction. The word loaded from memory is specified by the contents of the page index register indicated by M and the referenced state.

$$F = (M)$$

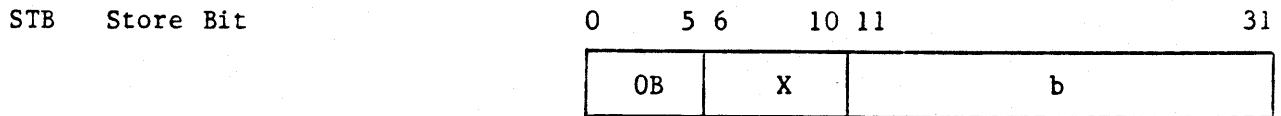
LDM Load Multiple



Description: Multiple registers are loaded starting at $M = (P) + 1 + D$ into register F and continuing with consecutive memory locations until $F+n$ is loaded.

STORE

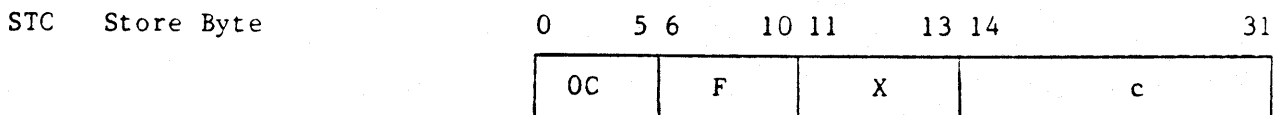
| Operation Field | Address Field | Interpretation |
|-----------------|---------------|-------------------------|
| STB | b,X | store bit |
| STC,F | c,X | store byte or character |
| STH,F | h,X | store half word |
| STHA,F | m,X | store half word address |
| STCA,F | m,X | store byte address |
| STBA,F | m,X | store bit address |
| ST,F | m,X | store word |
| STD,F | m,X | store double word |
| STP,F | m,X | store paged |
| STM,F | n,D | store multiple |
| STF | A,B,C | store field |



X = index register
 b = base bit address; B = b + (X)

Description: Store the contents of the bit register BR, 0 or 1, in the bit of memory specified by B. Other bits of the word in memory are not modified.

$$B = (BR)$$

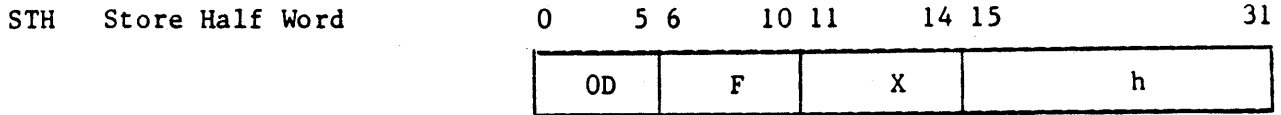


F = operand register
 X = index register (0-7)
 c = base byte address; C = c + (X)

Description: Store the lower eight bits of register F (bits 24-31) in the byte of memory specified by C. Other bytes of the word in memory are not modified.

$$C = (F)_{24-31}$$

Note: Only registers 0-7 can be used for indexing in an STC instruction.

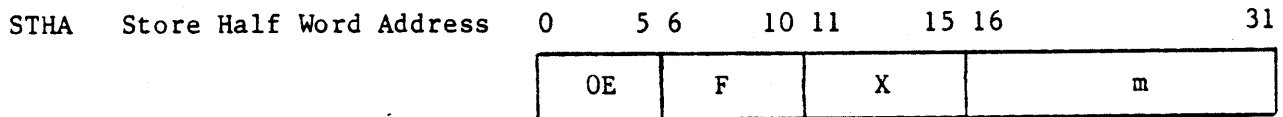


F = operand register
 X = index register (0-F)
 h = base half word address; H = h + (X)

Description: Store the lower 16 bits of register F (bits 16-31) in the half of the memory word specified by H. The other half of the word in memory is not modified.

$$H = (F)_{16-31}$$

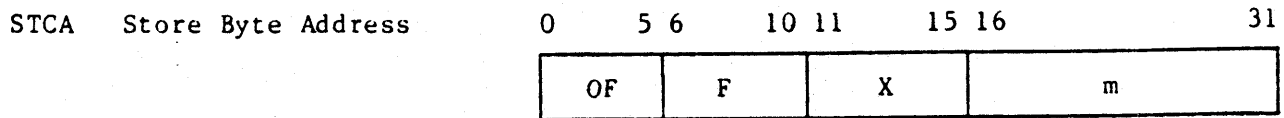
Note: Only registers 0-F can be used for indexing in an STH instruction.



F = operand register
 X = index register
 m = base word address; M = m + (X)

Description: Store the lower 17 bits of register F (bits 15-31) in the lower 17 bits of the memory word specified by M. The upper bits of the word in memory are not modified.

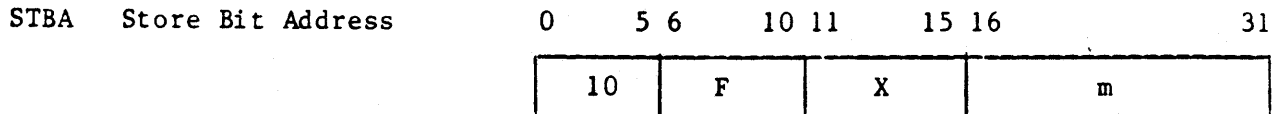
$$M_{15-31} = (F)_{15-31}$$



f = operand register
 x = index register
 m = base word register; M = m + (X)

Description: Store in the lower 18 bits of register F (bits 14-31) in the lower 18 bits of the memory word specified by M. The upper bits of the word in memory are not modified.

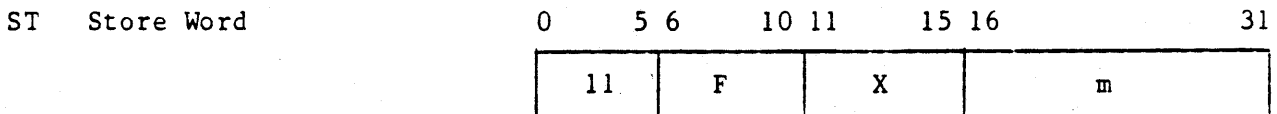
$$M_{11-31} = (F)_{14-31}$$



F = operand register
 X = index register
 m = base word address; M = m + (X)

Description: Store the lower 21 bits of register F (bits 11-31) in the lower 21 bits of the memory word specified by M. The upper bits of the word in memory are not modified.

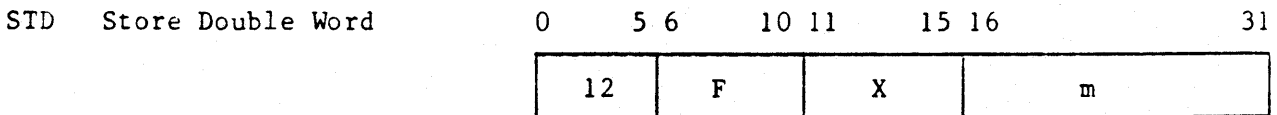
$$M_{11-31} = (F)_{11-31}$$



F = operand register
 X = index register
 m = base word address; M = m + (X)

Description: Store the contents of register F in the memory word specified by M.

$$M = (F)$$

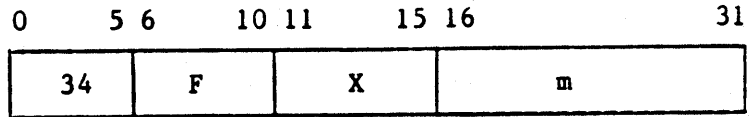


F = operand register
 X = index register
 m = base word address; M = m + (X)

Description: Store the contents of register F in the memory word specified by M; store the contents of F + 1 in the memory word specified by M + 1. Register F contains the most significant portion of the 64-bit operand and F + 1 contains the least significant portion.

$$M, M+1 = (F, F+1)$$

STP Store Paged

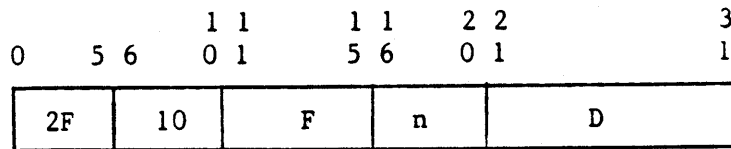


F = operand register
 X = index register
 m = base word address; $M = m + (X)$

Description: Store the contents of register F in the memory word specified by M, relocated to the state specified by the contents of the relocation register. The relocation register must have previously been initialized with an SST instruction. The word of memory is specified by the contents of the page index register indicated by M and the referenced state.

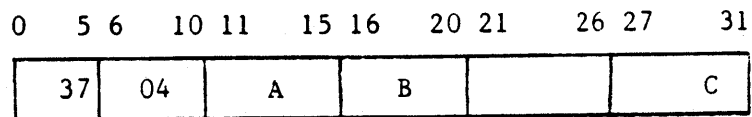
$$M = (F)$$

STM Store Multiple



Description: Multiple registers are stored starting at $M = (P) + D + 1$ with register F and continuing in consecutive memory locations until register $F + n$ is stored.

STF Store Field



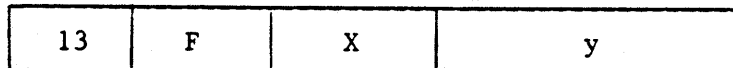
- (a) Store field
- (b) TO bit address
- (c) Number of bits in field ($1 < C < 32$)

Description: Store the contents of register A into the bit address specified in register B for the number of bits specified in register C. Data in register A is assumed to be right justified, zero filled.

FIXED POINT ARITHMETIC

| Operation Field | Address Field | Interpretation |
|-----------------|---------------|----------------------------|
| ADI, F | y, X | add immediate |
| AD, F | m, X | add |
| ADD, F | m, X | add double |
| SB, F | m, X | subtract |
| SBD, F | m, X | subtract double |
| MPI, F | y, X | multiply immediate |
| MP, F | m, X | multiply |
| DV, F | m, X | divide |
| RAD, F | m, X | replace add |
| MPS, F | m, X | multiply, single precision |
| DVS, F | m, X | divide, single precision |

ADI Add Immediate 0 5 6 10 11 15 16 31

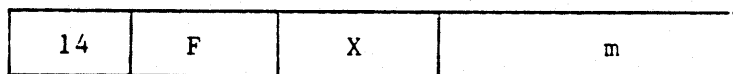


F = operand register
X = index register
y = base operand; Y = y + (X)

Description: Add the contents of register X to operand y and add the result to F; y is sign extended to 32 bits before generating Y. Bit 16 of y is the sign bit.

$$F = (F) + Y$$

AD Add 0 5 6 10 11 15 16 31

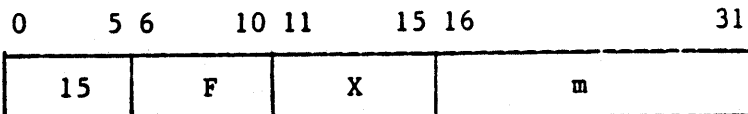


F = operand register
X = index register
m = base word address; M = m + (X)

Description: Add the contents of register F and the contents of memory specified by M and store the result in F.

$$F = (F) + (M)$$

ADD Add Double

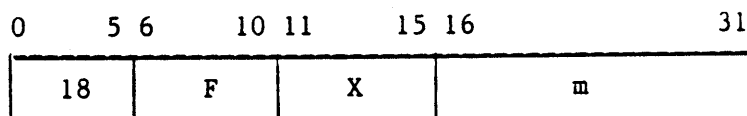


F = operand register
X = index register
m = base word address; M = m + (X)

Description: Add the contents of registers F and F+1 to the 64-bit quantity specified by M and M+1. The 64-bit result is stored in registers F and F+1.

$$F, F+1 = (F, F+1) + (M, M+1)$$

SB Subtract

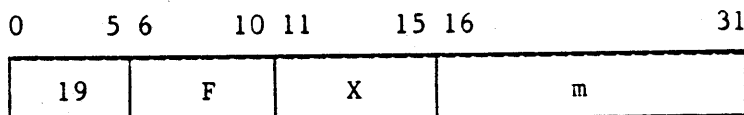


F = operand register
X = index register
m = base word address; M = m + (X)

Description: Subtract the contents of memory specified by M from the contents of register F and store the results in F.

$$F = (F) - (M)$$

SBD Subtract Double

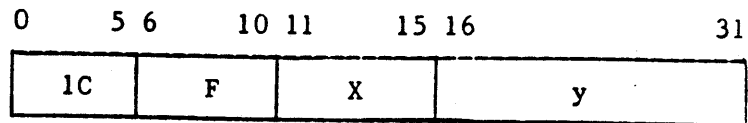


F = operand register
X = index register
m = base word address; M = m + (X)

Description: Subtract the 64-bit quantity specified by M and M+1 from the contents of registers F and F+1. The 64-bit result is stored in F and F+1.

$$F, F+1 = (F, F+1) - (M, M+1)$$

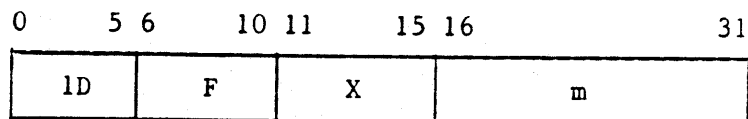
MPI Multiply Immediate



F = result register
X = operand register
y = immediate operand

Description: Multiply the contents of register X by y and store the result in F. The operands are treated as unsigned 16-bit quantities yielding an unsigned 16-bit result.

MP Multiply

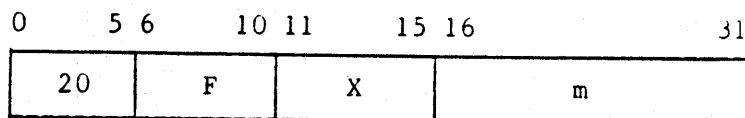


F = operand address
X = index register
m = base word address; $M = m + (X)$

Description: Multiply the contents of register F by the contents of memory specified by M and store the 64-bit result in registers F (most significant part) and F+1 (least significant).

$$F, F+1 = (F) * (M)$$

DV Divide

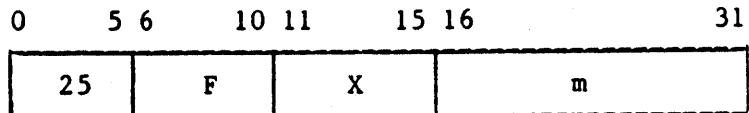


F = operand register
X = index register
m = base word address; $M = m + (X)$

Description: Divide the 64-bit quantity contained in registers F and F+1 by the contents of memory specified by M. The quotient is stored in F and the remainder in F+1.

$$F = (F, F+1) / (M), F+1 = \text{remainder}$$

RAD Replace Add

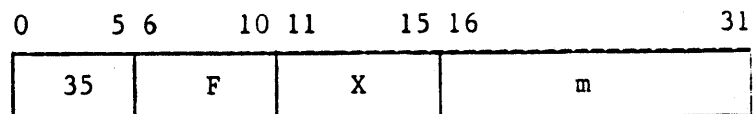


F = operand register
X = index register
m = base word address; M = m + (X)

Description: Add the contents of register F and the contents of memory specified by M and store the result in M. The contents of F are unmodified.

$$M = (F) + (M)$$

MPS Multiply, Single Precision

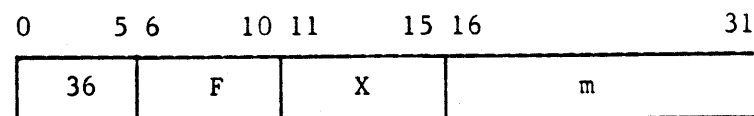


F = operand register
X = index register
m = base word address; M = m + (X)

Description: Multiply the contents of register F by the contents of memory specified by M and store the least significant 32 bits of the result in F.

$$F = (F) * (M)$$

DVS Divide, Single Precision



X = operand register
X = index register
m = base word address; M = m + (X)

Description: Divide the contents of register F, sign extended to 64 bits, by the contents of memory specified by M and store the quotient in register F.

$$F = (F)/(M)$$

coefficient of such numbers is C0-0 in hexadecimal. The exponent of such a result is adjusted to represent the true value of the number.

Exponent

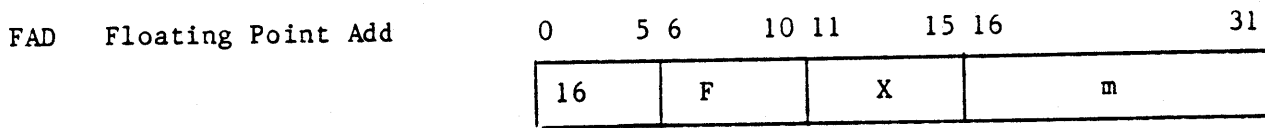
The exponent is expressed as a true biased 8-bit quantity with a value ranging from 00₁₆ to FF₁₆. Positive exponents range from 80₁₆ to FF₁₆ and negative exponents from 00₁₆ to 7F₁₆.

The representation of floating point numbers is similar to scientific notation, that is, a fraction multiplied by a number raised to a power.

$F * 2^E$ where: F = fraction

E = exponent

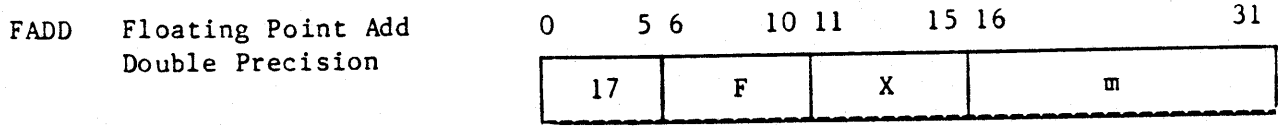
Note: The value zero is represented by a data value of all zeros.



F = operand register
 X = index register
 m = base word address; M = m + (X)

Description: Add the contents of register F and the contents of memory specified by M in floating point format and store the result in F.

$$F = (F) + (M)$$



F = operand register
 X = index register
 m = base word address; M = m + (X)

Description: Add the double precision contents of register F and F+1 and the contents of memory specified by M and M+1. The result is stored in registers F and F+1.

$$F, F+1 = (F, F+1) + (M, M+1)$$

| | | | | | | |
|-----|-------------------------|----|-----|-------|-------|----|
| FSB | Floating Point Subtract | 0 | 5 6 | 10 11 | 15 16 | 31 |
| | | 1A | F | X | m | |

F = operand register
 X = index register
 m = base word address; M = m + (X)

Description: Subtract the contents of memory specified by M from the contents of register F and store the result in F.

$$F = (F) - (M)$$

| | | | | | | |
|------|---|----|-----|-------|-------|----|
| FSBD | Floating Point Subtract Double Precision | 0 | 5 6 | 10 11 | 15 16 | 31 |
| | | 1B | F | X | m | |

F = operand register
 X = index register
 m = base word address; M = m + (X)

Description: Subtract the double precision contents of memory specified by M and M+1 from the contents of registers F and F+1. The result is stored in registers F and F+1.

$$F, F+1 = (F, F+1) - (M, M+1)$$

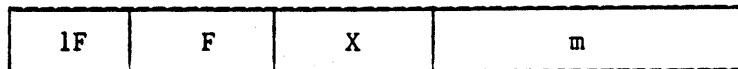
| | | | | | | |
|-----|-------------------------|----|-----|-------|-------|----|
| FMP | Floating Point Multiply | 0 | 5 6 | 10 11 | 15 16 | 31 |
| | | 1E | F | X | m | |

F = operand register
 X = index register
 m = base word address; M = m + (X)

Description: Multiply the contents of register F by the contents of memory specified by M and store the result in F.

$$F = (F) * (M)$$

FMPD Floating Point Multiply Double Precision 0 5 6 10 11 15 16 31

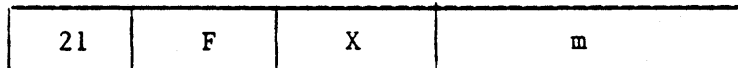


F = operand register
 X = index register
 m = base word address; M = m + (X)

Description: Multiply the double precision contents of registers F and F+1 by the contents of memory specified by M and M+1. The result is stored in F and F+1.

$$F, F+1 = (F, F+1) * (M, M+1)$$

FDV Floating Point Divide 0 5 6 10 11 15 16 31

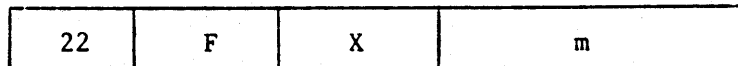


F = operand register
 X = index register
 m = base word address; M = m + (X)

Description: Divide the contents of register F by the contents of memory specified by M and store the result in F.

$$F = (F)/(M)$$

FDVD Floating Point Divide Double Precision 0 5 6 10 11 15 16 31



F = operand register
 X = index register
 m = base word address; M = m + (X)

Description: Divide the double precision contents of registers F and F+1 by the contents of memory specified by M and M+1. The result is stored in registers F and F+1.

$$F, F+1 = (F, F+1) / (M, M+1)$$

SHIFT

| Operation Field | Address Field | Interpretation |
|-----------------|---------------|----------------|
| SF, F | k, X | shift |
| SFD, F | k, X | shift double |

| | | | | | | |
|----|-------|----|-----|-------|-------|----|
| SF | Shift | 0 | 5 6 | 10 11 | 15 16 | 31 |
| | | 23 | F | X | k | |

F = operand register
 X = index register
 k = unmodified shift count; K = k+(X)

Description: Shift the contents of register F the number of places specified by the low order eight bits of K. The quantities k and (X), with the sign of k extended, are added to obtain K. The sign of K is tested. If positive, the quantity in F is shifted left circular; otherwise, it is shifted right end-off with sign extension.

| | | | | | | |
|-----|--------------|----|-----|-------|-------|----|
| SFD | Shift Double | 0 | 5 6 | 10 11 | 15 16 | 31 |
| | | 24 | F | X | k | |

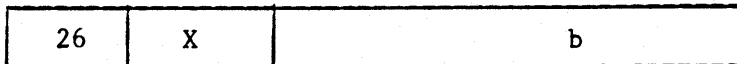
F = operand register
 X = index register
 k = unmodified shift count; K = k+(X)

Description: Shift the contents of registers F and F+1 the number of places specified by the low order eight bits of K. The quantities k and (X), with the sign of k extended, are added to obtain K. The sign of K is tested. If positive, the quantities in F and F+1 are shifted left circular; otherwise, they are shifted right end-off with sign extension.

LOGICAL

| Operation Field | Address Field | Interpretation |
|-----------------|---------------|--|
| AND | b,X | AND of BR and memory |
| OR | b,X | OR of BR and memory |
| XOR | b,X | EXCLUSIVE OR of BR and memory |
| LP,F | y | AND of register and immediate operand |
| LOR,F | y | OR of register and immediate operand |
| LXR,F | y | EXCLUSIVE OR of register and immediate operand |

AND Logical AND of Bit Register 0 5 6 10 11 31



F = index register
 b = base bit address; B = b + (X)

Description: Perform the logical AND of the bit register (BR) and the bit from memory specified by B. The result is stored in BR.

AND Operation

| BR | B | RESULT |
|----|---|--------|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

$$BR = (BR) \quad (B)$$

OR LOGICAL OR of Bit Register 0 5 6 10 11 31



F = index register
 b = base bit address; B = b + (X)

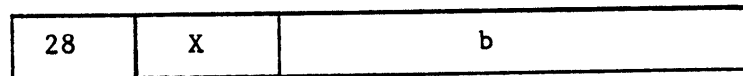
Description: Perform the logical OR of BR and the bit from memory specified by B. The result is stored in BR.

OR Operation

| BR | B | Result |
|-----------|---|--------|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |
| BR = (BR) | | (B) |

XOR Logical EXCLUSIVE OR
of Bit Register

0 5 6 10 11



X = index register
b = base bit address; B = b + (X)

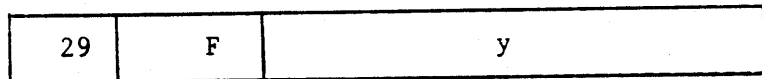
Description: Perform the EXCLUSIVE OR of BR and the bit from memory specified by B. The result is stored in BR.

EXCLUSIVE OR Operation

| BR | B | Result |
|-----------|---|--------|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |
| BR = (BR) | | (B) |

LP Logical Product

0 5 6 10 11 31



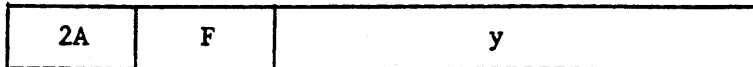
F = operand register
y = immediate operand; 21 bits zero extended

Description: Perform the logical product of the contents of register F and the operand y. The result is stored in F.

$$F = (F) \ y$$

LOR Logical OR

0 5 6 10 11 31



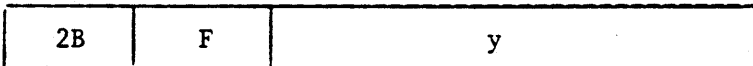
F = operand register
y = immediate operand; 21 bits zero extended

Description: Perform the logical OR of the contents of register F and the operand y. The result is stored in F.

$$F = (F) \text{ OR } y$$

LXR EXCLUSIVE OR

0 5 6 10 11 31



F = operand register
y = immediate operand; 21 bits zero extended

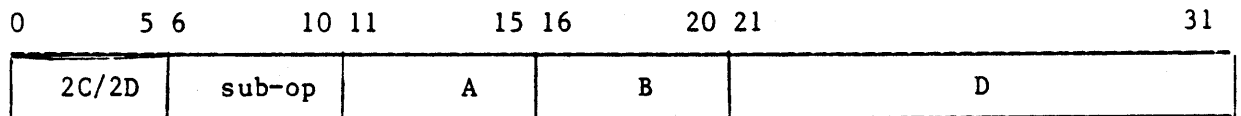
Description: Perform the exclusive OR of the contents of F and the operand y. The result is stored in F.

$$F = (F) \text{ XOR } y$$

TEST

| Operation Field | Address Field | Interpretation |
|-----------------|---------------|--------------------------------------|
| TST | A,B,D | test |
| TSTF | A,B,D | test floating point |
| TSTD | A,B,D | test double precision |
| TSTFD | A,B,D | test double precision floating point |

The test instructions have the following format:



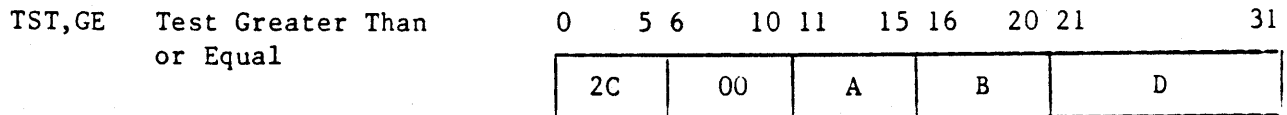
Sub-op = type of test

A = operand register 1

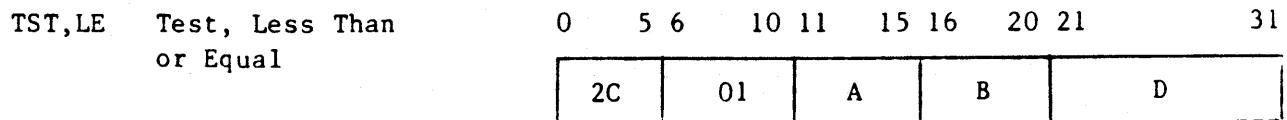
B = operand register 2

D = relative displacement; the range of D is +1023, -1024

(P) = location of test instruction

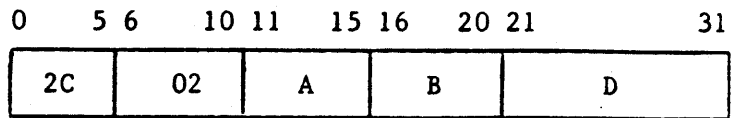


Description: Test the fixed point contents of A greater than or equal to (B). If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction.



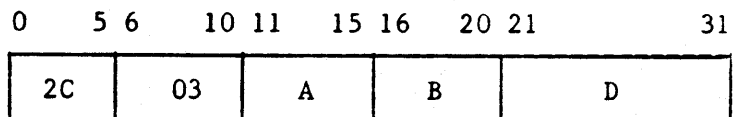
Description: Test A less than or equal to (B). If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction.

TST, EQ Test, Equal



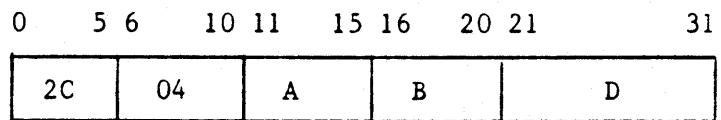
Description: Test (A) equal to (B). If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction.

TST, NE Test, Not Equal



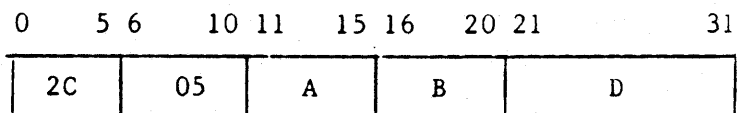
Description: Test (A) not equal to (B). If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction.

TST, GT Test, Greater Than



Description: Test (A) greater than (B). If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction.

TST, LT Test, Less Than



Description: Test (A) less than (B). If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction.

TSTF,GE Test Floating Point Greater Than or Equal

| | | | | | | | | | | |
|---|-----|-------|-------|-------|----|----|----|---|---|---|
| 0 | 5 6 | 10 11 | 15 16 | 20 21 | 31 | | | | | |
| <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%; border: 1px solid black; text-align: center;">2C</td> <td style="width: 20%; border: 1px solid black; text-align: center;">08</td> <td style="width: 20%; border: 1px solid black; text-align: center;">A</td> <td style="width: 20%; border: 1px solid black; text-align: center;">B</td> <td style="width: 20%; border: 1px solid black; text-align: center;">D</td> </tr> </table> | | | | | | 2C | 08 | A | B | D |
| 2C | 08 | A | B | D | | | | | | |

Description: Test floating point operands for (A) greater than or equal to (B). If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction.

TSTF,LE Test Floating Point Less Than or Equal

| | | | | | | | | | | |
|---|-----|-------|-------|-------|----|----|----|---|---|---|
| 0 | 5 6 | 10 11 | 15 16 | 20 21 | 31 | | | | | |
| <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%; border: 1px solid black; text-align: center;">2C</td> <td style="width: 20%; border: 1px solid black; text-align: center;">09</td> <td style="width: 20%; border: 1px solid black; text-align: center;">A</td> <td style="width: 20%; border: 1px solid black; text-align: center;">B</td> <td style="width: 20%; border: 1px solid black; text-align: center;">D</td> </tr> </table> | | | | | | 2C | 09 | A | B | D |
| 2C | 09 | A | B | D | | | | | | |

Description: Test floating point operands for (A) less than or equal to (B). If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction.

TSTF,EQ Test Floating Point Equal

| | | | | | | | | | | |
|---|-----|-------|-------|-------|----|----|----|---|---|---|
| 0 | 5 6 | 10 11 | 15 16 | 20 21 | 31 | | | | | |
| <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%; border: 1px solid black; text-align: center;">2C</td> <td style="width: 20%; border: 1px solid black; text-align: center;">0A</td> <td style="width: 20%; border: 1px solid black; text-align: center;">A</td> <td style="width: 20%; border: 1px solid black; text-align: center;">B</td> <td style="width: 20%; border: 1px solid black; text-align: center;">D</td> </tr> </table> | | | | | | 2C | 0A | A | B | D |
| 2C | 0A | A | B | D | | | | | | |

Description: Test floating point operands for (A) equal to (B). If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction.

TSTF,NE Test Floating Point, Not Equal

| | | | | | | | | | | |
|---|-----|-------|-------|-------|----|----|----|---|---|---|
| 0 | 5 6 | 10 11 | 15 16 | 20 21 | 31 | | | | | |
| <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%; border: 1px solid black; text-align: center;">2C</td> <td style="width: 20%; border: 1px solid black; text-align: center;">0B</td> <td style="width: 20%; border: 1px solid black; text-align: center;">A</td> <td style="width: 20%; border: 1px solid black; text-align: center;">B</td> <td style="width: 20%; border: 1px solid black; text-align: center;">D</td> </tr> </table> | | | | | | 2C | 0B | A | B | D |
| 2C | 0B | A | B | D | | | | | | |

Description: Test floating point operands for (A) not equal to (B). If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction.

TSTF,GT Test Floating Point, Greater Than

| | | | | | |
|----|-----|-------|-------|-------|----|
| 0 | 5 6 | 10 11 | 15 16 | 20 21 | 31 |
| 2C | 0C | A | B | D | |

Description: Test floating point operands for (A) than (B). If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction.

TSTF,LT Test Floating Point, Less Than

| | | | | | |
|----|-----|-------|-------|-------|----|
| 0 | 5 6 | 10 11 | 15 16 | 20 21 | 31 |
| 2C | 0D | A | B | D | |

Description: Test floating point operands for (A) less than (B). If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction.

TSTD,GE Test Double Precision, Greater Than or Equal

| | | | | | |
|----|-----|-------|-------|-------|----|
| 0 | 5 6 | 10 11 | 15 16 | 20 21 | 31 |
| 2D | 00 | A | B | D | |

Description: Test fixed point double precision operand contained in registers A and A+1 for greater than or equal to operand in B and B+1. If true, branch to address specified by (P)+1+D. Otherwise, execute the next instruction.

TSTD,LE Test Double Precision, Less Than or Equal

| | | | | | |
|----|-----|-------|-------|-------|----|
| 0 | 5 6 | 10 11 | 15 16 | 20 21 | 31 |
| 2D | 01 | A | B | D | |

Description: Test fixed point double precision operands for (A,A+1) less than or equal to (B,B+1). If true, branch to address specified by (P)+1+D. Otherwise, execute the next instruction.

TSTD,EQ Test Double Precision, 0 5 6 10 11 15 16 20 21 31
 Equal

| | | | | |
|----|----|---|---|---|
| 2D | 02 | A | B | D |
|----|----|---|---|---|

Description: Test fixed point double precision operands for (A,A+1) equal to (B,B+1). If true, branch to address specified by (P)+1+D. Otherwise, execute the next instruction.

TSTD,NE Test Double Precision, 0 5 6 10 11 15 16 20 21 31
 Not Equal

| | | | | |
|----|----|---|---|---|
| 2D | 03 | A | B | D |
|----|----|---|---|---|

Description: Test fixed point double precision operands for (A,A+1) not equal to (B,B+1). If true, branch to address specified by (P)+1+D. Otherwise, execute the next instruction.

TSTD,GT Test Double Precision, 0 5 6 10 11 15 16 20 21 31
 Greater Than

| | | | | |
|----|----|---|---|---|
| 2D | 04 | A | B | D |
|----|----|---|---|---|

Description: Test fixed point double precision operands for (A,A+1) greater than (B,B+1). If true, branch to address specified by (P)+1+D. Otherwise, execute the next instruction.

TSTD,LT Test Double Precision, 0 5 6 10 11 15 16 20 21 31
 Less Than

| | | | | |
|----|----|---|---|---|
| 2D | 05 | A | B | D |
|----|----|---|---|---|

Description: Test fixed point double precision operands for (A,A+1) less than (B,B+1). If true, branch to address specified by (P)+1+D. Otherwise, execute the next instruction.

TSTFD,GE Test Double Precision Floating Point, Greater Than or Equal

| | | | | | |
|----|-----|-------|-------|-------|----|
| 0 | 5 6 | 10 11 | 15 16 | 20 21 | 31 |
| 2D | 08 | A | B | D | |

Description: Test double precision floating point operand in (A,A+1) greater than or equal to double precision floating point operand in (B,B+1). If true, branch to address specified by (P)+1+D. Otherwise, execute the next instruction.

TSTFD,LE Test Double Precision Floating Point, Less Than or Equal

| | | | | | |
|----|-----|-------|-------|-------|----|
| 0 | 5 6 | 10 11 | 15 16 | 20 21 | 31 |
| 2D | 09 | A | B | D | |

Description: Test double precision floating point operands for (A,A+1) less than or equal to (B,B+1). If true, branch to address specified by (P)+1+D. Otherwise, execute the next instruction.

TSTFD,EQ Test Double Precision Floating Point, Equal

| | | | | | |
|----|-----|-------|-------|-------|----|
| 0 | 5 6 | 10 11 | 15 16 | 20 21 | 31 |
| 2D | 0A | A | B | D | |

Description: Test double precision floating point operands for (A,A+1) equal to (B,B+1). If true, branch to address specified by (P)+1+D. Otherwise, execute the next instruction.

TSTFD,NE Test Double Precision Floating Point, Not Equal

| | | | | | |
|----|-----|-------|-------|-------|----|
| 0 | 5 6 | 10 11 | 15 16 | 20 21 | 31 |
| 2D | 0B | A | B | D | |

Description: Test double precision floating point operands for (A,A+1) not equal to (B,B+1). If true, branch to address specified by (P)+1+D. Otherwise, execute the next instruction.

TSTFD,GT Test Double Precision 0 5 6 10 11 15 16 20 21 31
 Floating Point,
 Greater Than

| | | | | |
|----|----|---|---|---|
| 2D | 0C | A | B | D |
|----|----|---|---|---|

Description: Test double precision floating point operands for (A,A+1) greater than (B,B+1). If true, branch to address specified by (P)+1+D. Otherwise, execute the next instruction.

TSTFD,LT Test Double Precision 0 5 6 10 11 15 16 20 21 31
 Floating Point,
 Less Than

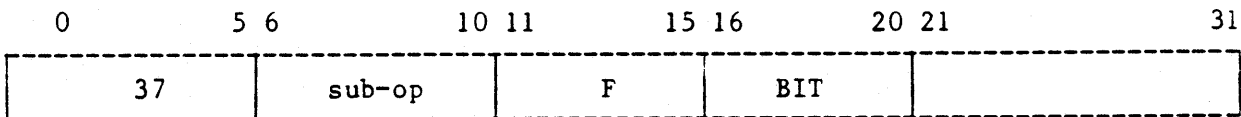
| | | | | |
|----|----|---|---|---|
| 2D | 0D | A | B | D |
|----|----|---|---|---|

Description: Test double precision floating point operands for (A,A+1) less than (B,B+1). If true, branch to address specified by (P)+1+D. Otherwise, execute the next instruction.

REGISTER BIT OPERATIONS

| Operation Field | Address Field | Interpretation |
|-----------------|---------------|----------------|
| TBIT,F | bit,X | toggle bit |
| SBIT,F | bit,X | set bit |
| CBIT,F | bit,X | clear bit |

The Register Bit Operations have the following format:

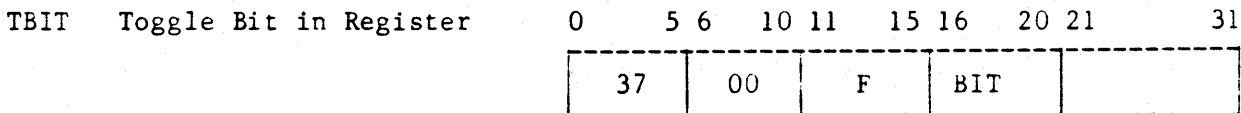


Sub-op = type of operation

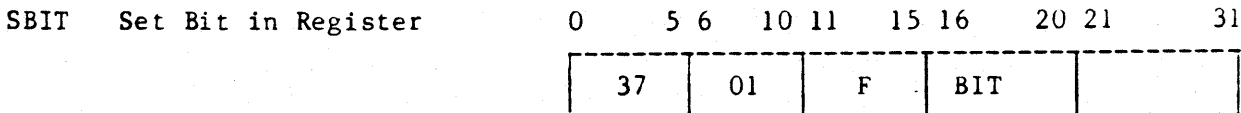
F = register

X = register

BIT = bit (0-31) + (X)



Description: Toggle the BIT in register F.



Description: Set BIT in register F.

CBIT Set Bit in Register

0 5 6 10 11 15 16 20 21 31

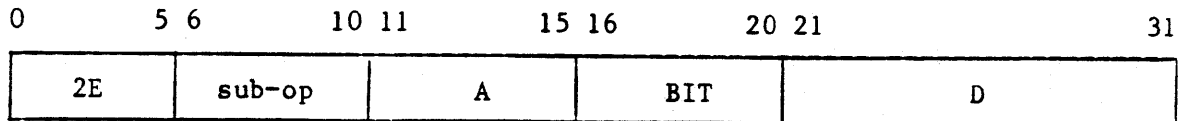
| | | | | |
|----|----|---|-----|--|
| 37 | 02 | F | BIT | |
|----|----|---|-----|--|

Description: Clear BIT in register F.

BIT SKIPS

| Operation Field | Address Field | Interpretation |
|-----------------|---------------|----------------|
| BSK | A,BIT,D | bit skip |

The bit skip instructions have the following format:



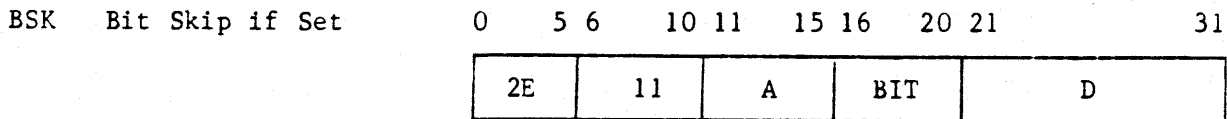
Sub-op = type of test and bit modification

A = test register

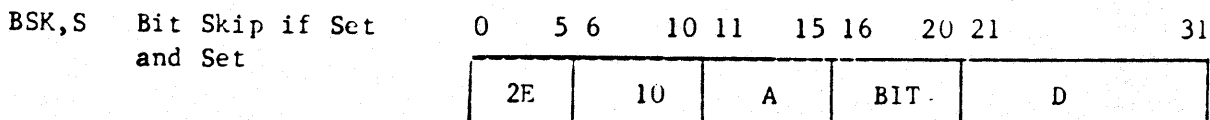
Bit = bit number to test; 0-31

D = relative displacement; the range of D is +1023, -1024

(P) = location of bit skip instruction



Description: Test the referenced BIT in register A equal to one. If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction.



Description: Test the referenced BIT in A equal to one. If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction. Unconditionally set the bit after the test.

| | | | | | | | | | | | | |
|-------|------------------------------|---|-----|-------|-------|-------|----|----|----|---|-----|---|
| BSK,C | Bit Skip if Set and Clear | 0 | 5 6 | 10 11 | 15 16 | 20 21 | 31 | | | | | |
| | | <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr> <td style="width: 16.6%;">2E</td> <td style="width: 16.6%;">12</td> <td style="width: 16.6%;">A</td> <td style="width: 16.6%;">BIT</td> <td style="width: 16.6%;">D</td> </tr> </table> | | | | | | 2E | 12 | A | BIT | D |
| 2E | 12 | A | BIT | D | | | | | | | | |

Description: Test the referenced BIT in A equal to one. If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction. Unconditionally clear the bit after the test.

| | | | | | | | | | | | | |
|-------|-------------------------------|---|-----|-------|-------|-------|----|----|----|---|-----|---|
| BSK,T | Bit Skip if Set and Toggle | 0 | 5 6 | 10 11 | 15 16 | 20 21 | 31 | | | | | |
| | | <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr> <td style="width: 16.6%;">2E</td> <td style="width: 16.6%;">13</td> <td style="width: 16.6%;">A</td> <td style="width: 16.6%;">BIT</td> <td style="width: 16.6%;">D</td> </tr> </table> | | | | | | 2E | 13 | A | BIT | D |
| 2E | 13 | A | BIT | D | | | | | | | | |

Description: Test the referenced BIT in A equal to one. If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction. Unconditionally toggle the bit after the test.

| | | | | | | | | | | | | |
|-------|-------------------|---|-----|-------|-------|-------|----|----|----|---|-----|---|
| BSK,Z | Bit Skip if Clear | 0 | 5 6 | 10 11 | 15 16 | 20 21 | 31 | | | | | |
| | | <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr> <td style="width: 16.6%;">2E</td> <td style="width: 16.6%;">14</td> <td style="width: 16.6%;">A</td> <td style="width: 16.6%;">BIT</td> <td style="width: 16.6%;">D</td> </tr> </table> | | | | | | 2E | 14 | A | BIT | D |
| 2E | 14 | A | BIT | D | | | | | | | | |

Description: Test the referenced BIT in A equal to zero. If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction.

| | | | | | | | | | | | | |
|--------|------------------------------|---|-----|-------|-------|-------|----|----|----|---|-----|---|
| BSK,ZS | Bit Skip if Clear and Set | 0 | 5 6 | 10 11 | 15 16 | 20 21 | 31 | | | | | |
| | | <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr> <td style="width: 16.6%;">2E</td> <td style="width: 16.6%;">15</td> <td style="width: 16.6%;">A</td> <td style="width: 16.6%;">BIT</td> <td style="width: 16.6%;">D</td> </tr> </table> | | | | | | 2E | 15 | A | BIT | D |
| 2E | 15 | A | BIT | D | | | | | | | | |

Description: Test the referenced BIT in A equal to zero. If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction. Unconditionally set the bit after the test.

BSK,ZC Bit Skip if Clear and Clear 0 5 6 10 11 15 16 20 21 31

| | | | | |
|----|----|---|-----|---|
| 2E | 16 | A | BIT | D |
|----|----|---|-----|---|

Description: Test the referenced BIT in A equal to zero. If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction. Unconditionally clear the bit after the test.

BSK,ZT Bit Skip if Clear and Toggle 0 5 6 10 11 15 16 20 21 31

| | | | | |
|----|----|---|-----|---|
| 2E | 17 | A | BIT | D |
|----|----|---|-----|---|

Description: Test the referenced BIT in A equal to zero. If true, branch to the address specified by (P)+1+D. Otherwise, execute the next instruction. Unconditionally toggle the bit after the test.

FILE SKIPS

| Operation Field | Address Field | Interpretation |
|-----------------|---------------|----------------|
| FSK | y,X | file skip |

The file skip instructions have the following format:

| | | | | |
|----|--------|-------|-------|----|
| 0 | 5 6 | 10 11 | 15 16 | 31 |
| 2F | sub-op | X | y | |

Sub-op = type of test

X = operand file

y = immediate operand; 16 bits zero extended

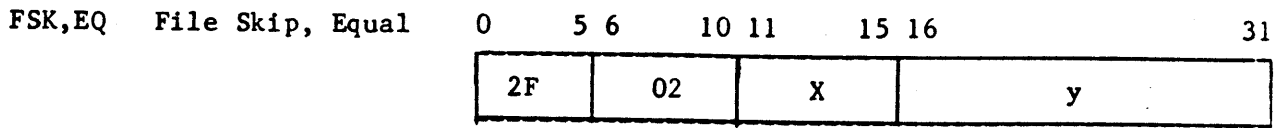
(P) = location of file skip instruction

| | | | | |
|--|----|-----|-------|-------|
| FSK,GE File Skip, Greater Than or Equal | 0 | 5 6 | 10 11 | 15 16 |
| | 2F | 00 | X | y |

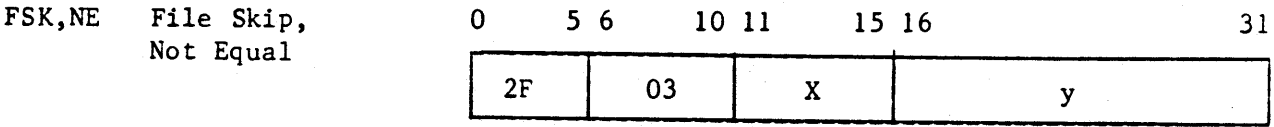
Description: Test the contents of register X greater than or equal to the immediate operand y. If true, skip to instruction specified by (P)+2. Otherwise, execute the next instruction.

| | | | | | |
|---|----|-----|-------|-------|----|
| FSK,LE File Skip, Less Than or Equal | 0 | 5 6 | 10 11 | 15 16 | 31 |
| | 2F | 01 | X | y | |

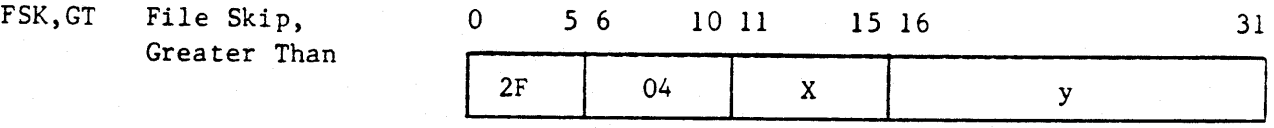
Description: Test the contents of register X less than or equal to the immediate operand y. If true, skip to address specified by (P)+2. Otherwise, execute the next instruction.



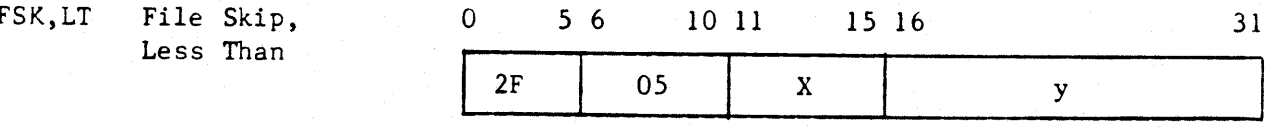
Description: Test the contents of register X equal to the immediate operand y. If true, skip to address specified by (P)+2. Otherwise, execute the next instruction.



Description: Test the contents of register X not equal to the immediate operand y. If true, skip to the address specified by (P)+2. Otherwise, execute the next instruction.



Description: Test the contents of register X greater than the immediate operand y. If true, skip to the address specified by (P)+2. Otherwise, execute the next instruction.



Description: Test the contents of register X less than the immediate operand y. If true, skip to the address specified by (P)+2. Otherwise, execute the next instruction.

JUMPS

| Operation Field | Address Field | Interpretation |
|-----------------|---------------|---------------------------------|
| UJP | m,X | unconditional jump |
| UJI | m,X | unconditional jump indirect |
| RTJ | m,X | return jump |
| JSX | m,X | jump, set index |
| † AIF | | arithmetic IF |
| † AIFD | | arithmetic IF, double precision |
| BJPT | m,X | BR jump, true |
| BJPF | m,X | BR jump, false |
| HLT | m,X | halt |
| XJP | m,X | index jump |
| MON,F | y | monitor call |
| XSK | y,X | index skip |
| † ED01 | | End-Do, increment one |
| † ED02 | | End-Do, increment constant |
| † ED03 | | End-Do, increment variable |
| DXJP | m,X | delayed XJP |
| LCPN,F | | load CPU number |

† Not available as a COMPASS mnemonic instruction.

These instructions use a suboperation field for further definition of the function.

UJP Unconditional Jump 0 5 6 10 11 15 16 31

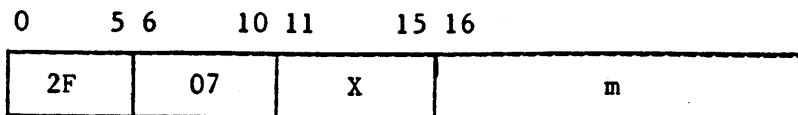
| | | | |
|----|----|---|---|
| 2F | 06 | X | m |
|----|----|---|---|

X = index register
m = base word address; M=m+(X)

Description: Unconditional jump to the address specified by M.

$$P = M$$

UJI Unconditional Jump Indirect

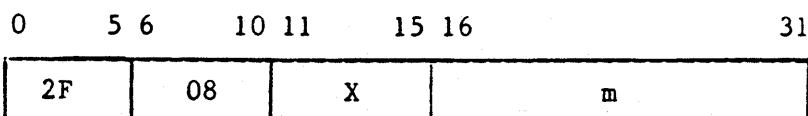


X = index register
 m = base word address; $M=m+(X)$

Description: Unconditionally jump to the address specified by (M).

$$P = (M)$$

RTJ Return Jump

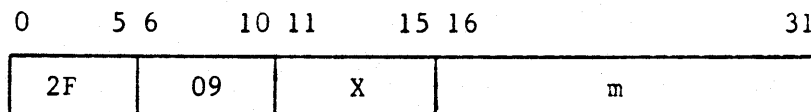


X = index register
 m = base word address; $M=m+(X)$

Description: Replace the address portion of M with (P)+1; jump to the memory address specified by M+1.

$$P = M+1; (P) + 1 \quad M_{16-31}$$

JSX Jump and Set Index

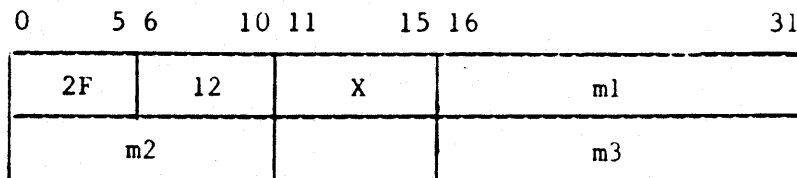


X = index register
 m = base word address

Description: Place (P)+1 in X and jump to the address specified by m.

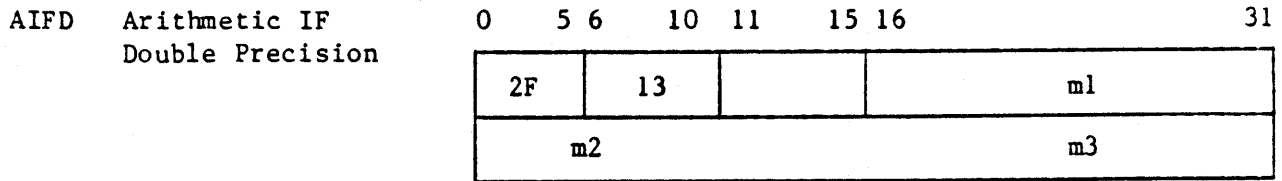
$$X = (P)+1; P = m$$

AIF Arithmetic IF



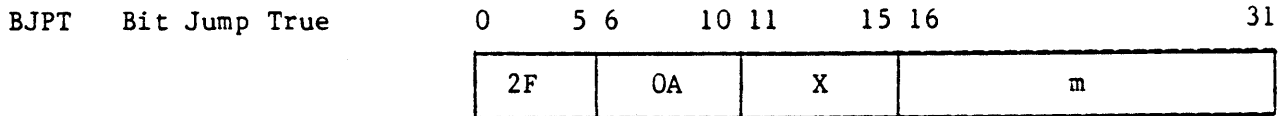
X = index register
 m1 = Base word address; $M1 = m1$
 m2 = Base word address; $M2 = m2 + P + 1$
 m3 = Base word address; $M3 = m3$

Description: Jump to the address specified by m1 if the index register is zero. Jump to the address specified by m2 if the index register is negative. Jump to the address specified by m3 if the index register is positive, $\neq 0$.



X = index register
 m1 = base word address; M1 = m1
 m2 = base word address; M2 = m2 + P = 1
 m3 = base word address; M3 = m3

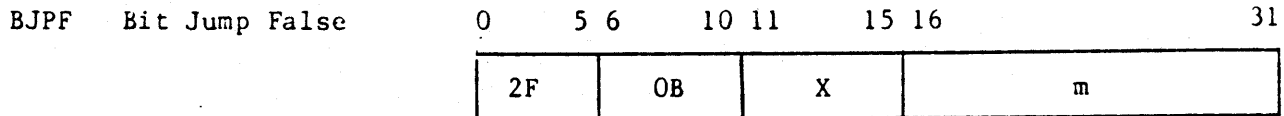
Description: Jump to the address specified by M1 if the index register is zero. Jump to the address specified by M2 if the index register is negative. Jump to the address specified by M3 if the index register is positive, $\neq 0$.



X = index register
 m = base word address; M=m+(X)

Description: Jump to the address specified by M if the contents of the bit register = 1. Otherwise, execute the next instruction.

P = M



X = index register
 m = base word address; M=m+(X)

Description: Jump to the address specified by M if the contents of the bit register = 0. Otherwise, execute the next instruction.

P = M

HLT Stop Program Execution 0 5 6 10 11 15 16 31

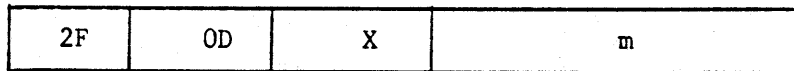


X = index register
 m = base word address; $M=m+(X)$

Description: Unconditionally stop program execution. Upon restart, jump to the address specified by M. This instruction can only be executed in Monitor Mode.

P = M

XJP Index Jump and Decrement 0 5 6 10 11 15 16 31



X = index register
 m = base word address

Description: Test the contents of register X = 0. If $(X) \neq 0$, jump to the address specified by m and replace the (X) by $(X)-1$. Otherwise, execute the next instruction.

P = M

MON Monitor Call 0 5 6 10 11 15 16 31

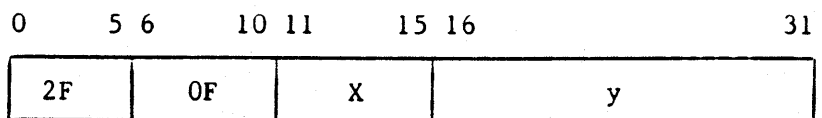


F = start of parameter list
 y = monitor service request

Description: This instruction discontinues execution in the current state and initiates execution in the privileged monitor mode (state 0). The constant y and the four registers F through F+3 are made available to the monitor. Execution of the monitor call instruction proceeds as follows:

1. The execution environment is collected and saved in word 1 of the current exchange package area.
2. The instruction pointer is stored into word 2 of the current exchange package area.
3. The contents of registers F through F+3 are written into words 4 through 7 of the current exchange package area.
4. The y field of the instruction is loaded into register 31 of state zero.
5. The SXPA (successor exchange package address) field of word 3 of the current exchange package is read and defines the address of the new exchange package area.
6. The interrupt system is disabled.
7. Word 3 of the new exchange package is redefined by writing the address of the old exchange package area into the PXPA (predecessor exchange package address) field.
8. The location defined for the MON instruction trap is read and establishes the initial instruction address for execution in the monitor.

XSK Index Skip



X = index register
y = immediate operand

Description: Add one to the contents of register X and compare the result equal to the immediate operand y, not sign extended. If $(X)+1 = y$, execute the next instruction from $(P)+2$. Otherwise, execute the next instruction.

ED01 End-Do
Increment One

| | | | | | | | |
|----|----|---|----|----|----|----|----|
| 0 | 5 | 6 | 10 | 11 | 15 | 16 | 31 |
| 2F | 14 | | X | | A | | |
| m | | | | B | | | |

(A) = address of the Do loop index variable
 (B) = address of the Do loop control variable
 (m) = base word address; $M = m + 1$
 (X) = index register

Description: The ED01 instruction performs a Do loop which is equivalent to the following FORTRAN expression:

Do A = n,B,1 where n is any integer value.

The content of the index variable is incremented by one and stored in the index register. The content of the index register is then tested against the control variable. If $(X) > B$, a jump is performed to the next instruction. If $(X) \leq B$, a jump is performed to the address specified by M, and the value in the index register is stored into the index variable.

ED02 End-Do Increment
Constant

| | | | | | | | |
|----|----|---|----|----|----|----|----|
| 0 | 5 | 6 | 10 | 11 | 15 | 16 | 31 |
| 2F | 15 | | X | | A | | |
| B | | | | C | | | |
| m | | C | | C | | | |

(A) = address of the Do loop index variable
 (B) = Do loop increment constant
 (C) = address of the Do loop control variable
 (m) = base word address; $M = m + 1$
 (X) = index register

Description: The ED02 instruction performs a Do loop which is equivalent to the following FORTRAN expression:

Do A = n,C,(B) where n is any integer value.

The content of the index variable is incremented by the content of the B register and stored in the index register. The content of the index register is then tested against the control variable. If $(X) > (B)$, a jump is performed to the next instruction. If $(X) \leq (B)$, a jump is performed to the address specified by M, and the value contained in the index register is stored into the index variable.

ED03 End-Do Increment Variable

| | |
|----|--|
| | 0 5 6 10 11 15 16 31 |
| 2F | 16 X A |
| | B |
| | m C |

- (A) = address of the Do loop index variable
- (B) = address of the Do loop increment variable
- (C) = address of the Do loop control variable
- (m) = base word address; $M = m + 2$
- (X) = index register

Description: The ED03 instruction performed a Do loop which is equivalent to the following FORTRAN expression:

Do A = n,C,B where n is any integer value

The content of the index variable is incremented by the content of the increment variable and stored into the index register. The content of the index register is then tested against the control variable. If $(X) > (B)$, a jump is performed to the next instruction. if $(X) \leq (B)$, a jump is performed to the address specified by M, and the value contained in the index register is stored into the index variable.

DXJP Delayed XJP

0 5 6 10 11 15 16 31

| | | | |
|----|----|---|---|
| 2F | 1E | X | m |
|----|----|---|---|

x = index register

m = base word address; M = m

Description: Test the contents of register X = 0. If (X) = 0, execute: the instruction following the DXJP instruction and jump to the address specified in M. Replace (X) by (X)-1. If X=0 jump to P+2 and execute that instruction.

LCPN Load CPU
Number

0 5 6 10 11 15 16 31

| | | | |
|----|----|---|--|
| 2F | 1F | F | |
|----|----|---|--|

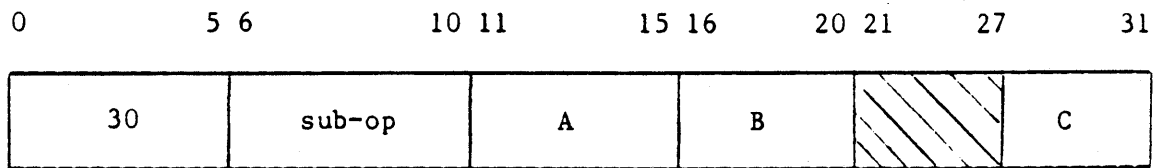
Description: Load register F with the CPU number. The CPU number may be between one and five.

REGISTER OPERATIONS

| Operation Field | Address Field | Interpretation |
|-----------------|---------------|---------------------------------|
| R | A,B,C | single precision fixed point |
| NBR | | complement bit register |
| CBR | | clear bit register |
| RMS | C | read Millisecond Clock |
| RF | A,B,C | single precision floating point |
| RD | A,B,C | double precision fixed point |
| RFD | A,B,C | double precision floating point |
| † RJD | C | read Julian day time |

† Not available as a COMPASS mnemonic instruction.

The register operation instructions have the following format:

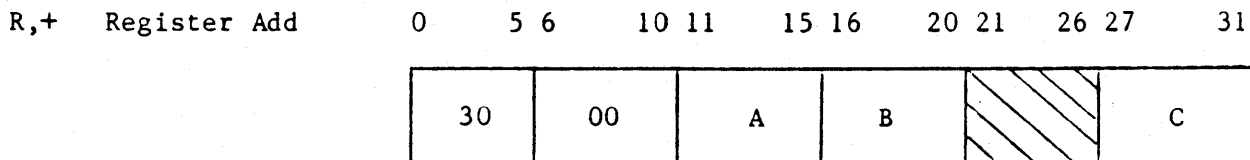


Sub-op = operation

A = source operand register 1

B = source operand register 2

C = destination register



Description: Add the contents of register A to the contents of register B and store the result in register C.

$$C = (A)+(B)$$

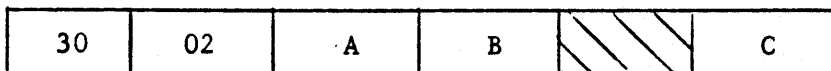
R,- Register Subtract 0 5 6 10 11 15 16 20 21 26 27 31



Description: Subtract the contents of register B from the contents of register A and store the result in register C.

$$C = (A) - (B)$$

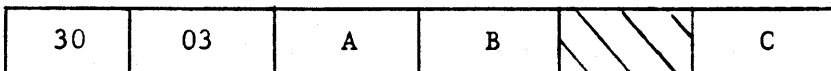
R,* Register Multiply 0 5 6 10 11 15 16 20 21 26 27 31



Description: Multiply the contents of register A by the contents of register B and store the result in registers C and C+1.

$$C, C+1 = (A) * (B)$$

R,/ Register Divide 0 5 6 10 11 15 16 20 21 26 27 31



Description: Divide the contents of registers A and A+1 by the contents of register B and store the quotient in register C and the remainder in register C+1.

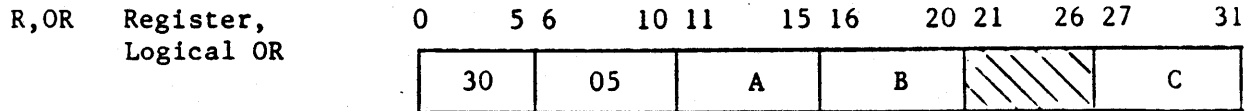
$$C = (A, A+1) / (B); C+1 = \text{remainder}$$

R,AND Register Logical Product 0 5 6 10 11 15 16 20 21 26 27 31



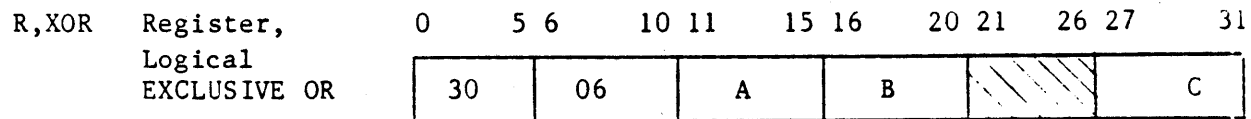
Description: Perform the logical AND of the contents of register A and the contents of register B and store the result in register C.

$$C = (A) \wedge (B)$$



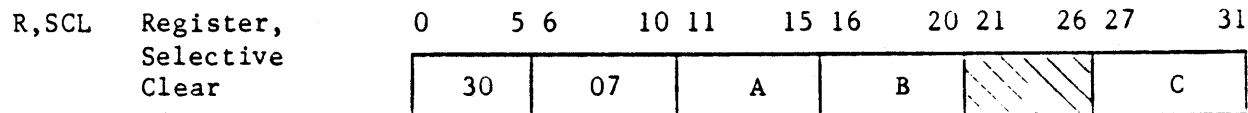
Description: Perform the logical OR of the contents of register A and the contents of register B and store the result in register C.

$$C = (A) \vee (B)$$



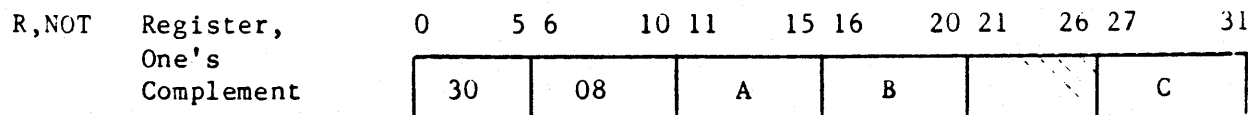
Description: Perform the EXCLUSIVE OR of the contents of register A and the contents of register B and store the result in register C.

$$C = (A) \vee (B)$$



Description: Selectively clear the contents of register A by the mask in register B. For each bit set to 1 in B, the corresponding bit in the result is set to 0. The result is stored in register C.

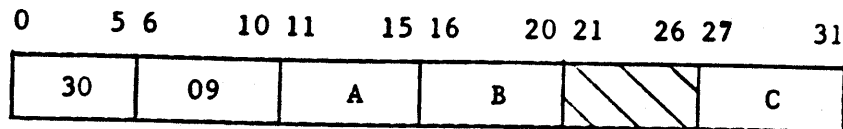
$$C = (A) \vee (\bar{B})$$



Description: Transfer the one's complement of the initial contents of register B to register C and the one's complement of register A to register B.

$$C = (\bar{B}); B = (\bar{A})$$

R,XFR Register, Transfer



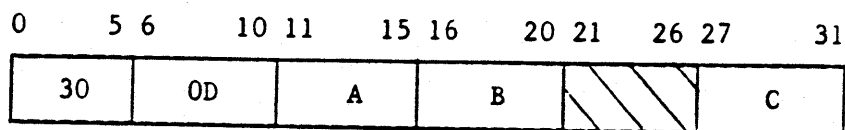
Description: Transfer the initial contents of register B to register C and the contents of register A to register B.

$$C = (B); B = (A)$$

Note: By selecting various values for fields A, B and C, a move, copy or swap operation is performed.

| | | | |
|------|-------|-------|----------------------------------|
| move | R,XFR | A,B,C | result A = (A), B = (A), C = (B) |
| copy | R,XFR | A,B,B | result A = (A), B = (A) |
| swap | R,XFR | A,B,A | result A = (B), B = (A) |

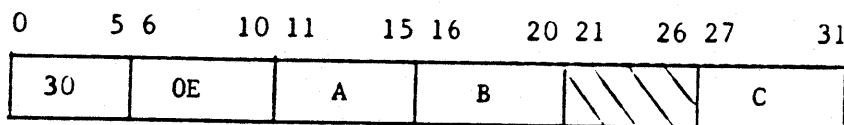
R,S* Register Single Precision Multiply



Description: Multiply the contents of register A by the contents of register B and store the least significant 32 bits of the result in register C.

$$C = (A) * (B)$$

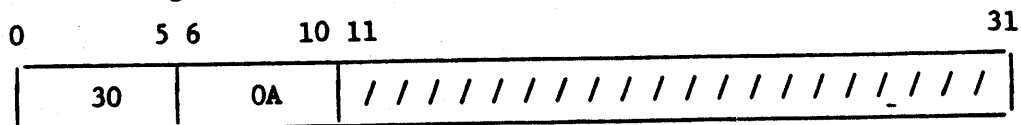
R,S/ Register Single Precision Divide



Description: Divide the contents of register A, sign extended to 64 bits, by the contents of register B and store the quotient in register C.

$$C = (A)/(B)$$

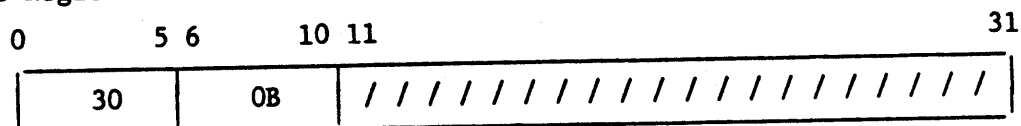
NBR Complement Bit Register



Description: Complement the contents of the bit register.

$$BR = (\overline{BR})$$

SBR Set Bit Register



Description: Set bit register to one.

$$BR = 1$$

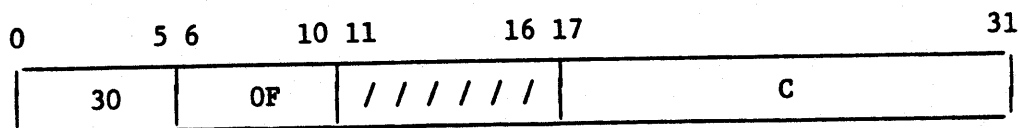
CBR Clear Bit Register



Description: Clear bit register to zero.

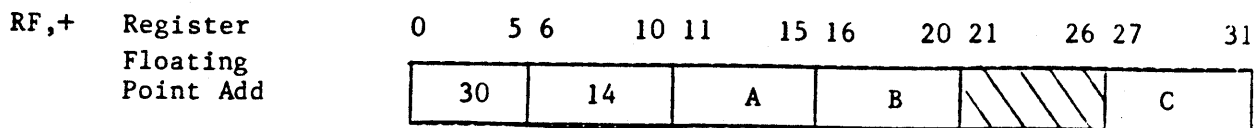
$$BR = 0$$

RMS Read Millisecond Clock Into Register



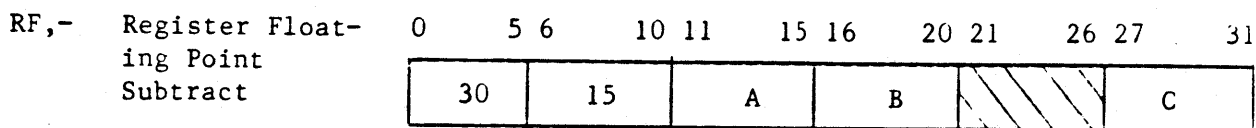
Description: The millisecond time of day clock is transferred to register C.

C = time of day clock



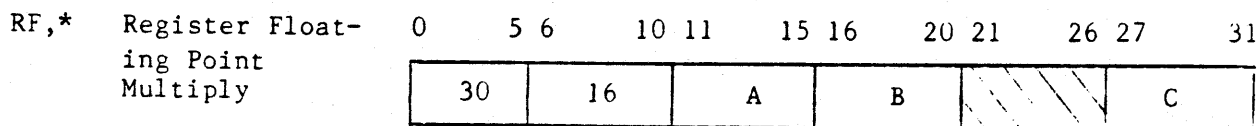
Description: Add the contents of register A to the contents of register B, in floating point mode, and store the result in register C.

$$C = (A) + (B)$$



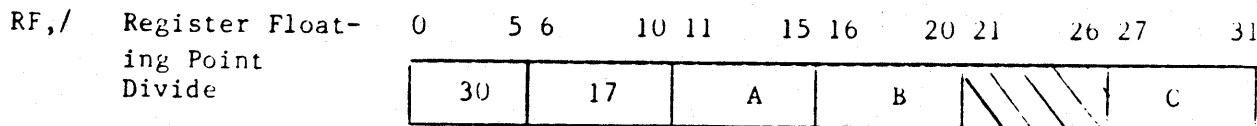
Description: Subtract the contents of register B from the contents of register A and store the result in register C.

$$C = (A) - (B)$$



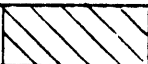
Description: Multiply the contents of register A by the contents of register B and store the result in register C.

$$C = (A) * (B)$$



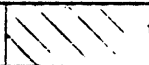
Description: Divide the contents of register A by the contents of register B and store the result in register C.

$$C = (A)/(B)$$

| | | | | | | | | |
|-------|--|----|-----|-------|-------|---|-------|----|
| RFD,+ | Register Double Precision Floating Point Add | 0 | 5 6 | 10 11 | 15 16 | 20 21 | 26 27 | 31 |
| | | 30 | 18 | A | B |  | C | |

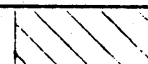
Description: Add the contents of registers A and A+1 to the contents of B and B+1 in floating point mode. The result is stored in registers C and C+1.

$$C, C+1 = (A, A+1) + (B, B+1)$$

| | | | | | | | | |
|-------|---|----|-----|-------|-------|---|-------|----|
| RFD,- | Register Double Precision Floating Point Subtract | 0 | 5 6 | 10 11 | 15 16 | 20 21 | 26 27 | 31 |
| | | 30 | 19 | A | B |  | C | |

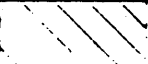
Description: Subtract the contents of registers B and B+1 from the contents of registers A and A+1 and store the result in registers C and C+1.

$$C, C+1 = (A, A+1) - (B, B+1)$$

| | | | | | | | | |
|-------|---|----|-----|-------|-------|---|-------|----|
| RFD,* | Register Double Precision Floating Point Multiply | 0 | 5 6 | 10 11 | 15 16 | 20 21 | 26 27 | 31 |
| | | 30 | 1A | A | B |  | C | |

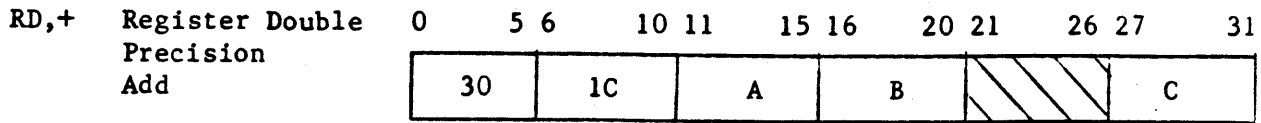
Description: Multiply the contents of registers A and A+1 by the contents of registers B and B+1 and store the result in registers C and C+1.

$$C, C+1 = (A, A+1) * (B, B+1)$$

| | | | | | | | | |
|-------|---|----|-----|-------|-------|---|-------|----|
| RFD,/ | Register Double Precision Floating Point Divide | 0 | 5 6 | 10 11 | 15 16 | 20 21 | 26 27 | 31 |
| | | 30 | 1B | A | B |  | C | |

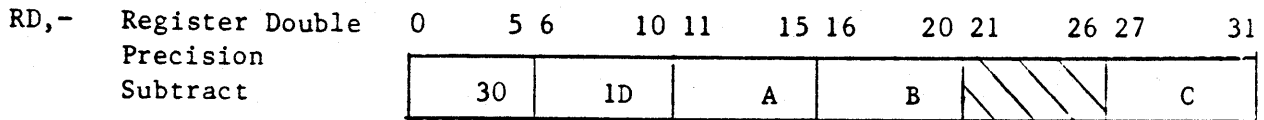
Description: Divide the contents of registers A and A+1 by the contents of registers B and B+1 and store the result in registers C and C+1.

$$C, C+1 = (A, A+1) / (B, B+1)$$



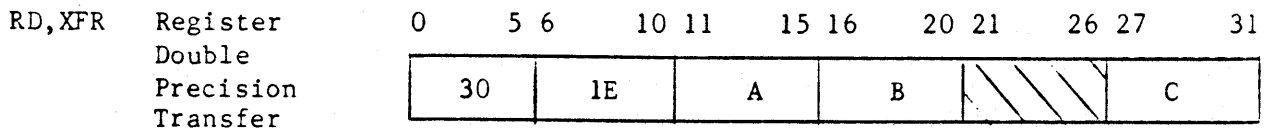
Description: Add the contents of registers A and A+1 to the contents of registers B and B+1 in fixed point mode. The result is stored in registers C and C+1.

$$C, C+1 = (A, A+1) + (B, B+1)$$



Description: Subtract the contents of registers B and B+1 from the contents of registers A and A+1 and store the result in registers C and C+1.

$$C, C+1 = (A, A+1) - (B, B+1)$$



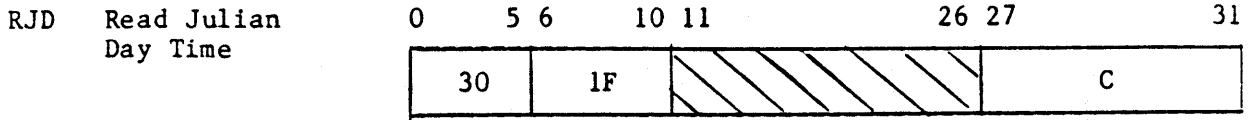
Description: Transfer the initial contents of registers B and B+1 to registers C and C+1. Transfer the contents of registers A and A+1 to registers B and B+1.

$$C, C+1 = (B, B+1); B, B+1 = (A, A+1)$$

Note: By selecting various values for fields A, B and C, a move, copy or swap operation is performed.

move RD,XFR A,B,C result A,A+1 = (A,A-1)
B,B+1 = (A,A-1)

| | | | | |
|------|--------|-------|--------|-----------------|
| copy | RD,XFR | A,B,B | result | C,C+1 = (B,B-1) |
| | | | | A,A+1 = (A,A-1) |
| | | | | B,B+1 = (B,B-1) |
| swap | RD,XFR | A,B,A | result | A,A+1 = (B,B-1) |
| | | | | B,B+1 = (A,A-1) |

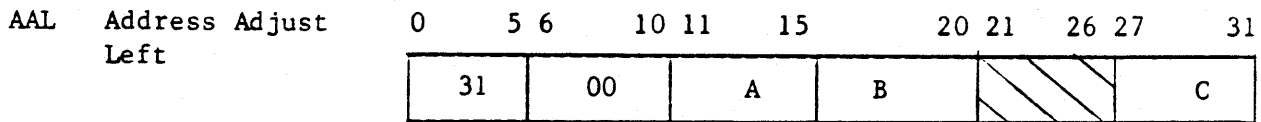


Description: The millisecond Julian time of day is transferred to register C.
C = Julian time of day clock

FUNCTION

| Operation Field | Address Field | Interpretation |
|-----------------|---------------|---|
| AAL F | A,B,C A,C | address adjust left perform named function |

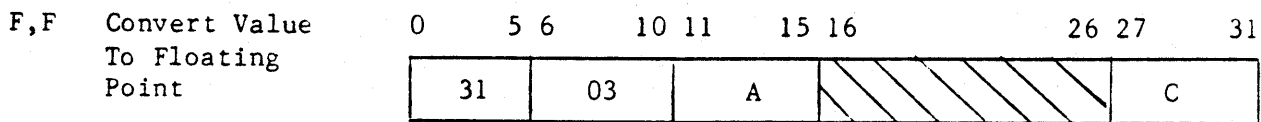
These instructions use a suboperation field for further definition of the function.



Description: Convert the word address contained in register A to a half word address (store in register B) and a character address (store in register C).

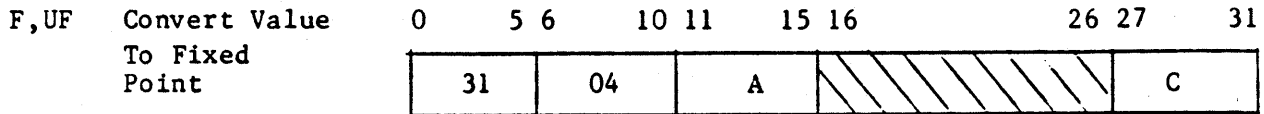
$$(B) = (A)*2$$

$$(C) = (A)*4$$



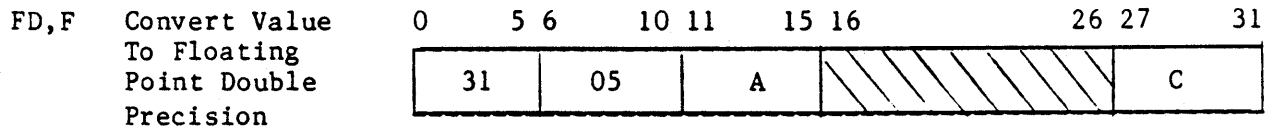
Description: Convert the integer value in register A and store the floating point equivalent in register C. Since the floating point coefficient is 24 bits in length, large 32-bit integer values may be truncated.

$$C = (A); \text{ floated}$$



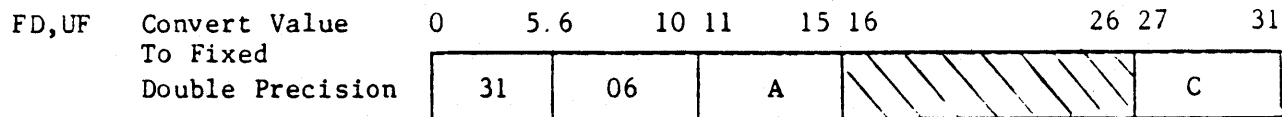
Description: Convert the floating point value in register A and store the integer equivalent in register C. If the floating point value is beyond the range of numbers expressable in 32 bits, either too large or too small, the value zero is stored in C and the function fault is set.

$$C = (A); \text{ integer part}$$



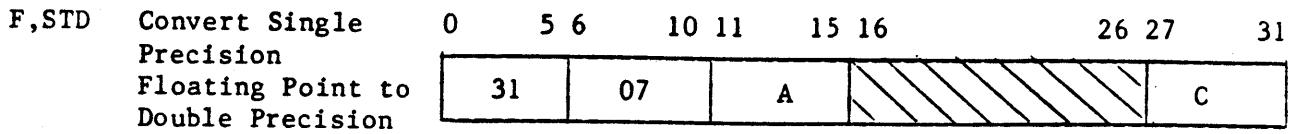
Description: Convert the integer value in registers A and A+1 and store the floating point equivalent in registers C and C+1. Since the double precision floating point coefficient is 56 bits in length, large 64-bit integer values may be truncated.

$$C, C+1 = (A, A+1); \text{ floated}$$



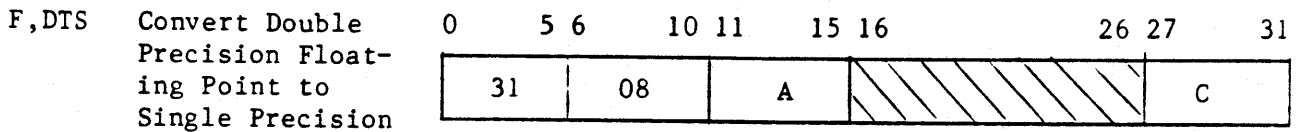
Description: Convert the floating point value in registers A and A+1 and store the integer equivalent in registers C and C+1. If the floating point value is beyond the range of numbers expressable in 64 bits, either too large or too small, the value zero is stored in C and the function fault is set.

$$C, C+1 = (A, A+1); \text{ integer part}$$



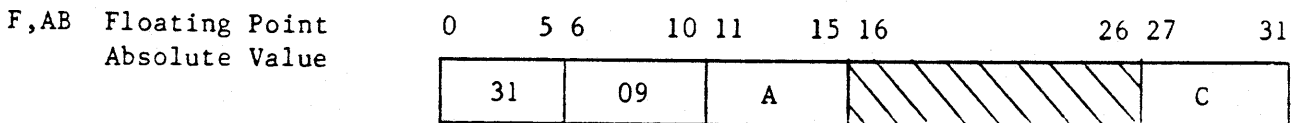
Description: Convert the single precision floating point value in register A to double precision and store the result in registers C and C+1.

$$C, C+1 = (A)$$

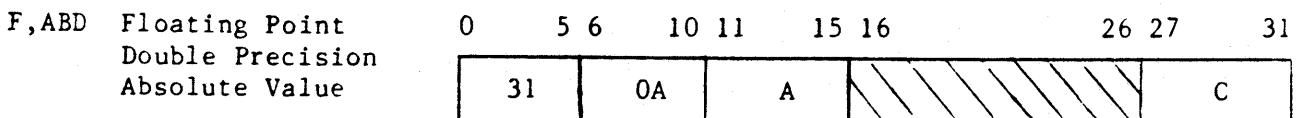


Description: Convert the double precision floating point value in registers A and A+1 to single precision and store the result in register C.

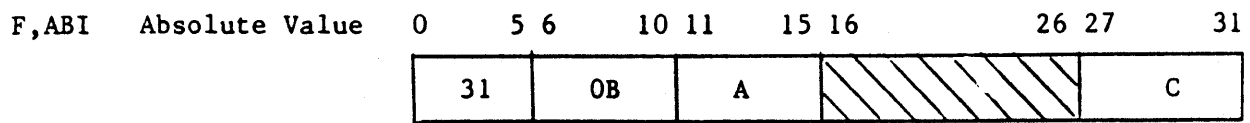
$$C = (A, A+1)$$



Description: The floating point number in C is set to the absolute value of the floating point number in A.



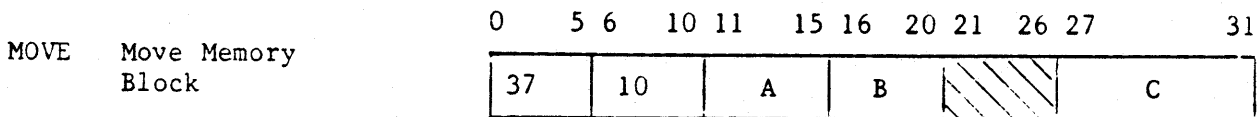
Description: The double precision floating point number in C, C+1 is set to the absolute value of the double precision floating point number in A, A+1.



Description: The absolute value of the number in A is transferred to (C).

BLOCK TRANSFERS

| Operation Field | Address Field | Interpretation |
|-----------------|---------------|------------------------|
| MOVE | A,B,C | move memory block |
| MOVC | A,B,C | move character block |
| FILL | A,B,C | fill memory block |
| MOVT | A,B,C | move and transliterate |
| MOVA | A,B,C | move and align data |
| MOVU | A,B,C | move and unalign data |
| MOVP | A,B,C | pack byte |
| MOVN | A,B,C | unpack byte |



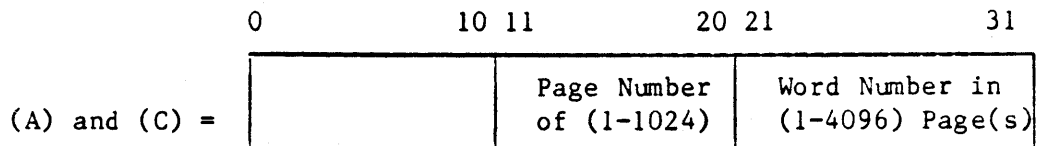
A = FROM first word address
 B = number of words to transfer
 C = TO first word address

Description: The number of words specified in register B are transferred from successive memory locations starting with the location specified in register A, and continuing to the location specified in register C. If the high order bit in register C is set, the transfer is "last to first" with registers A and C decrementing by one. If the high order bit in register C is not set, the transfer is "first to last" with registers A and C incrementing by one. Register B is decrementing by one following each transfer. Interrupts are checked after each word transfer.

If an interrupt occurs during the transfer, the transfer and the instruction are terminated, P is not advanced and the interrupt is processed normally. On return from the interrupt, the instruction is reinitiated and the transfer is resumed at the point of interruption.

If this instruction is executed from Program Mode, the memory addresses are normal 16-bit addresses specifying memory assigned to that state. Memory protect violations apply and are treated in the normal way.

If this instruction is executed from Monitor Mode, the memory addresses are 21-bit addresses as described below. Normal protect violations apply and are treated in the normal way.



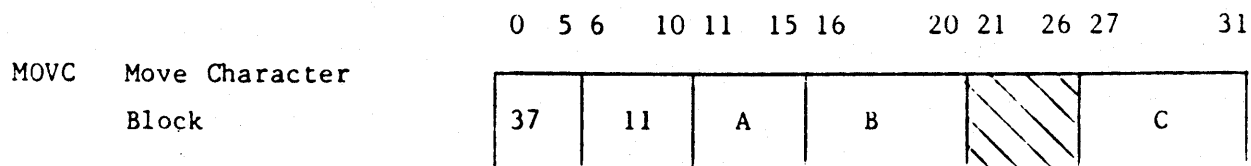
Upon completion of the transfer, registers A, B, and C contain the following:

(A) = address of the last FROM word transferred + 1

(B) = 0

(C) = address of the last TO word set + 1

NOTE: If this instruction is used to propagate data, as in zero-filling a vector, the address in C must be at least 2 words beyond the address in A. This is due to the look-ahead feature of the hardware.



A = FROM memory first byte address

B = number of bytes to transfer

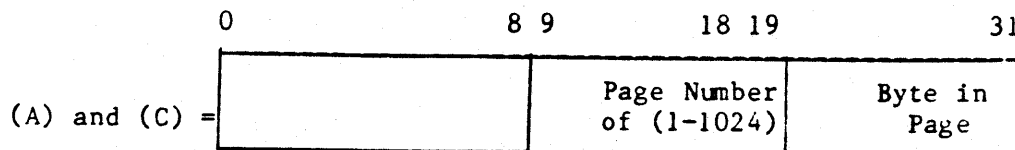
C = TO memory first byte address

Description: The number of characters specified in register B are transferred from successive memory locations starting with the location specified in register A, and continuing to the location specified in register C. If the high order bit in register C is set, the transfer is "last to first" with registers A and C decrementing by one. If the high order bit in register C is not set, the transfer is "first to last" with registers A and C incrementing by one. Register B is decremented by one following each transfer. Interrupts are checked after each byte transfer.

If an interrupt occurs during the transfer, the transfer and the instruction are terminated, P is not advanced and the interrupt is processed normally. On return from the interrupt, the instruction is reinitiated and the transfer is resumed at the point of interruption.

If this instruction is executed from Program Mode, the memory addresses are normal 16-bit addresses specifying memory assigned to that state. Memory protect violations apply and are treated in the normal way.

If this instruction is executed from Monitor Mode, the memory addresses are 23-bit addresses as described below. Normal protect violations apply and are treated in the normal way.

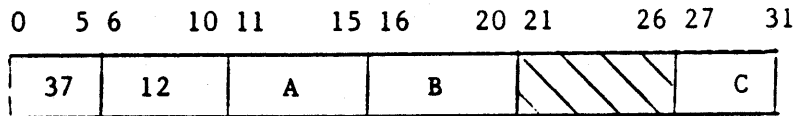


Upon completion of the transfer, registers A, B, and C contain the following:

- (A) = address of the last FROM byte transferred + 1
- (B) = 0
- (C) = address of the last TO byte set + 1

NOTE: If this instruction is used to propagate data, as in zero-filling a vector, the byte address in (C) must be at least 2 bytes beyond the address in (A). This is due to the look-ahead feature of the hardware.

FILL Fill Memory Block



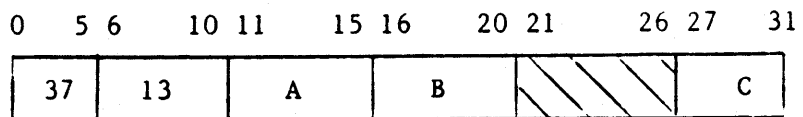
- (A) = fill value
- (B) = number of words to move
- (C) = TO first word address or byte

Description: The Fill instruction allows MP-60 programs to fill successive buffer locations with the value specified in register A. If the high order bit of register C is set, the value contained in register C is the TO first word address. In that case the unit transfer size is a word. If the high order bit of register C is not set, the value contained in register C is the TO first byte address, and the unit transfer size is a byte.

On completion of the transfer, registers A, B, and C contain the following:

- A = unchanged
- B = 0
- C = last TO word address + 1

MOVT Move and Transliterate



- (A) = FROM first byte address
- (B) = number of bytes to move
- (C) = TO starting byte address
- (B+1) = starting byte address of the transliteration table

The transliteration table has the following format:

| | |
|-------------|--------|
| ordinal - 1 | char 1 |
| ordinal - 2 | char 2 |
| . | . |
| . | . |
| ordinal - i | char i |
| . | . |
| . | . |
| ordinal - n | char n |

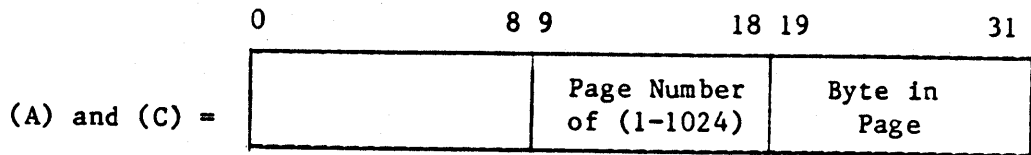
where ordinal 1 - n are input characters and char 1 - n are the replacement characters to be written by MOVT to the next TO address.

Description: The MOVT instruction performs a character by character transfer of data beginning with the byte address specified in register A and continuing to the area beginning with the byte address specified in register C. As the transfer takes place data is translated by indexing into the transliteration table whose starting byte is specified in register B+1. Registers A and C are incremented and register B is decremented following each transfer. Interrupts are checked following each byte transfer.

If an interrupt occurs during the transfer, the transfer and the instruction are terminated, P is not advanced, and the interrupt is processed normally. On return from the interrupt, the instruction is reinitiated and the transfer resumed at the point of interruption.

If this instruction is executed from Program Mode, the memory addresses are normal 18-bit addresses specifying memory assigned to that state. Memory protect violations apply and are treated in the normal way.

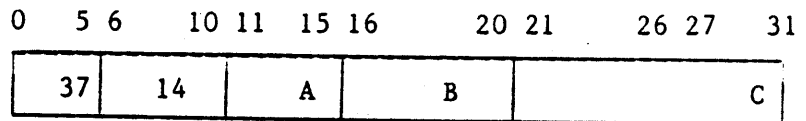
If this instruction is executed from Monitor Mode, the memory addresses are 23-bit addresses as described below. Normal protect violations apply and are treated in the normal way.



Upon completion of the transfer, registers A, B, and C contain the following:

- (A) = last FROM byte address +1
- (B) = 0
- (C) = last TO byte address set +1

MOVA Move and Allign
Data



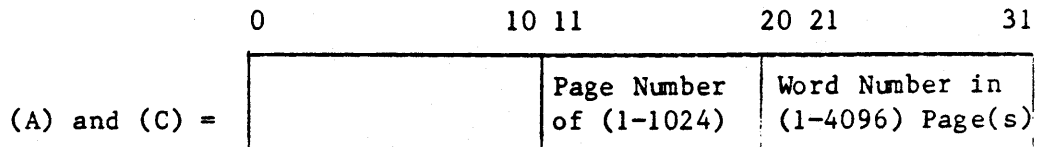
- (A) = FROM first word address
- (B) = bit offset in first word
- (C) = TO starting word address
- (B+1) = number of words to transfer, where a partial word is counted as a word
- (B+2) = number of bits in the last transfer

Description: The MOVA instruction allows MP-60 programs to specify a data transfer with a non-word aligned FROM address. Starting with the bit offset specified in register B of the word address specified in register A, data is transferred in 32-bit chunks to the area whose starting address is specified in register C. When all but one of the words to be transferred have been moved, the value in register B+2 is used to make the last transfers. As each 32-bit chunk is transferred, register C is incremented and register B+1 is decremented. Interrupts are checked following each word transfer.

If an interrupt occurs during the transfer, the transfer and the instruction are terminated, P is not advanced and the interrupt is processed normally. On return from the interrupt, the instruction is reinitiated and the transfer resumed at the point of interruption.

If this instruction is executed from Program Mode, the memory addresses are normal 16-bit addresses specifying memory assigned to that state. Memory protect violations apply and are treated in the normal way.

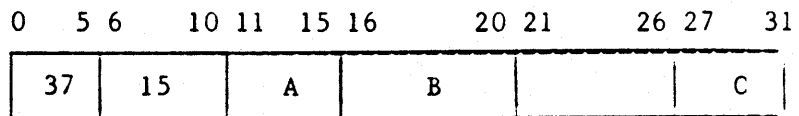
If this instruction is executed from Monitor Mode, the memory addresses are 21-bit addresses as described below. Normal protect violations apply and are treated in the normal way.



Upon completion of the transfer, registers A, B, C, B+1, and B+2 contain the following:

- (A) = last FROM word address + 1
- (B) = unchanged
- (C) = last TO word address set + 1
- (B+1) = 0
- (B+2) = unchanged

MOVU Move and
 Unalign Data



- (A) = FROM first word address
- (B) = starting bit number in destination word
- (C) = TO starting word address
- (B+1) = if (B) = 1, number of words to transfer
if 1 < (B) < 32, number of words to transfer + 1
- (B+2) = number of bits in last word

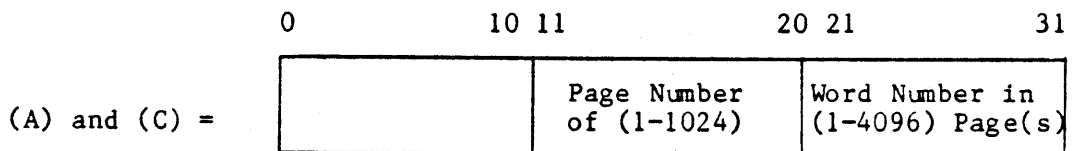
Description: The MOVU instruction allows MP-60 programs to specify a data transfer to a non-word aligned address. Data is transferred from the area beginning with the address specified in register A to the area beginning with the word specified in register C and bit position specified in register B. Data is transferred in 32-bit chunks. When all but one of the words to be transferred have been moved, the value in register B+2 is used to make the last

transfers. As each 32-bit data group is moved, register A is incremented and register B+1 is decremented. Interrupts are checked following each word transfer.

If an interrupt occurs during the transfer, the transfer and the instruction are terminated, P is not advanced and the interrupt is processed normally. On return from the interrupt, the instruction is reinitiated and the transfer resumed at the point of interruption.

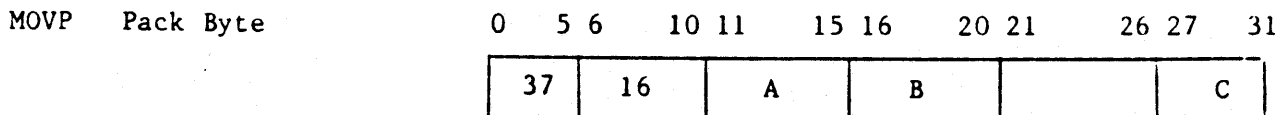
If this instruction is executed from Program Mode, the memory addresses are normal 16-bit addresses specifying memory assigned to that state. Memory protect violations apply and are treated in the normal way.

If this instruction is executed from Monitor Mode, the memory addresses are 21-bit addresses as described below. Normal protect violations apply and are treated in the normal way.



Upon completion of the transfer, registers A, B, C, B+1 and B+2 contain the following:

- (A) = last FROM word address + 1
- (B) = unchanged
- (C) = last TO word address set + 1
- (B+1) = 0
- (B+2) = unchanged



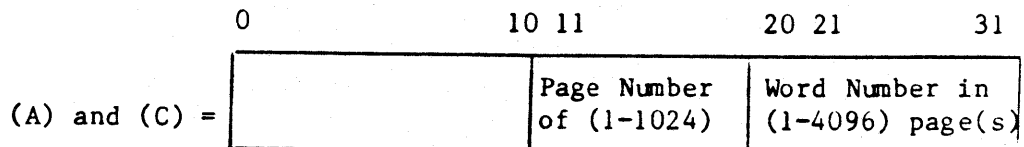
- (A) = FROM byte address
- (B) = number of bytes to pack (multiple of 4)
- (C) = TO byte address

Description: The MOVP instruction allows an MP-60 program to reformat 8-bit bytes which contain only 6 bits of data into 6-bit bytes. The data to be placed into the resulting 6-bit bytes must be right justified in the FROM bytes as the highest order two bits of the 8-bit FROM bytes are ignored. Data is transferred by packing 32-bit data groups into 24 bits. It is therefore necessary that the data to be packed end on a 32-bit multiple. If the data does not terminate in a multiple of 32 bits, zeros must be added to fill out the last 32 bits of the FROM data.

Interrupts are checked following each 32-bit transfer. If an interrupt occurs during the transfer, the instruction is terminated, P is not advanced, and the interrupt is processed normally. On return from the interrupt, the instruction is reinitiated and the transfer resumed at the point of interruption.

If this instruction is executed from Program Mode, the memory addresses are normal 16-bit addresses specifying memory assigned to that state. Memory protect violations apply and are treated in the normal way.

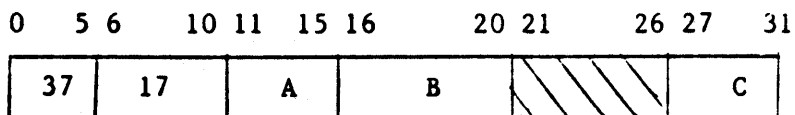
If this instruction is executed from Monitor Mode, the memory addresses are 21-bit addresses as described below. Normal protect violations apply and are treated in the normal way.



Upon completion of the transfer, registers A, B, and C contain the following:

- (A) = last FROM byte transferred + 1
- (B) = 0
- (C) = last TO byte packed + 1

MOVN Unpack Byte



- (A) = FROM byte address
- (B) = number of bytes to unpack (in multiples of 3)
- (C) = TO byte address

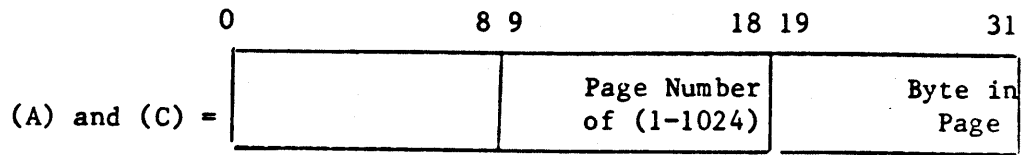
Description: The MOVN instruction allows the user to reformat 6-bit data bytes into 8-bit data bytes. The highest order 2 bits of the resulting 8-bit bytes are guaranteed to be unset.

Data is transferred by unpacking 24-bit data groups into 32 bits and moving the result to the next TO address. It is therefore necessary that the data to be unpacked end on a 24-bit multiple. If the data does not terminate on a multiple of 24 bits, zeros must be added to fill out the last 24 bits of the FROM data.

Interrupts are checked following each 24-bit transfer. If an interrupt occurs during the transfer, the transfer and the instruction are terminated, P is not advanced, and the interrupt is processed normally. On return from the interrupt, the instruction is reinitiated and the transfer resumed at the point of interruption.

If this instruction is executed from Program Mode, the memory addresses are normal 18-bit addresses specifying memory assigned to that state. Memory protect violations apply and are treated in the normal way.

If this instruction is executed from Monitor Mode, the memory addresses are 23-bit addresses as described below. Normal protect violations apply and are treated in the normal way.



Upon completion of the transfer, registers A, B, and C contain the following:

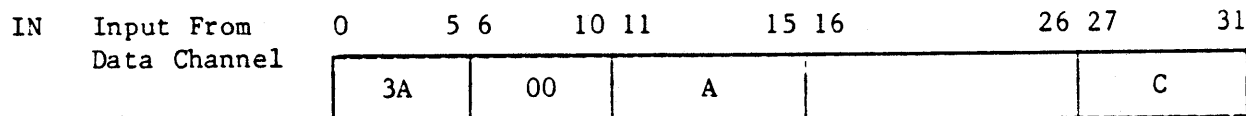
- (A) = last FROM byte transferred + 1
- (B) = 0
- (C) = last unpacked TO byte + 1

SPECIAL FUNCTIONS

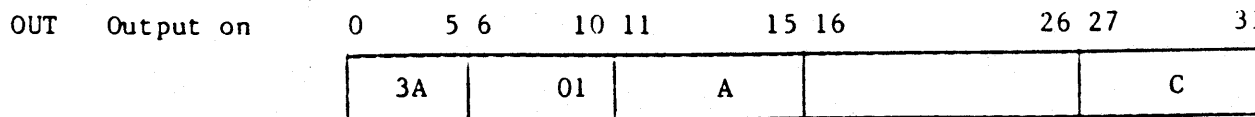
| Operation Field | Address Field | Interpretation |
|-----------------|---------------|----------------------------------|
| IN | A,C | input |
| OUT | A,C | output |
| NIO | A,C | set/sample input/output |
| SPS | A,C | sample status |
| SMIO | A,C | set Micro I/O Table |
| RMIO | A,C | read Micro I/O Table |
| WPF | A,C | write page file |
| RPF | A,C | read page file |
| SPF | A,C | set page files |
| WSR | A,C | write State Register |
| RSR | A,C | read State Register |
| SSRM,F | | selectively set Real Time Mask |
| SCRM,F | | selectively clear Real Time Mask |
| RLM,F | | read Interrupt Mask |
| RRM,F | | read Real Time Mask |
| † SJD,F | | set Julian day |

† Not available as a COMPASS mnemonic instruction.

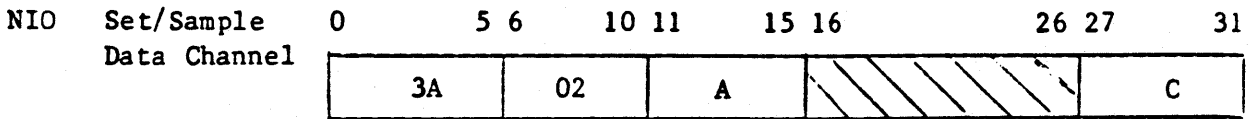
This group of instructions are only legal when executed in Monitor Mode.



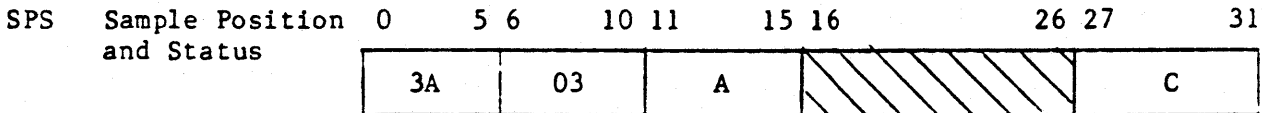
Description: The IN instruction uses register A as the address register and performs an input operation from the I/O TTY card with the data destined to register C. If an internal reject occurs, then control internal passes to P+1, and if an external reject occurs, then control passes to P+2, and on normal return control pass to P+3.



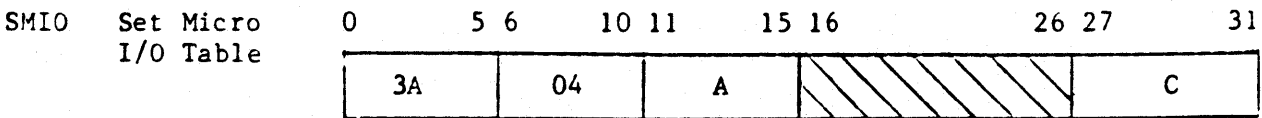
Description: The OUT instruction uses register A as the address register and register C as the data register and performs an output operation to the I/O TTY card. If an internal reject/time out occurs, then control passes to P+1. If an external reject occurs, then control passes to P+2 and on a normal return control passes to P+3.



Description: This instruction performs a set/sample of the data channel. Register A is used for addressing and control, while register C is the data register. Register C contains the data to be output in a set operation, while on a sample operation, the data is destined to register C.

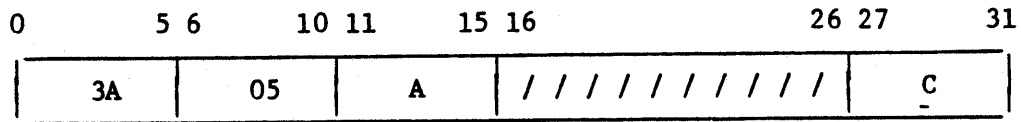


Description: This instruction is used to sample the position and status information after an ADT end of operation interrupt.



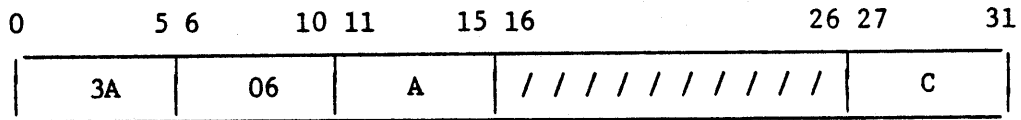
Description: This instruction is used to set up the micro I/O ADT registers. Register A contains the ADT register number, while registers C, C+1 contain the information set into the ADT registers.

RMIO Read Micro I/O Table



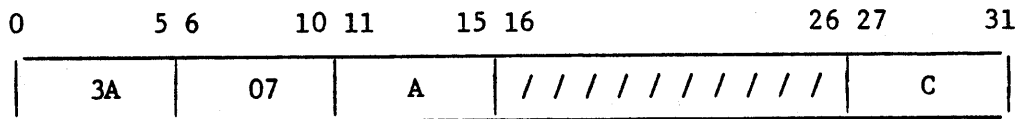
Description: This instruction is used to read the micro I/O ADT registers. Register A contains the ADT register number, while registers C, C+1 receive the information contained in the ADT registers.

WPF Write Page File



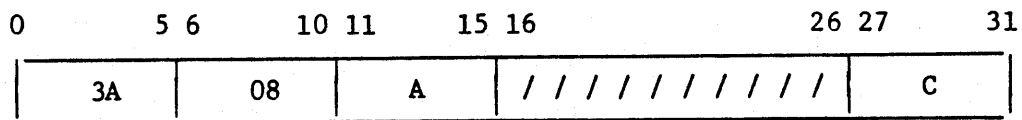
Description: The WPF instruction is used to write page index file entries. Register A contains the page index file address and register C contains the data to be written into the specified page index file.

RPF Read Page File

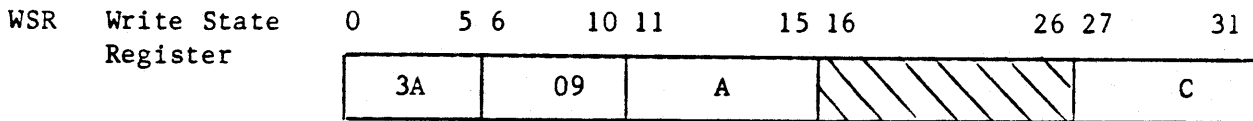
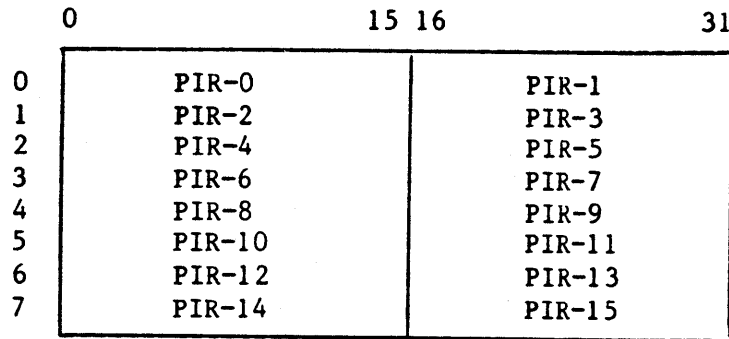


Description: The RPF instruction is used to read page index file entries. Register A contains the page index file address and register C receives the data in the specified page index file.

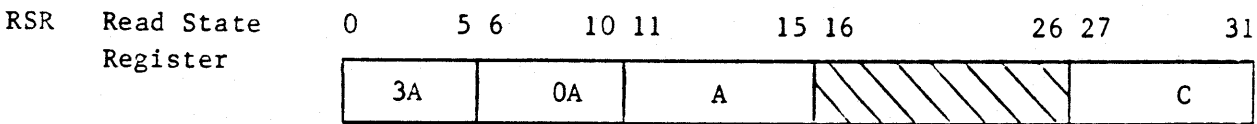
SPF Set Page File



Description: The SPF instruction is used to set 16 page index file entries. Register A contains the page index file address and register C contains the address of 8 words which are to be set into the specified page index file. The formats of the 8 words are defined as follows:



Description: The WSR instruction is used to write the DMA state registers. Register A contains the DMA channel number and register C contains the data. Bit 28 in register A must be set.

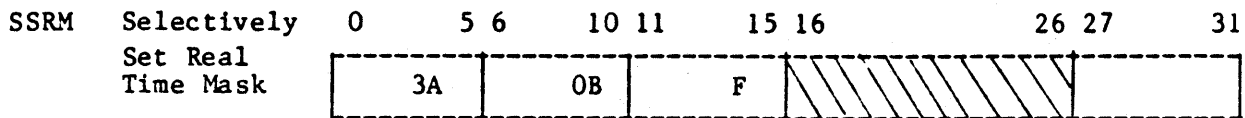


Description: The RSR instruction is used to read the state registers. Register A contains the channel number and register C will contain the data read from the specified state register. Register A is as follows:

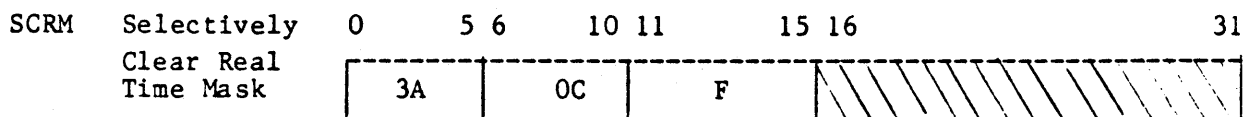
Bit 27 set, then the CPU state register is returned.

Bit 28 set, then bits 30-31 specify the DMA channel number returned.

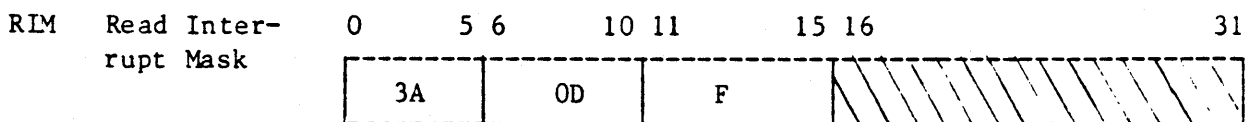
Bit 29 set, then bits 30-31 specify the snapshot register returned.



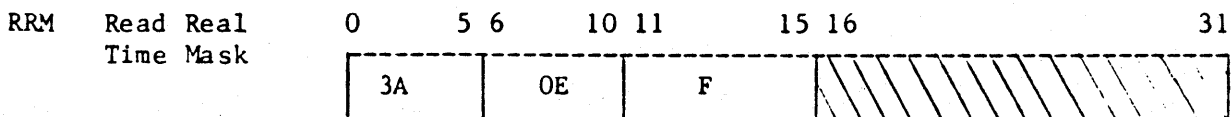
Description: For bits set in register F, set the corresponding bits in the real-time interrupt mask.



Description: For bits set in register F, clear the corresponding bits in the real-time interrupt mask.

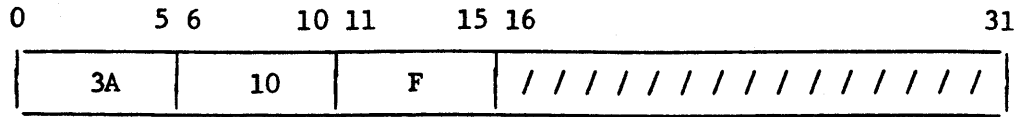


Description: Read the contents of the interrupt mask into register F.



Description: Read the contents of the real-time interrupt mask into register F.

SJD Set Julian Day

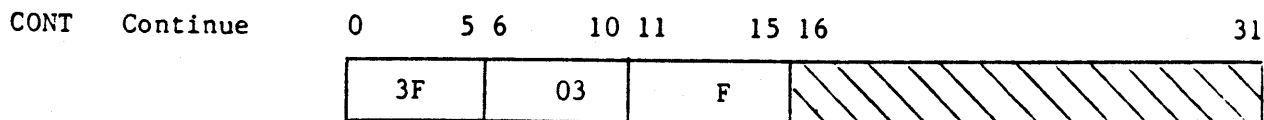


Description: Set Julian day time to the value contained in register F.

EXTERNAL FUNCTIONS

| Operation Field | Address Field | Interpretation |
|-----------------|---------------|----------------------------------|
| CONT, F | | continue |
| LXPA, F | | load exchange package address |
| EINT | | enable interrupt system |
| SPG | A, C | set Page Register |
| RPG | A, C | read Page Register |
| SSIM, F | | selectively set Interrupt Mask |
| SCIM, F | | selectively clear Interrupt Mask |
| SRTC, F | | set Real Time Clock |
| SIT, F | | set Interval Timer |
| TRC, F | | transfer Real Time Clock |
| DINT | | disable Interrupt System |
| SST, F | | set State Relocation Register |
| SOPR | A, C | save Operand Registers |
| LOPR | A, C | load Operand Registers |
| DST, F | | send Interrupt to CPU |
| LPIR | A, C | load Page Index Registers |
| LMM | A, C | load Micro Memory |
| PAUS, F | | pause |
| SCPN, F | | set central processor number |
| OST | A, B, C | operating system thread |
| OSU | A, C | operating system unthread |

This group of instructions are privileged and are only legal when executed in monitor mode.



Description: This instruction establishes a new environment and execution is started under its control. The new environment may be in a new state or in the same at a new location. The state issuing the CONT instruction has an exchange package address, CXPA, known to the emulator. Register F contains the address, NXPA, of a new exchange package area. The following establishes the new environment and the execution initiation:

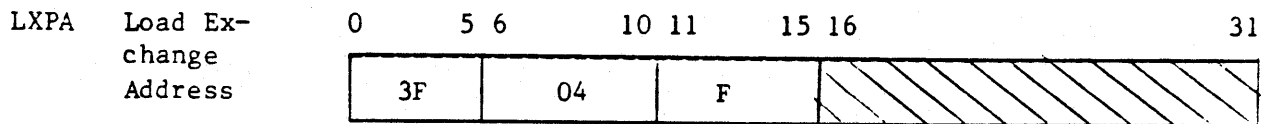
Word 1 of the new exchange package defines the non-real time interrupt mask, the bit register, the next state in which execution will continue, and provides the initial/restored fault conditions.

Word 2 of the new exchange package defines the next instruction to be executed.

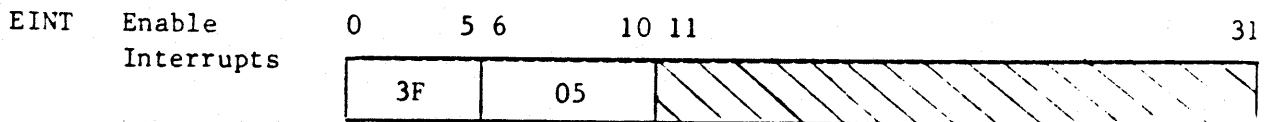
Word 3 of the new exchange package is redefined by writing the value CXPA into the SXPA field.

The NXPA becomes the new exchange package address known to the emulator, CXPA = NXPA.

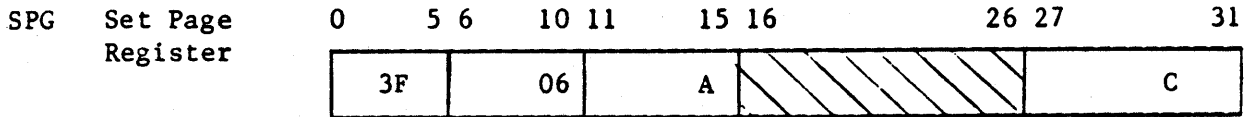
The interrupt system is enabled.



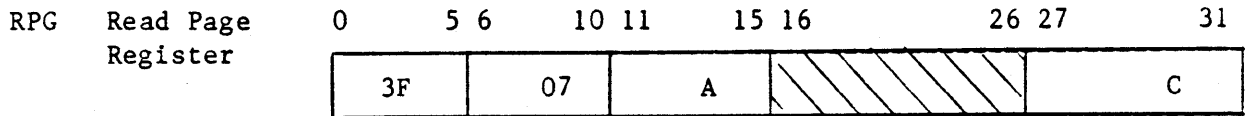
Description: This instruction returns the predecessor exchange package address to register F.



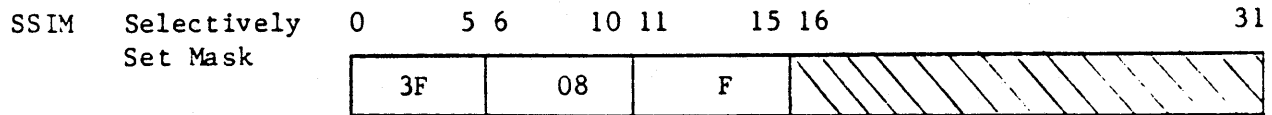
Description: Enable the real-time interrupt system.



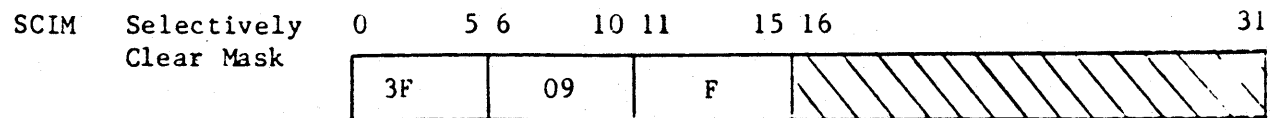
Description: Set the page register specified by the contents of register A to the contents of register C. Both the logical page (0-15) and state (0-7) are specified by the contents of register A. The state is specified in bits 20-22 and the logical page in bits 16-19.



Description: Read the page register specified by the contents of register A into register C. Register A specifies the logical page (0-15) in bits 16-19, and the state (0-7) in bit 20-22.

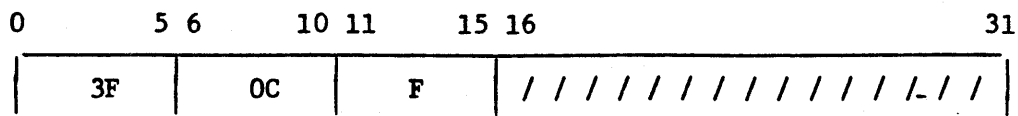


Description: For bits set in register F, set the corresponding bits in the interrupt mask register.



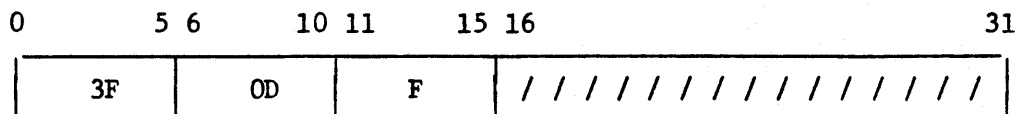
Description: For bits set in register F, clear the corresponding bits in the interrupt mask register.

SRTC Set Real-Time Clock



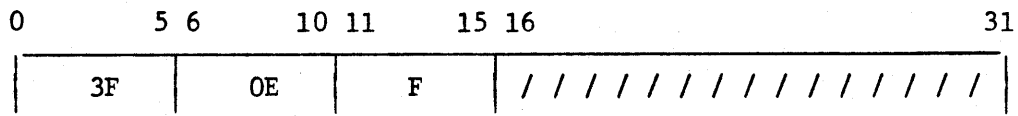
Description: Set the real-time clock to the millisecond value contained in register F.

SIT Set Interval Time Clock



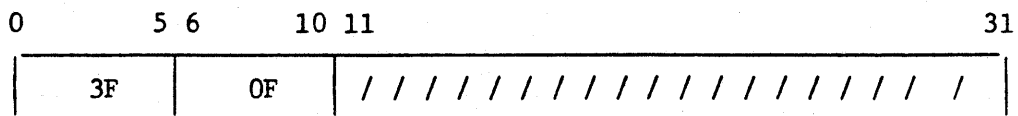
Description: Set the interval timer to the millisecond count contained in register F.

TRC Transfer Real-Time Clock

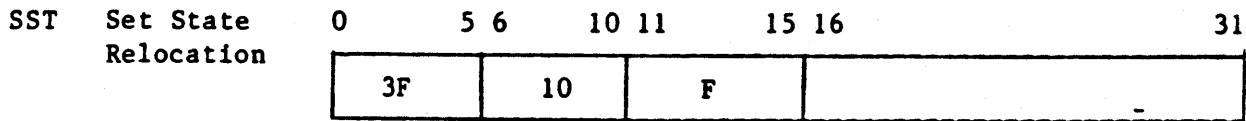


Description: Transfer the contents of the real-time clock to register F.

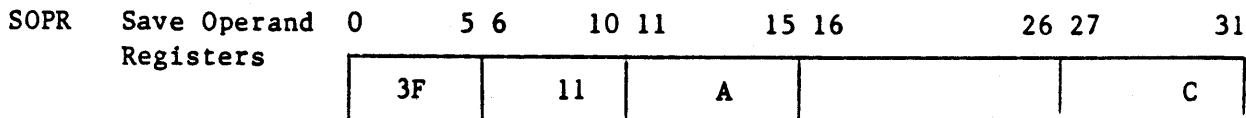
DINT Disable Interrupts



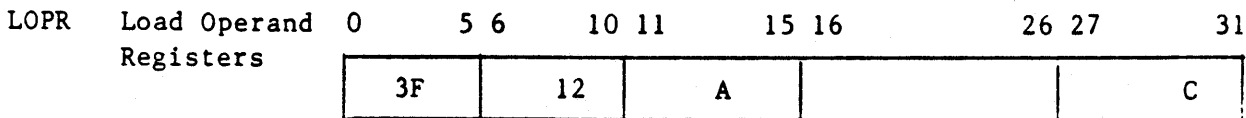
Description: Disable the recognition of maskable interrupts.



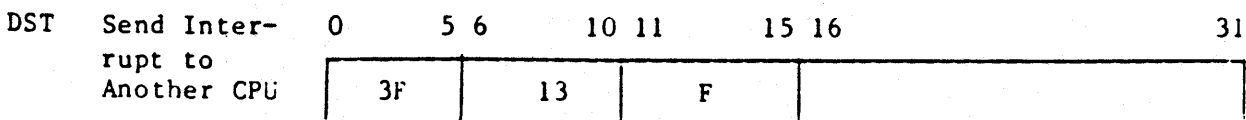
Description: Set the state relocation register to the state (0-7) specified by bits 20-22 in register F.



Description: Save the operand registers of the designated state into a memory buffer. The state is specified in register A bits 20-22 (normally obtained by loading word 1 of the exchange package). The specified register set is stored into the 31-word buffer defined by register C.

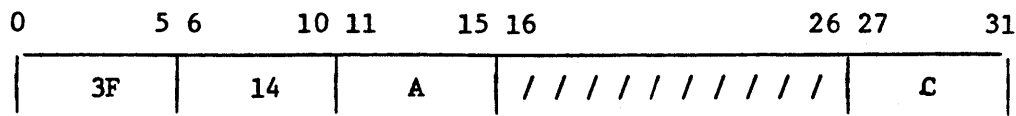


Description: Load the operand registers of the designated state from a memory buffer. The state is specified in register A bits 20-22 (normally obtained by loading word 1 of the exchange package). The specified register set is loaded from a 31-word buffer, defined by register C.



Description: Sends interrupt to CPU specified in the lower three bits of F.

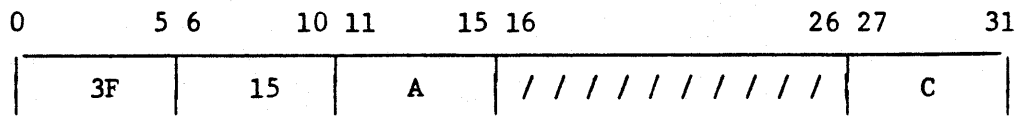
LPIR Load Page Indexes



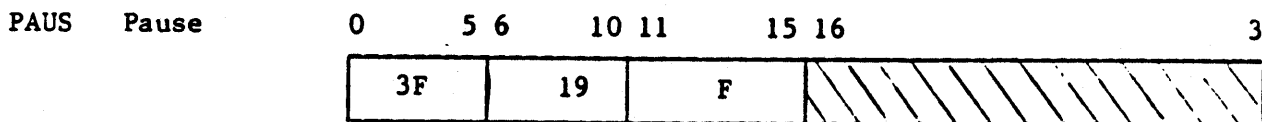
Description: This instruction sets new values into all 16 page index registers for a given state. Register A contains the state in bits 20-22 (normally obtained by loading word 1 of the exchange package). Register C contains the address of the 8 words which are to be set into the specified page register set. The 8 words are defined as follows:

| | | | | |
|---|----------|-------|--|----------|
| | 0 | 15 16 | | 31 |
| 0 | PIR - 0 | | | PIR - 1 |
| 1 | PIR - 2 | | | PIR - 3 |
| 2 | PIR - 4 | | | PIR - 5 |
| 3 | PIR - 6 | | | PIR - 7 |
| 4 | PIR - 8 | | | PIR - 9 |
| 5 | PIR - 10 | | | PIR - 11 |
| 6 | PIR - 12 | | | PIR - 13 |
| 7 | PIR - 14 | | | PIR - 15 |

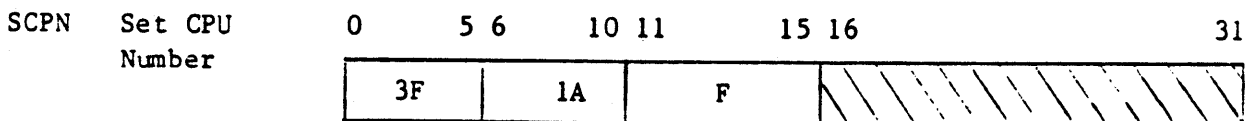
LMM Load Micromemory



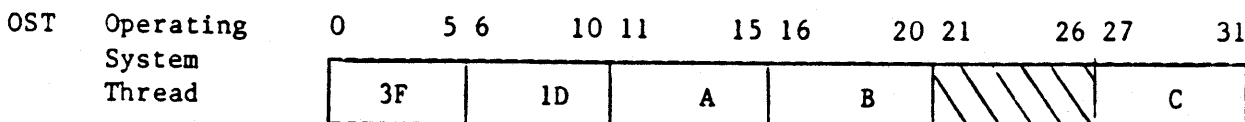
Description: Move microcode in main memory specified by the FWA in register A into micromemory specified by the micromemory address in register C.



Description: This instruction is a variable time-length NOP. Register F specifies a delay interval in micro seconds. During the pause time span, interrupts are continuously sampled. The instruction terminates when the specified time has elapsed or an interrupt is sensed.

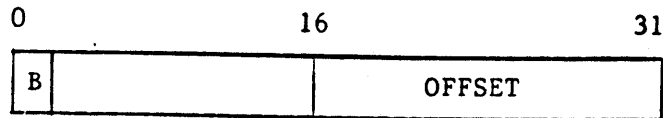


Description: This instruction defines the processor number. The processor number is contained in register F, bits 17-19. This number is stored into word 1 of each exchange package defined by the emulator during interrupt and MON instruction processing.

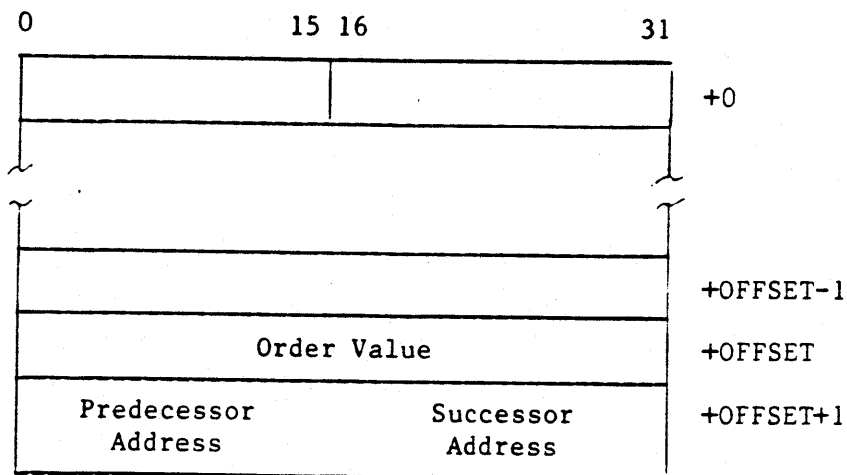


Description: This instruction implements a list threading operation by scanning through a linked list of ordered entries to find a position to enter a new entry into the list.

Register A contains the address of the new entry to be added. Register C contains the address of the top-of-list pointer for the list to be scanned. Register B contains the two quantities, a flag and an offset. Register B has the following format:



The list of entries and the new entry have the following format:

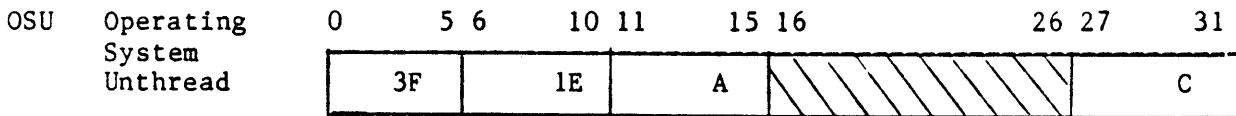


The offset is used to locate the two thread "control" words. The order value for the new entry is compared against the order value of the list entries to determine the placement of the new entry in the list. The list is maintained in descending order. If the new entry order word is the same value as one or more list entry order words, then the value of the B flag is used to determine the placement of the new entry. If B = 0, then placement is after equivalent values, and if B = 1, then placement is before equivalent values.

An empty list is denoted by a zero value in the top of list pointer. The successor address field of the last list entry contains the value zero.

When the new entry's position is identified, the new entry is inserted into the linked list by the following steps:

1. The new entry's predecessor field and successor field is defined.
2. The successor field of the predecessor is redefined to point to the new entry.
3. The predecessor field of the successor is redefined to point to the new entry.
4. The new entry is now linked into the threaded list.



Description: This instruction removes an entry from a threaded list created by the OST instruction. Register A contains the address of the list entry to be removed, while register C contains the offset in bits 16 to 31.

The removal operation is accomplished by redefining the predecessor and successor fields in the "adjacent" entries in the list, if any.

SUPPLEMENTARY FUNCTIONS

| Operation Field | Address Field | Interpretation |
|-----------------|---------------|--------------------------|
| AABL | A,B,C | address adjust bit left |
| AABR | A,B,C | address adjust bit right |
| PTHD | A,B,C | prioritized thread |
| UTHD | A,C | unthread |

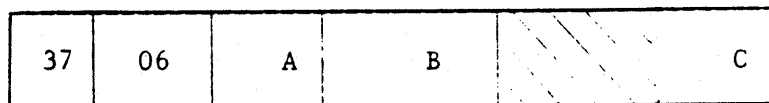
AABL Address Adjust 0 5 6 10 11 15 16 20 21 26 27 31
 Bit Left



Description: The AABL instruction computes a bit address from a word address specified in register A and a bit offset specified in register B. The result is stored in register C according to the formula:

$$C = (A) * 32 + (B)$$

AABR Address Adjust 0 5 6 10 11 15 16 20 21 26 27 31
 Bit Right



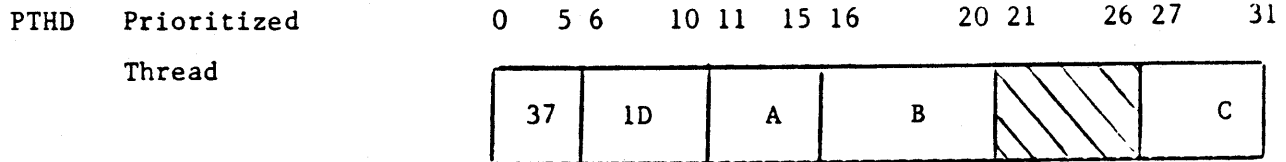
- (A) = bit address
- (B) = bit offset
- (C) = word address

Description: The AABR instruction computes the word address and bit offset from the bit address specified in register A. The bit offset is stored into register B according to the formula:

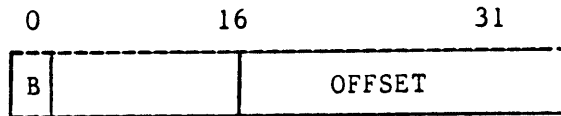
$$(B) = (A) .AND. 1F_{16}$$

The word address is stored into register C according to the formula:

$$(C) = (A)/32$$



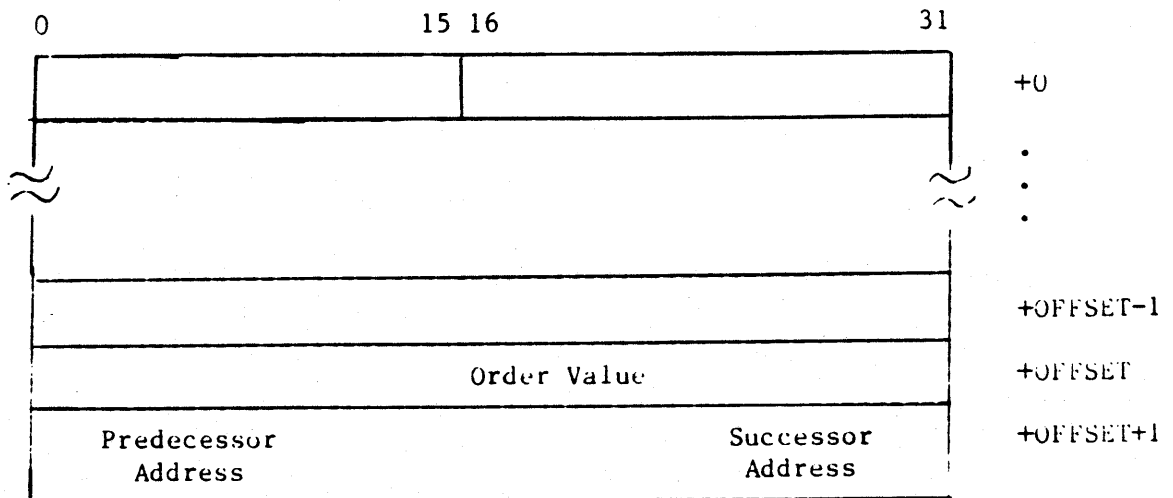
(A) = address of new entry
 (B) = flag and offset as below:



(C) = address of the top of linked list pointer

Description: This instruction implements a list threading operation by scanning through a linked list of ordered entries to find a position to enter a new entry into the list.

The list of entries has the following format:



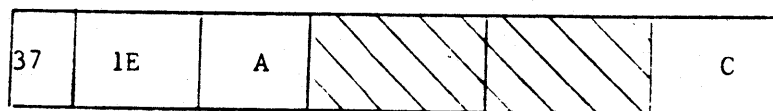
The offset is used to locate the two thread "control" words. The order value for the new entry is compared against the order value of the list entries to determine the placement of the new entry in the list. The list is maintained in descending order. If the new entry order word is the same value as one or more list entry order words, then the value of the B flag is used to determine the placement of the new entry. If B = 0, then placement is after equivalent values; if B = 1, then placement is before equivalent values.

An empty list is denoted by a zero value in the top of list pointer. The successor address field of the last list entry contains the value zero.

When the new entry's position is identified, the new entry is inserted into the linked list by the following steps:

1. The new entry's predecessor and successor fields are defined.
2. The successor field of the predecessor is redefined to point to the new entry.
3. The predecessor field of the successor is redefined to point to the new entry.
4. The new entry is now linked into the threaded list.

UTHD Unthread 0 5 6 10 11 15 16 20 21 26 27 31



(A) = address of list entry to be removed

(C) = offset in bits 16 to 31

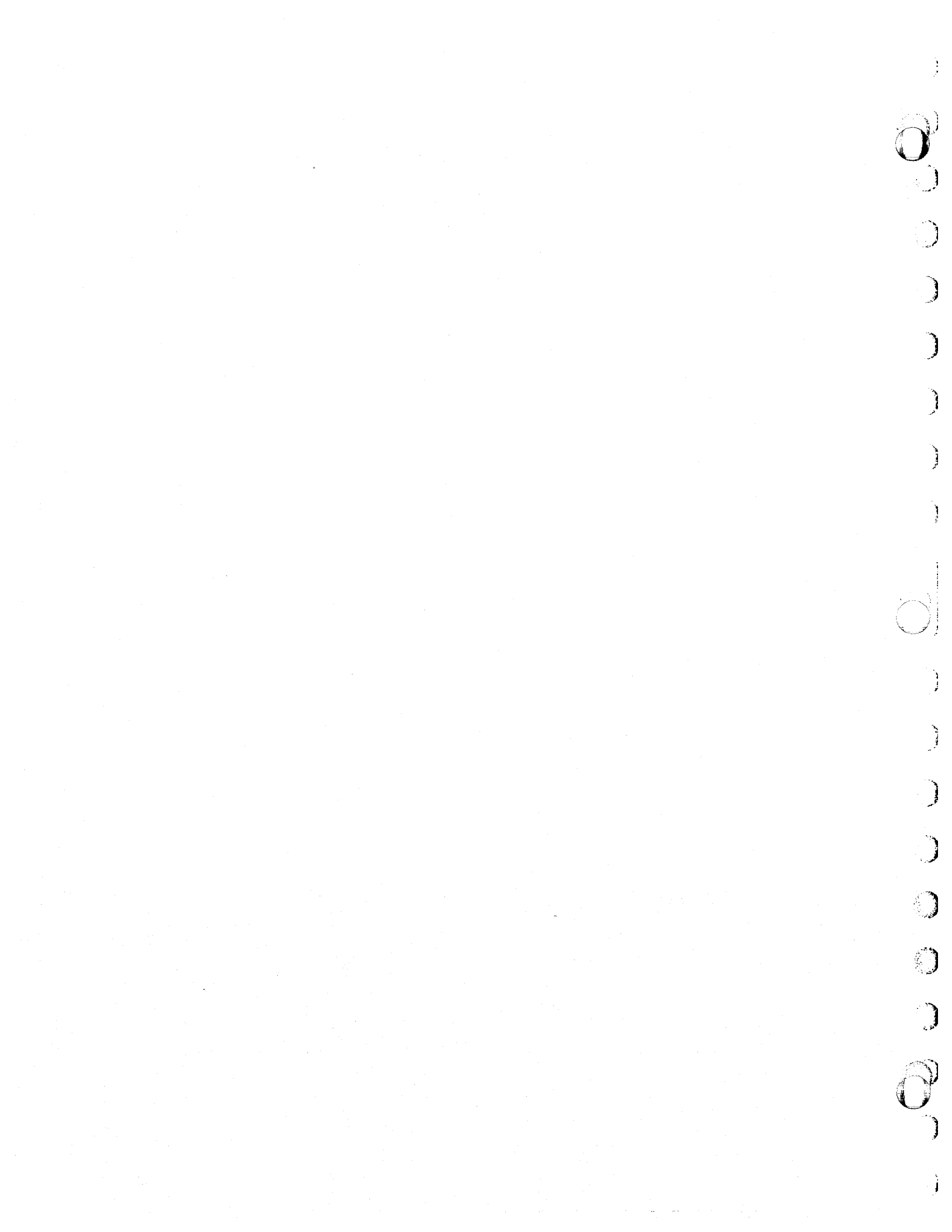
Description: This instruction removes an entry from a threaded list created by the OST or PTHD instructions.

The removal operation is accomplished by redefining the predecessor and successor fields in the "adjacent" entries in the list, if any.

POWERS OF TWO TABLE

A

| 2° | | 2° | |
|------------------------------|----|--|--|
| 1 | 0 | 10 | |
| 2 | 1 | 05 | |
| 4 | 2 | 025 | |
| 8 | 3 | 0125 | |
| 16 | 4 | 0082 5 | |
| 32 | 5 | 0031 25 | |
| 64 | 6 | 0015 625 | |
| 128 | 7 | 0007 812 5 | |
| 256 | 8 | 0003 906 25 | |
| 512 | 9 | 0001 953 125 | |
| 1 024 | 10 | 0000 976 562 5 | |
| 2 048 | 11 | 0000 488 281 25 | |
| 4 096 | 12 | 0000 244 140 625 | |
| 8 192 | 13 | 0000 122 070 312 5 | |
| 16 384 | 14 | 0000 061 035 156 25 | |
| 32 768 | 15 | 0000 030 517 578 125 | |
| 65 536 | 16 | 0000 015 258 789 062 5 | |
| 131 072 | 17 | 0000 007 629 394 531 25 | |
| 262 144 | 18 | 0000 003 814 697 265 625 | |
| 524 288 | 19 | 0000 001 907 348 632 812 5 | |
| 1 048 576 | 20 | 0000 000 953 674 316 406 25 | |
| 2 097 152 | 21 | 0000 000 476 837 158 203 125 | |
| 4 194 304 | 22 | 0000 000 238 418 579 101 562 5 | |
| 8 388 608 | 23 | 0000 000 119 209 289 550 781 25 | |
| 16 777 216 | 24 | 0000 000 059 604 644 775 390 625 | |
| 33 554 432 | 25 | 0000 000 029 802 322 387 695 312 5 | |
| 67 108 864 | 26 | 0000 000 014 901 161 193 847 656 25 | |
| 134 217 728 | 27 | 0000 000 007 450 580 596 923 828 125 | |
| 268 435 456 | 28 | 0000 000 003 725 290 298 461 914 062 5 | |
| 536 870 912 | 29 | 0000 000 001 862 645 149 230 957 031 25 | |
| 1 073 741 824 | 30 | 0000 000 000 931 322 574 615 478 515 625 | |
| 2 147 483 648 | 31 | 0000 000 000 465 661 287 307 739 257 812 5 | |
| 4 294 967 296 | 32 | 0000 000 000 232 830 643 653 869 628 906 25 | |
| 8 589 934 592 | 33 | 0000 000 000 116 415 321 826 934 814 453 125 | |
| 17 179 869 184 | 34 | 0000 000 000 058 207 660 913 467 407 226 562 5 | |
| 34 359 738 368 | 35 | 0000 000 000 029 103 830 456 733 703 613 281 25 | |
| 68 719 476 736 | 36 | 0000 000 000 014 551 915 228 366 851 806 640 625 | |
| 137 438 953 472 | 37 | 0000 000 000 007 275 957 614 183 425 903 320 312 5 | |
| 274 877 906 944 | 38 | 0000 000 000 003 637 978 807 091 712 951 660 156 25 | |
| 549 755 813 888 | 39 | 0000 000 000 001 818 989 403 545 856 475 830 078 125 | |
| 1 099 511 827 776 | 40 | 0000 000 000 000 909 494 701 772 928 237 915 039 062 5 | |
| 2 199 023 255 552 | 41 | 0000 000 000 000 454 747 350 886 464 118 957 519 531 25 | |
| 4 398 046 511 104 42 | 42 | 0000 000 000 000 227 373 675 443 232 059 478 759 765 625 | |
| 8 796 093 022 208 43 | 43 | 0000 000 000 000 113 686 837 721 616 029 739 379 882 812 5 | |
| 17 592 186 044 416 44 | 44 | 0000 000 000 000 056 843 418 860 808 014 869 689 941 406 25 | |
| 35 184 372 088 832 45 | 45 | 0000 000 000 000 028 421 709 430 404 007 434 844 970 703 125 | |
| 70 368 744 177 664 46 | 46 | 0000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5 | |
| 140 737 488 355 328 47 | 47 | 0000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25 | |
| 281 474 976 710 656 48 | 48 | 0000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625 | |
| 562 949 953 421 312 49 | 49 | 0000 000 000 000 001 776 356 839 400 250 464 677 810 668 945 312 5 | |
| 1 125 899 906 842 624 50 | 50 | 0000 000 000 000 000 888 178 419 700 125 232 338 905 334 472 656 25 | |
| 2 251 799 813 685 248 51 | 51 | 0000 000 000 000 000 444 089 209 850 062 616 189 452 667 236 328 125 | |
| 4 503 599 627 370 496 52 | 52 | 0000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5 | |
| 9 007 199 254 740 992 53 | 53 | 0000 000 000 000 000 111 022 302 462 515 654 042 363 166 809 082 031 25 | |
| 18 014 398 509 481 984 54 | 54 | 0000 000 000 000 000 055 511 151 231 257 827 021 181 583 404 541 015 625 | |
| 36 028 797 018 963 968 55 | 55 | 0000 000 000 000 000 027 755 575 615 628 913 510 590 791 702 270 507 812 5 | |
| 72 057 594 037 927 936 56 | 56 | 0000 000 000 000 000 013 877 787 807 814 456 755 295 395 851 135 253 906 25 | |
| 144 115 188 075 855 872 57 | 57 | 0000 000 000 000 000 006 938 893 903 907 228 377 647 697 925 567 626 953 125 | |
| 288 230 376 151 711 744 58 | 58 | 0000 000 000 000 000 003 469 446 951 953 614 188 823 848 962 783 813 476 562 5 | |
| 576 460 752 303 423 488 59 | 59 | 0000 000 000 000 000 001 734 723 475 976 807 094 411 924 481 391 906 738 281 25 | |
| 1 152 921 504 806 846 976 60 | 60 | 0000 000 000 000 000 000 867 361 737 988 403 547 205 962 240 695 953 369 140 625 | |



I/O PROGRAMMING FORMATS

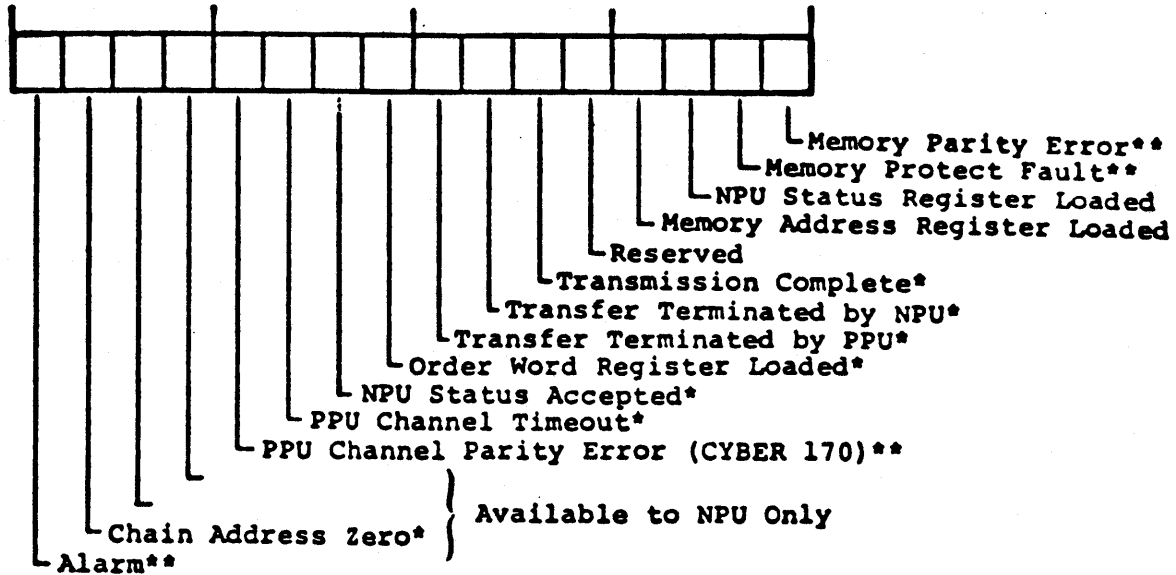
B

COMMUNICATIONS COUPLER PPU FUNCTION CODES

| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|-----------------------|----|---|---|---|---|---|---|---|---|---|---|---|
| Equip Code (=7) | 1 | 0 | 0 | 0 | 0 | X | X | X | X | X | X | Clear Coupler |
| | 0 | 1 | 0 | 0 | 0 | X | X | X | X | X | X | Clear CP |
| | 0 | 0 | 1 | 0 | 0 | X | X | X | X | X | X | Stop CP |
| | 0 | 0 | 0 | 1 | 0 | X | X | X | X | X | X | Start CP |
| | X | X | X | X | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Input Memory Address Zero* |
| | X | X | X | X | 0 | 0 | 0 | 1 | 0 | 0 | 1 | Input Memory Address One* |
| | X | X | X | X | 0 | 0 | 1 | 0 | | | | |
| | X | X | X | X | 0 | 0 | 1 | 1 | | | | Input Data |
| | X | X | X | X | 0 | 1 | 0 | 0 | | | | Input CP Status |
| | X | X | X | X | 0 | 1 | 0 | 1 | | | | Input Coupler Status |
| | X | X | X | X | 0 | 1 | 1 | 0 | | | | Input Order Word* |
| | X | X | X | X | 0 | 1 | 1 | 1 | | | | Input Program |
| | X | X | X | X | 1 | 0 | 0 | 0 | | | | Output Memory Address Zero (upper byte) |
| | X | X | X | X | 1 | 0 | 0 | 1 | | | | Output Memory Address One (lower byte) |
| | X | X | X | X | 1 | 0 | 1 | 0 | | | | |
| | X | X | X | X | 1 | 0 | 1 | 1 | | | | |
| | X | X | X | X | 1 | 1 | 0 | 0 | | | | Output Data |
| | X | X | X | X | 1 | 1 | 0 | 1 | | | | Output Program |
| | X | X | X | X | 1 | 1 | 1 | 0 | | | | Output Order Word |
| | X | X | X | X | 1 | 1 | 1 | 1 | | | | |

*Hardware Maintenance feature

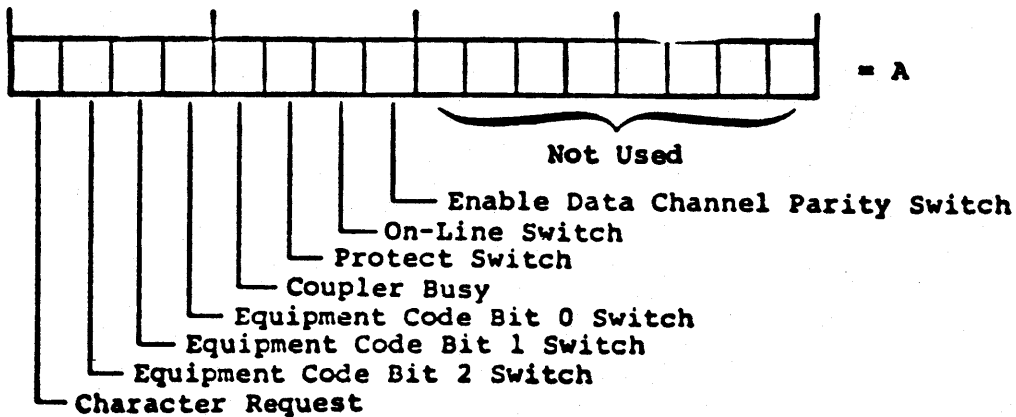
COMMUNICATIONS COUPLER STATUS FORMAT



**Alarm Condition (all alarms generate NPU Interrupt)
 *NPU Interrupt Condition

NOTE: All non-alarm interrupt conditions except OWRL are cleared by input coupler status command, clear coupler function, and clear coupler command. OWRL interrupt condition is cleared by input OW command, clear coupler function and command, and Master Clear. Alarm interrupt conditions are cleared only by clear coupler functions and commands and Master Clear.

COMMUNICATIONS COUPLER INPUT SWITCH STATUS FORMAT

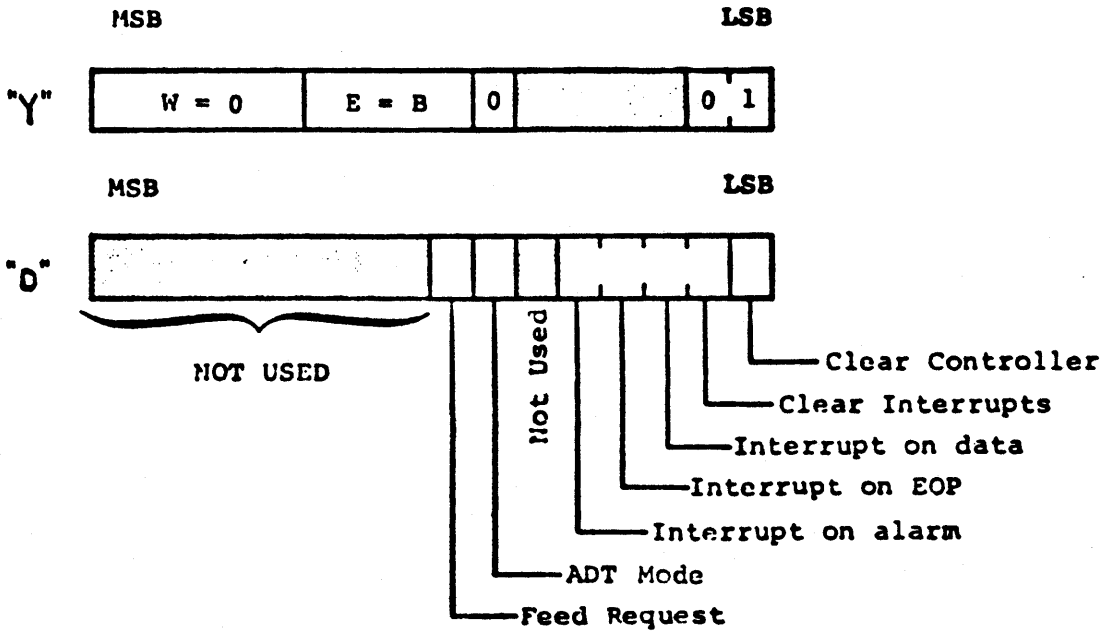


COMMUNICATIONS COUPLER - CP SET/SAMPLE INSTRUCTION FORMAT

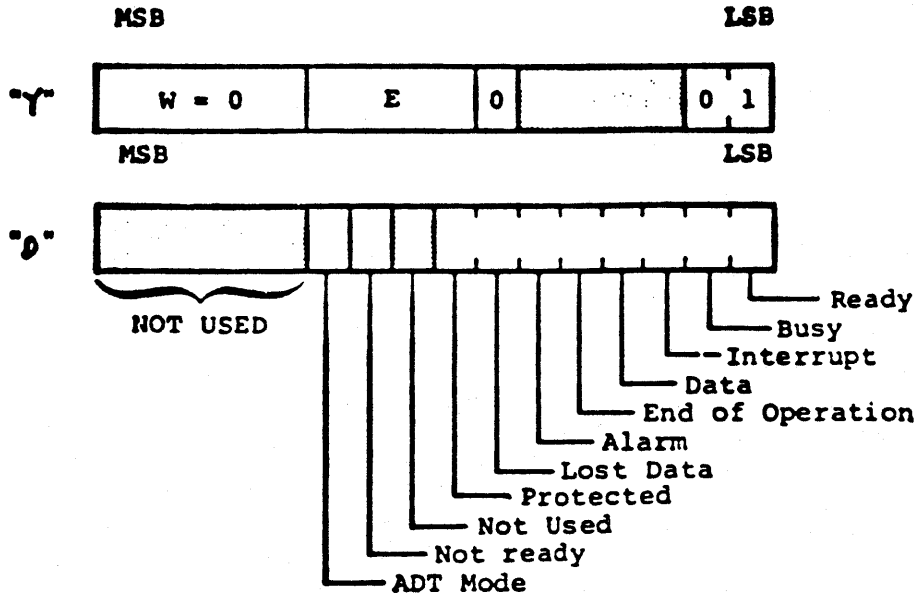
| 0 0 0 0 | | | | 0 1 E E | | E | | | |
|---|--|--------------------------------------|-------|---------|--|---|----------|--|--|
| Equipment Address | | | | | | | Not Used | | |
| First Coupler = 1100 ('C') (Standard) | Second Coupler = 1101 ('D') (Optional) | Sample (Read) | 0 0 0 | 0 0 | Input Memory Address Zero* | | | | |
| | | | 0 0 1 | 0 0 | Input Memory Address | | | | |
| | | | 0 1 0 | 0 0 | Input First/Present Character Displacement | | | | |
| | | | 0 1 1 | 0 0 | Input CP Status* | | | | |
| | | | 1 0 0 | 0 0 | Input Coupler Status | | | | |
| | | | 1 0 1 | 0 0 | Input Order Word | | | | |
| | | | 1 1 0 | 0 0 | Input IO* | | | | |
| | | | 1 1 1 | 0 0 | | | | | |
| | | | 0 0 0 | 0 1 | Input Last Word From Data Channel* | | | | |
| | | | 0 0 1 | 0 1 | Input FDMAR0/FDMAR1* | | | | |
| 0 1 0 | 0 1 | Input FDMAR0/Flag Mux* | | | | | | | |
| 0 1 1 | 0 1 | Input FDMAR1/Flag Mux/Flag Register* | | | | | | | |
| 1 0 0 | 0 1 | Input Switch Status | | | | | | | |
| 1 0 1 | 0 1 | Input Character* | | | | | | | |
| Set (Write) | | | 0 0 0 | 1 0 | Output Memory Address Zero* | | | | |
| | | | 0 0 1 | 1 0 | | | | | |
| | | | 0 1 0 | 1 0 | | | | | |
| | | | 0 1 1 | 1 0 | Output FCD, PCD, LCD* | | | | |
| | | | 1 0 0 | 1 0 | Output CP Status | | | | |
| | | | 1 0 1 | 1 0 | Output Buffer Length | | | | |
| | | | 1 1 0 | 1 0 | Output Order Word* | | | | |
| | | | 1 1 1 | 1 0 | | | | | |
| 0 0 0 | 1 1 | Clear Coupler | | | | | | | |
| 0 0 1 | 1 1 | Terminate Transfer | | | | | | | |
| 0 1 0 | 1 1 | | | | | | | | |
| 0 1 1 | 1 1 | | | | | | | | |
| 1 0 0 | 1 1 | Output Test* | | | | | | | |
| 1 0 1 | 1 1 | Input Test* | | | | | | | |
| 1 1 0 | 1 1 | Output Memory Address | | | | | | | |
| 1 1 1 | 1 1 | Output Character* | | | | | | | |

*Hardware maintenance feature

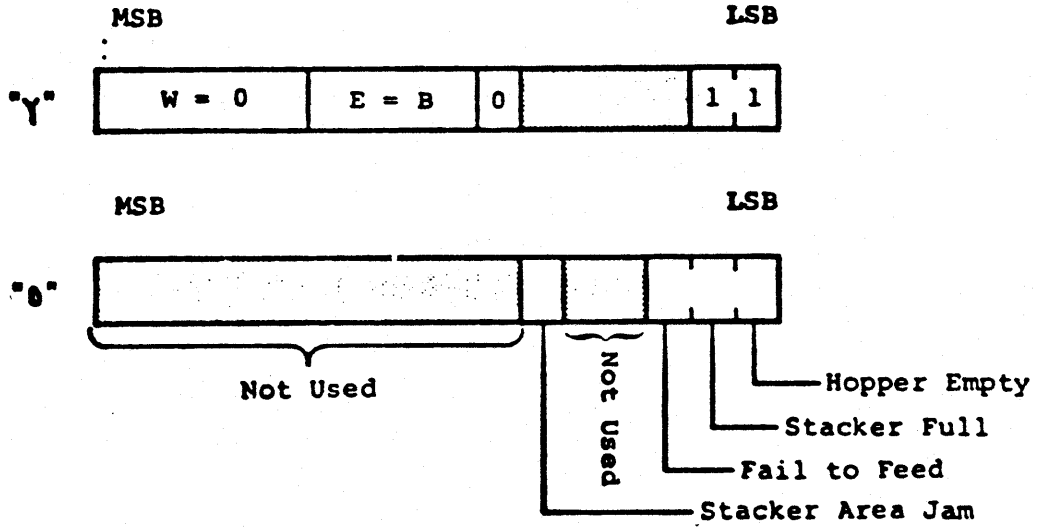
PERIPHERAL (CARD READER) CONTROLLER - DIRECTOR FUNCTION FORMAT



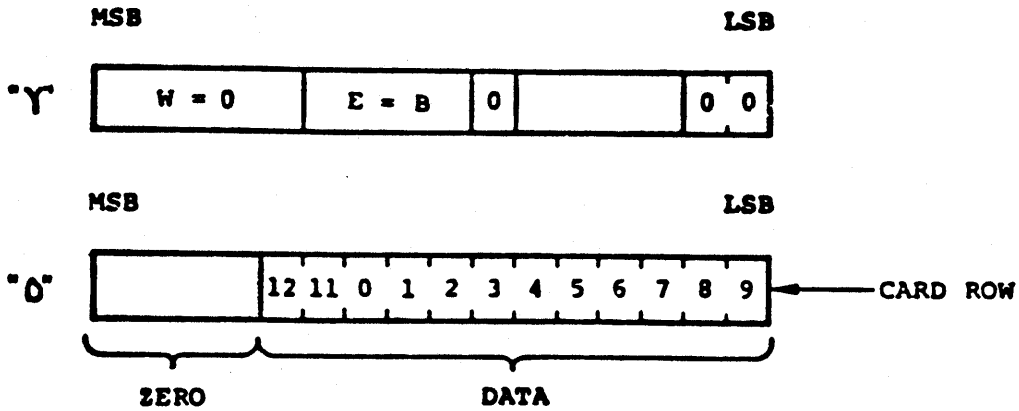
PERIPHERAL (CARD READER) CONTROLLER - DIRECTOR STATUS 1 FORMAT



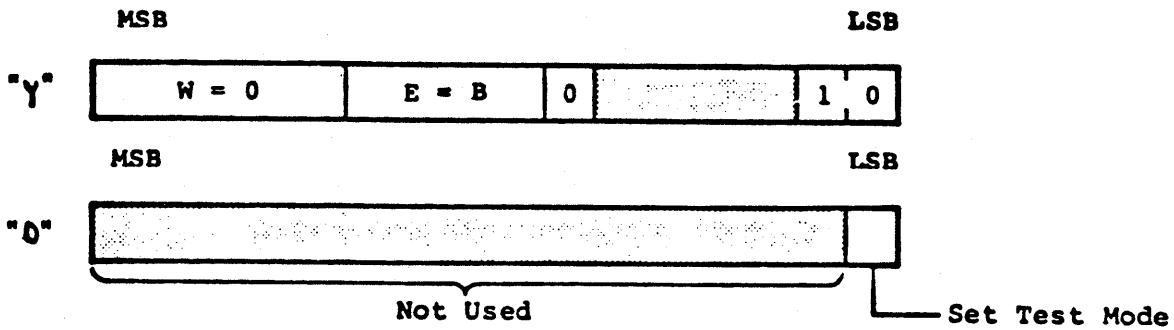
PERIPHERAL (CARD READER) CONTROLLER - DIRECTOR STATUS 2 FORMAT



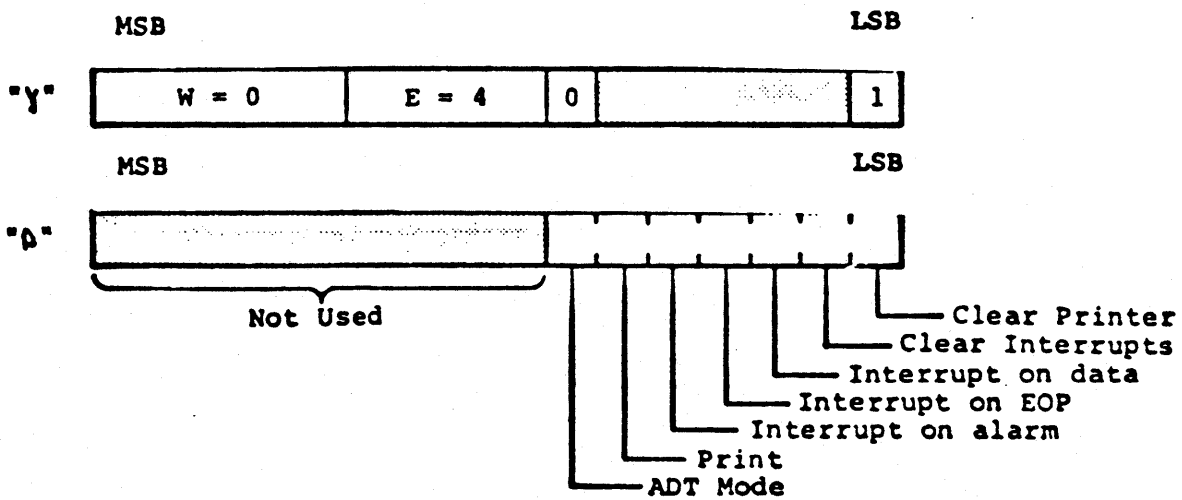
PERIPHERAL (CARD READER) CONTROLLER - DATA TRANSFER COMMAND FORMAT



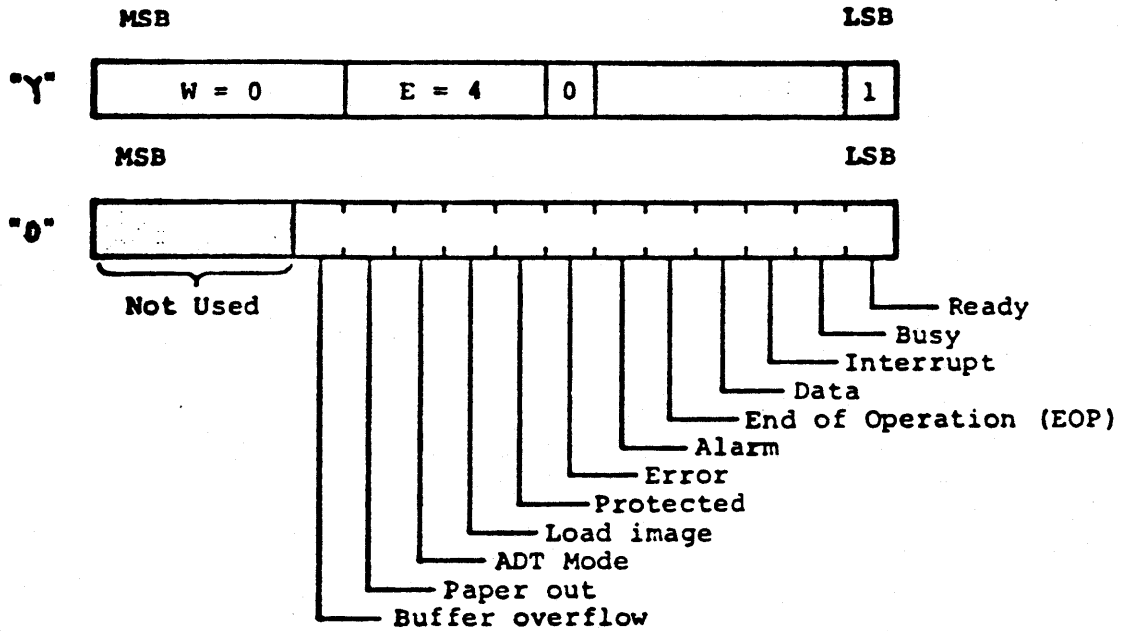
PERIPHERAL CONTROLLER - SELF-TEST COMMAND FORMAT



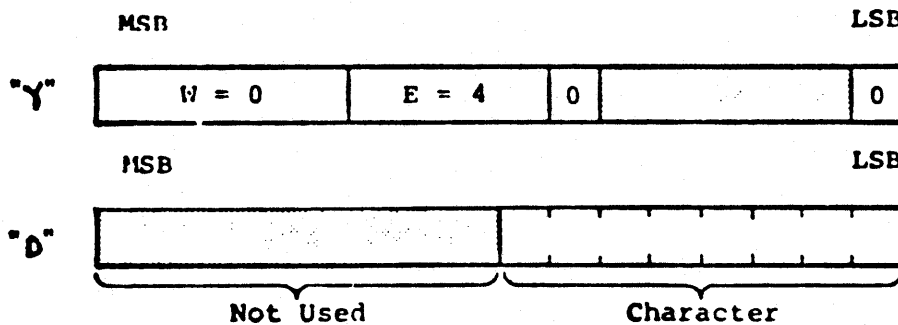
PERIPHERAL (LINE PRINTER) CONTROLLER - DIRECTOR FUNCTION FORMAT

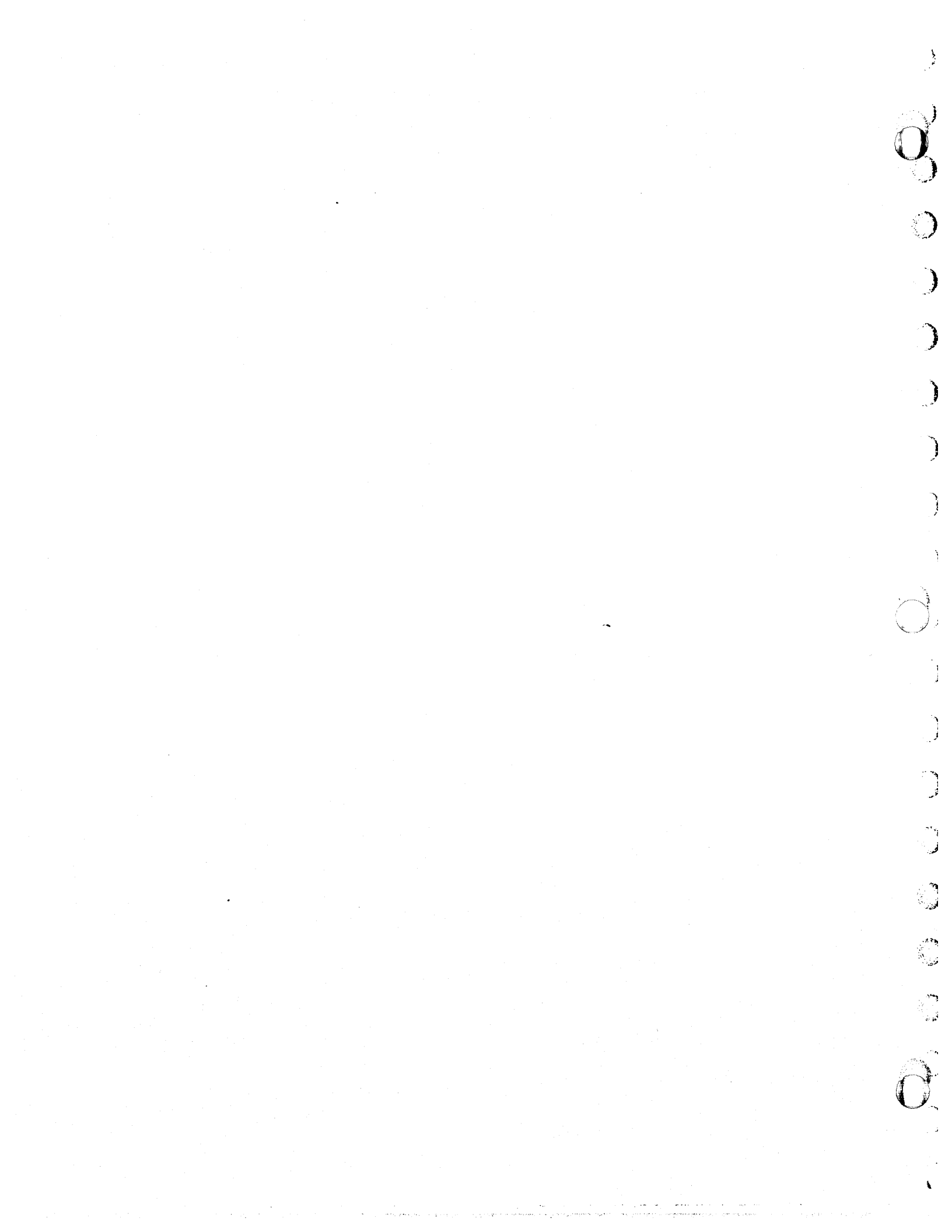


PERIPHERAL (LINE PRINTER) CONTROLLER - DIRECTOR STATUS RESPONSE FORMAT



PERIPHERAL (LINE PRINTER) CONTROLLER - DATA TRANSFER COMMAND FORMAT

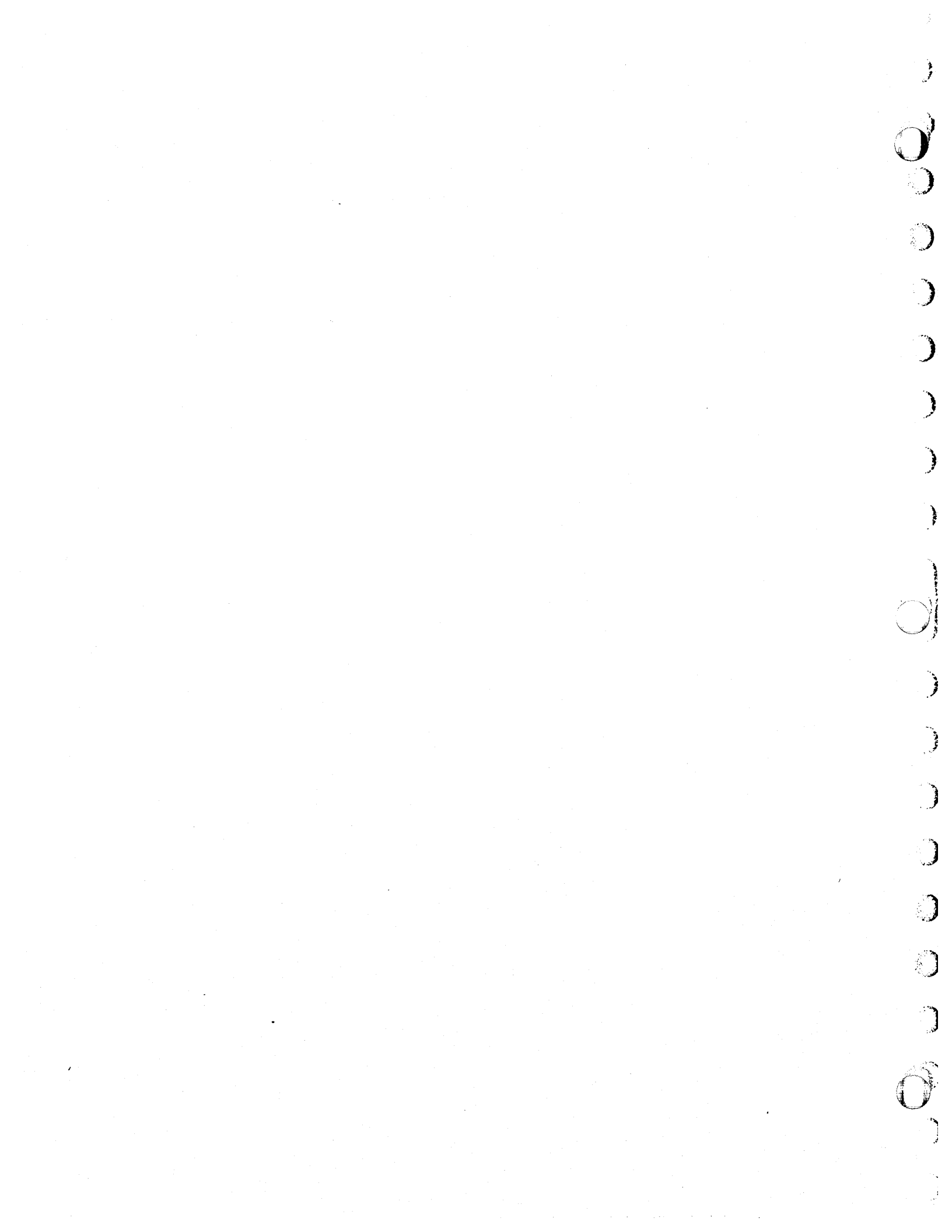




HEXADECIMAL CONVERSION TABLES

C

| HEX. | DEC. | HEX. | DEC. | HEX. | DEC. | HEX. | DEC. | HEX. | DEC. |
|--------|----------|---------|-----------|----------|-----------|----------|------------|----------|------------|
| 1 | 1 | 10 | 16 | 100 | 256 | 1000 | 4096 | 10000 | 65536 |
| 2 | 2 | 20 | 32 | 200 | 512 | 2000 | 8192 | 20000 | 131072 |
| 3 | 3 | 30 | 48 | 300 | 768 | 3000 | 12288 | 30000 | 196608 |
| 4 | 4 | 40 | 64 | 400 | 1024 | 4000 | 16384 | 40000 | 262144 |
| 5 | 5 | 50 | 80 | 500 | 1280 | 5000 | 20480 | 50000 | 327680 |
| 6 | 6 | 60 | 96 | 600 | 1536 | 6000 | 24576 | 60000 | 393216 |
| 7 | 7 | 70 | 112 | 700 | 1792 | 7000 | 28672 | 70000 | 458752 |
| 8 | 8 | 80 | 128 | 800 | 2048 | 8000 | 32768 | 80000 | 524288 |
| 9 | 9 | 90 | 144 | 900 | 2304 | 9000 | 36864 | 90000 | 589824 |
| A | 10 | A0 | 160 | A00 | 2560 | A000 | 40960 | A0000 | 655360 |
| B | 11 | B0 | 176 | B00 | 2816 | B000 | 45056 | B0000 | 720896 |
| C | 12 | C0 | 192 | C00 | 3072 | C000 | 49152 | C0000 | 786432 |
| D | 13 | D0 | 208 | D00 | 3328 | D000 | 53248 | D0000 | 851968 |
| E | 14 | E0 | 224 | E00 | 3584 | E000 | 57344 | E0000 | 917504 |
| F | 15 | F0 | 240 | F00 | 3840 | F000 | 61440 | F0000 | 983040 |
| HEX. | DEC. | HEX. | DEC. | HEX. | DEC. | HEX. | DEC. | HEX. | DEC. |
| 100000 | 1048576 | 1000000 | 16777216 | 10000000 | 268435456 | 20000000 | 536870912 | 30000000 | 805306368 |
| 200000 | 2097152 | 2000000 | 33554432 | 30000000 | 50331648 | 40000000 | 1073741824 | 50000000 | 1342177280 |
| 300000 | 3145728 | 3000000 | 50331648 | 40000000 | 67108864 | 50000000 | 1342177280 | 60000000 | 1610612736 |
| 400000 | 4194304 | 4000000 | 67108864 | 50000000 | 83886080 | 60000000 | 1610612736 | 70000000 | 1879048192 |
| 500000 | 5242880 | 5000000 | 83886080 | 60000000 | 100663296 | 70000000 | 1879048192 | 80000000 | 2147483648 |
| 600000 | 6291456 | 6000000 | 100663296 | 70000000 | 117440512 | 80000000 | 2147483648 | 90000000 | 2415919104 |
| 700000 | 7340032 | 7000000 | 117440512 | 80000000 | 134217728 | 90000000 | 2415919104 | A0000000 | 2684354560 |
| 800000 | 8388608 | 8000000 | 134217728 | 90000000 | 150994944 | A0000000 | 2684354560 | B0000000 | 2952790016 |
| 900000 | 9437184 | 9000000 | 150994944 | A0000000 | 167772160 | B0000000 | 2952790016 | C0000000 | 3221225472 |
| A00000 | 10485760 | A000000 | 167772160 | B0000000 | 184549376 | D0000000 | 3489660928 | E0000000 | 3758096384 |
| B00000 | 11534336 | B000000 | 184549376 | C0000000 | 201326592 | E0000000 | 3758096384 | F0000000 | 4026531840 |
| C00000 | 12582912 | C000000 | 201326592 | D0000000 | 218103808 | F0000000 | 4026531840 | | |
| D00000 | 13631488 | D000000 | 218103808 | | | | | | |
| E00000 | 14680064 | E000000 | 234881024 | | | | | | |
| F00000 | 15728640 | F000000 | 251658240 | | | | | | |



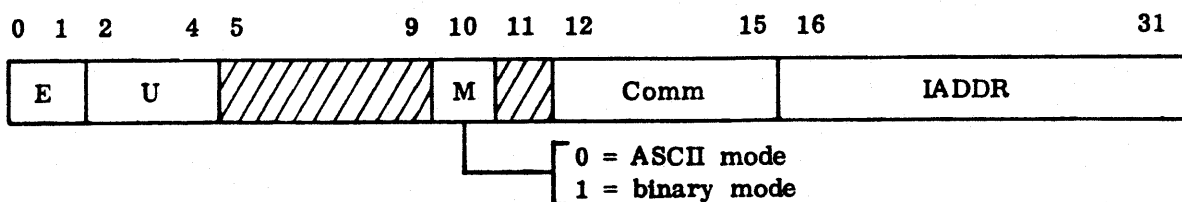
IOC INTERFACE

D

This appendix describes the operational characteristics of the IOC interface.

COMMAND FORMAT

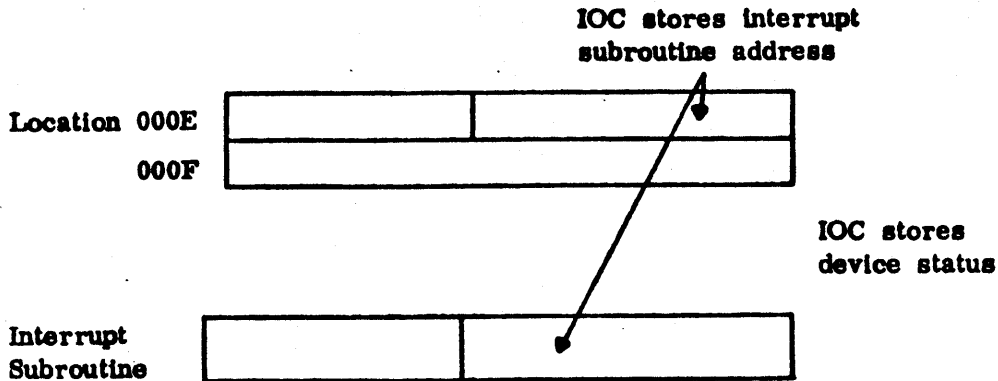
The format of the I/O controller (IOC) command word is shown below. All IOCs use the same command format for all peripheral equipment.



Meaning:

- E - 2-bit device equipment number (0-3) for the referenced IOC
- U - 3-bit device unit number (0-7); applicable to multiunit equipment
- M - 1-bit data mode indicator. A set bit indicates binary mode data; a clear bit indicates ASCII (character) data
- Comm - 4-bit device operation command (0-F₁₆)
- IADDR - 16-bit interrupt subroutine address. The interrupt subroutine address is stored by the IOC in the lower 16 bits of the first interrupt cell in memory assigned to the referenced IOC (refer to Table 3-2) upon completion of the requested command. In addition, the device status is stored in the lower 16 bits of the interrupt subroutine address.

For example (IOC 0):



Note: The interrupt subroutine address is restricted to the lower 8192 locations of memory

COMMAND TABLE FORMAT

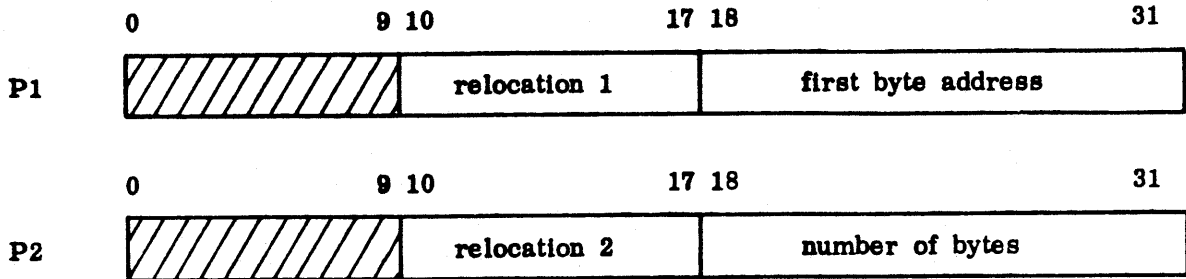
The command word is included as the first word in the command table. The command table occupies memory locations 0001 through 0005.

| <u>Location</u> | | | <u>Contents</u> |
|-----------------|---------|-------|-----------------|
| 0001 | Command | IADDR | Command word |
| 0002 | | | P1 |
| 0003 | | | P2 |
| 0004 | | | P3 |
| 0005 | | | P4 |

The start IOC (SIO) instruction signals the referenced IOC (0-3) to read one or more words from the command table. The number of parameters associated with a command (P1-P4) and the format of the parameters may vary by IOC and device type. However, P1 and P2 generally specify data buffer control.

BUFFER SPECIFICATION FORMAT

The data buffering region for a read or write command is specified by parameter words P1 and P2.



The starting address (P1) indicates the first byte address (lower 14 bits) of the data buffer in the buffer's logical page. The relocation 1 field (bits 10-17) specifies the buffer starting physical page, the buffer length (P2) contains the buffer length in number of bytes, and the relocation 2 field specifies the buffer ending physical page. A buffer length of zero (bits 18-31 of P2) specifies a transfer of the maximum allowable size, 16,384 bytes or 4096 words. A buffer may cross page boundaries, in which case the relocation fields of P1 and P2 will be different values. It is the responsibility of the I/O initiating routine to ensure that memory protection violations do not occur. Certain devices or modes of operation may only transfer data in whole words. For these devices, the IOC assumes that the first byte address is on a word boundary and that the length specifies the number of words times four.

DEVICE COMMANDS

An IOC may accept up to 16 commands per equipment and unit. In general the command structure is as follows:

| <u>Command Number</u> | <u>Function</u> |
|-----------------------|---------------------------------|
| 0 | Rewind unit/return to zero seek |
| 1 | Read (unit to memory) |
| 2 | Write (memory to unit) |
| 3 | Write end of file/format disk |
| 4 | Erase/autoload disk |
| 5 | Backspace one record |
| 6 | Search end of file forward |
| 7 | Undefined |
| 8 | Unload unit |
| 9 | Read |

Command Number

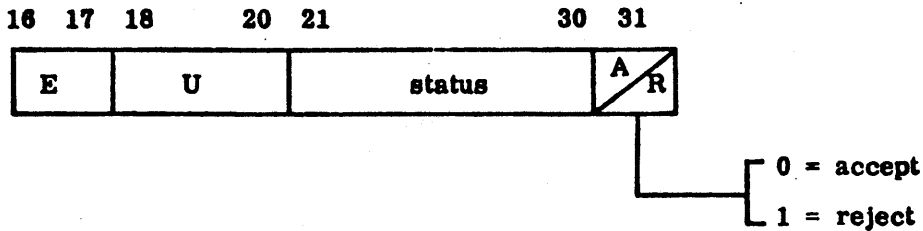
Function

| | |
|---|-----------------------------|
| A | Write |
| B | Write end of file |
| C | Erase |
| D | Backspace |
| E | Search end of file backward |
| F | Undefined |

A given command may be undefined for certain device types or may differ as to function.

DEVICE STATUS

The format of the status word returned by an IOC is as follows:



Meaning:

- E - 2-bit device equipment number
- U - 3-bit device unit number
- Status - One or more set bits to indicate device conditions
- A/R - Zero signifies no errors detected; one signifies an error condition was detected.

Certain bits in the status word are defined for all device types. The remaining bits in the status are device dependent. The defined bit positions are as follows:

| <u>Bit No.</u> | <u>Meaning</u> |
|----------------|---|
| 30 | 1 = device not ready |
| 28 | 1 = binary data mode; 0 = ASCII data mode |
| 26 | 1 = memory error detected |

Additional information on specific peripheral devices is contained in the MP-60 system peripheral reference manual.

MP-60 MACHINE LANGUAGE INSTRUCTION FORMATS

E

BIT NUMBER IN INSTRUCTION WORD

| FORMAT | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 |
|--------|---------------------|--------|------------------------|-----------------------------|--------------------|---|---|---|---|---|-------------------------|---|---|---|---|---|---|---|---|---|---|---|
| | 1 1 1 1 1 1 1 1 1 1 | | | | | | | | | | 2 2 2 2 2 2 2 2 2 2 3 3 | | | | | | | | | | | |
| 1 | OP CODE | X | b -- BASE BIT ADDRESS | | | | | | | | | | | | | | | | | | | |
| 2 | OP CODE | F | y -- IMMEDIATE OPERAND | | | | | | | | | | | | | | | | | | | |
| 3 | OP CODE | F | XC | c -- BASE BYTE ADDRESS | | | | | | | | | | | | | | | | | | |
| 4 | OP CODE | F | XH | h -- BASE HALF WORD ADDRESS | | | | | | | | | | | | | | | | | | |
| 5 | OP CODE | F | X | m -- BASE WORD ADDRESS | | | | | | | | | | | | | | | | | | |
| 6 | OP CODE | F | X | y -- IMMEDIATE OPERAND | | | | | | | | | | | | | | | | | | |
| 7 | OP CODE | SUB OP | A | B | D - RELATIVE ADDR. | | | | | | | | | | | | | | | | | |
| 8 | OP CODE | SUB OP | A | BIT | D - RELATIVE ADDR. | | | | | | | | | | | | | | | | | |
| 9 | OP CODE | SUB OP | A | B | UNUSED | | | | | | | | | | | | | C | | | | |
| 10 | OP CODE | SUB OP | X | m -- BASE WORD ADDRESS | | | | | | | | | | | | | | | | | | |
| 11 | OP CODE | SUB OP | X | y -- IMMEDIATE OPERAND | | | | | | | | | | | | | | | | | | |

DEFINITION OF FIELDS IN INSTRUCTIONS BY FORMAT

- 1 F = FILE (0--31) b = 21 BIT MEMORY BIT ADDRESS
- 2 F = FILE (0--31) y = 21 BIT UNSIGNED IMMEDIATE OPERAND
- 3 F = FILE (0--31) XC = INDEX (0--7) c = 18 BIT BYTE ADDRESS
- 4 F = FILE (0--31) XH = INDEX (0--15) h = 17 BIT HALF WORD ADDRESS
- 5 F = FILE (0--31) X = INDEX (0--31) m = 16 BIT WORD ADDRESS
- 6 F = FILE (0--31) X = INDEX (0--31) Y = 16 BIT SIGNED IMMEDIATE
- 7 A = OPERAND FILE (0--31) B = OPERAND FILE (0--31) D = JUMP ADDRESS
- 8 A = OPERAND FILE (0--31) BIT = BIT NUMBER (0--31) D = JUMP ADDRESS
- 9 A = OPERAND 1 (0--31) B = OPERAND 2 (0--31) C = DESTINATION (0--31)
- 10 X = INDEX OR FILE (0--31) m = 16 BIT WORD ADDRESS
- 11 X = FILE (0--31) y = 16 BIT SIGNED IMMEDIATE OPERAND

INDEXING MODES:

$$B = b + (X)$$

$$C = c + (XC)$$

$$Y = h + (XH)$$

$$Y = y + (X)$$

$$M = m + (X)$$

NOTE: REGISTER ZERO MUST BE ZERO

INSTRUCTION CROSS REFERENCE

| <u>Op Code</u> | <u>Page</u> | <u>Format</u> | <u>Time</u> |
|----------------|-------------|---------------|---------------------|
| NOP | 00 | 6-8 | 5 0.676 |
| LDB | 01 | 6-9 | 1 2.8 |
| LDC | 02 | 6-10 | 3 1.568 |
| LDH | 03 | 6-10 | 4 1.568 |
| LDI | 04 | 6-10 | 6 0.896 or 1.75 |
| LDHA | 05 | 6-11 | 4 1.232 |
| LDCA | 06 | 6-11 | 3 1.232 |
| LDBA | 07 | 6-11 | 1 1.008 |
| LDA | 08 | 6-12 | 5 0.896 |
| LD | 09 | 6-12 | 5 1.446 |
| LDD | 0A | 6-12 | 5 2.128 |
| STB | 0B | 6-15 | 1 2.856 |
| STC | 0C | 6-15 | 3 1.232 |
| STH | 0D | 6-16 | 4 1.232 |
| STHA | 0E | 6-16 | 5 2.128 |
| STCA | 0F | 6-16 | 5 2.128 |
| STBA | 10 | 6-17 | 5 2.128 |
| ST | 11 | 6-17 | 5 1.076 |
| STD | 12 | 6-17 | 5 1.15 |
| ADI | 13 | 6-19 | 6 1.12 or 1.288 |
| AD | 14 | 6-19 | 5 1.456 |
| ADD | 15 | 6-20 | 5 2.345 |
| FAD | 16 | 6-24 | 5 2.52 - 5.8 - 9.24 |
| FADD | 17 | 6-24 | 5 |
| SB | 18 | 6-20 | 5 1.456 |
| SBD | 19 | 6-20 | 5 2.296 |
| FSB | 1A | 6-25 | 5 2.52 - 5.8 - 9.24 |
| FSBD | 1B | 6-25 | 5 |
| MPI | 1C | 6-21 | 6 4.032 -- 4.928 |
| MP | 1D | 6-21 | 5 |
| FMP | 1E | 6-25 | 5 |
| FMPD | 1F | 6-26 | 5 X |

INSTRUCTION CROSS REFERENCE (Contd)

| | <u>Op Code</u> | | <u>Page</u> | <u>Format</u> | <u>Time</u> |
|-------|----------------|-------|-------------|---------------|-----------------|
| | DV | 20 | 6-21 | 5 | |
| | FDV | 21 | 6-26 | 5 | X |
| | FDVD | 22 | 6-26 | 5 | X |
| LEFT | SF | 23 | 6-28 | 6 | 1.512 + .056(N) |
| RIGHT | SF | 23 | 6-28 | 6 | 1.904 + .056(N) |
| LEFT | SFD | 24 | 6-28 | 6 | 2.016 + .056(N) |
| RIGHT | SFD | 23 | 6-28 | 6 | 2.408 + .056(N) |
| | RAD | 25 | 6-22 | 5 | 2.016 |
| | AND | 26 | 6-29 | 1 | 1.008 or 2.968 |
| | OR | 27 | 6-29 | 1 | 1.008 or 2.968 |
| | XOR | 28 | 6-30 | 1 | 2.8 |
| | LP | 29 | 6-30 | 2 | 1.008 |
| | LOR | 2A | 6-31 | 2 | 1.008 |
| | LXR | 2B | 6-31 | 2 | 1.008 |
| | TST,GE | 2C.00 | 6-32 | 7 | 1.344 -- 2.464 |
| | TST,LE | 2C.01 | 6-32 | 7 | 1.344 -- 2.464 |
| | TST,EQ | 2C.02 | 6-33 | 7 | 1.12 -- 2.464 |
| | TST,NE | 2C.03 | 6-33 | 7 | 1.344 -- 2.352 |
| | TST,GT | 2C.04 | 6-33 | 7 | 1.344 -- 2.464 |
| | TST,LT | 2C.05 | 6-33 | 7 | 1.344 -- 2.464 |
| | TSTF,GE | 2C.08 | 6-34 | 7 | 1.344 -- 2.632 |
| | TSTF,LE | 2C.09 | 6-34 | 7 | 1.344 -- 2.632 |
| | TSTF,EQ | 2C.0A | 6-34 | 7 | 1.12 -- 2.464 |
| | TSTF,NE | 2C.0B | 6-34 | 7 | 1.344 -- 2.352 |
| | TSTF,GT | 2C.0C | 6-35 | 7 | 1.344 -- 2.632 |
| | TSTF,LT | 2C.0D | 6-35 | 7 | 1.344 -- 2.632 |

INSTRUCTION CROSS REFERENCE (Contd)

| <u>Op Code</u> | | <u>Page</u> | <u>Format</u> | <u>Time</u> |
|----------------|-------|-------------|---------------|----------------|
| TSTD,GE | 2D.00 | 6-35 | 7 | 1.736 -- 1.136 |
| TSTD,LE | 2D.01 | 6-35 | 7 | 1.736 -- 1.136 |
| TSTD,EQ | 2D.02 | 6-36 | 7 | 1.568 -- 2.744 |
| TSTD,NE | 2D.03 | 6-36 | 7 | 1.848 -- 3.08 |
| TSTD,GT | 2D.04 | 6-36 | 7 | 1.736 -- 1.136 |
| TSTD,LT | 2D.05 | 6-36 | 7 | 1.736 -- 1.136 |
| TSTDF,GE | 2D.08 | 6-37 | 7 | 1.736 -- 3.64 |
| TSTDF,LE | 2D.09 | 6-37 | 7 | 1.736 -- 3.64 |
| TSTDF,EQ | 2D.0A | 6-37 | 7 | 1.568 -- 2.744 |
| TSTDF,NE | 2D.0B | 6-37 | 7 | 1.848 -- 3.08 |
| TSTDF,GT | 2D.0C | 6-38 | 7 | 1.736 -- 3.64 |
| TSTDF,LT | 2D.0D | 6-38 | 7 | 1.736 -- 3.64 |
| BSK | 2E.10 | 6-41 | 8 | 1.456 -- 2.688 |
| BSK,S | 2E.11 | 6-41 | 8 | 1.456 -- 2.688 |
| BSK,C | 2E.12 | 6-42 | 8 | 1.456 -- 2.688 |
| BSK,T | 2E.13 | 6-42 | 8 | 1.456 -- 2.688 |
| BSK,Z | 2E.14 | 6-42 | 8 | 1.456 -- 2.688 |
| BSK,ZS | 2E.15 | 6-42 | 8 | 1.456 -- 2.688 |
| BSK,ZC | 2E.16 | 6-43 | 8 | 1.456 -- 2.688 |
| BSK,ZT | 2E.17 | 6-43 | 8 | 1.456 -- 2.688 |
| FSK,GE | 2F.00 | 6-44 | 11 | 1.4 -- 1.96 |
| FSK,LE | 2F.01 | 6-44 | 11 | 1.4 -- 1.96 |
| FSK,EQ | 2F.02 | 6-45 | 11 | 1.4 -- 1.96 |
| FSK,NE | 2F.03 | 6-45 | 11 | 1.4 -- 1.96 |
| FSK,GT | 2F.04 | 6-45 | 11 | 1.4 -- 1.96 |
| FSK,LT | 2F.05 | 6-45 | 11 | 1.4 -- 1.96 |
| UJP | 2F.06 | 6-46 | 10 | 1.456 |
| UJI | 2F.07 | 6-47 | 10 | 2.52 |

INSTRUCTION CROSS REFERENCE (Contd)

| <u>Op Code</u> | <u>Page</u> | <u>Format</u> | <u>Time</u> |
|----------------|-------------|---------------|--------------------|
| RTJ | 2F.08 | 6-47 | 10 1.512 |
| JSX | 2F.09 | 6-47 | 10 1.456 |
| BJPT | 2F.OA | 6-48 | 10 1.288 -- #1.792 |
| BJPF | 2F.OB | 6-48 | 10 1.288 -- #1.792 |
| HLT | 2F.OC | 6-49 | 10 IDLES CPU |
| XJP | 2F.OD | 6-49 | 10 1.232 -- 1.512 |
| MON | 2F.OE | 6-49 | 11 8.624 |
| XSK | 2F.OF | 6-50 | 10 1.624 -- 2.184 |
| STM | 2F.10 | 6-18 | 7 2.184 + .616(N) |
| LDM | 2F.11 | 6-13 | 7 2.352 + .84(N) |
| AIF | 2F.12 | 6-47 | 10 |
| AIFD | 2F.13 | 6-48 | 10 |
| ED01 | 2F.14 | 6-51 | 10 |
| ED02 | 2F.15 | 6-51 | 10 |
| ED03 | 2F.16 | 6-52 | 10 |
| ILLEGAL | 2F.17 | | ILLEGAL SUB-OP |
| ILLEGAL | 2F.18 | | ILLEGAL SUB-OP |
| ILLEGAL | 2F.19 | | ILLEGAL SUB-OP |
| ILLEGAL | 2F.1A | | ILLEGAL SUB-OP |
| ILLEGAL | 2F.1B | | ILLEGAL SUB-OP |
| ILLEGAL | 2F.1C | | ILLEGAL SUB-OP |
| ILLEGAL | 2F.1D | | ILLEGAL SUB-OP |
| DXJP | 2F.1E | 6-53 | 10 |
| LCPN | 2F.1F | 6-53 | 11 2.52 |
| R,+ | 30.00 | 6-54 | 9 1.008 |
| R,- | 30.01 | 6-55 | 9 1.008 |
| R,* | 30.02 | 6-55 | 9 |
| R,/ | 30.03 | 6-55 | 9 |
| R,AND | 30.04 | 6-55 | 9 0.952 |
| R,OR | 30.05 | 6-56 | 9 0.952 |
| R,XOR | 30.06 | 6-56 | 9 0.952 |
| R,SCL | 30.07 | 6-56 | 9 0.952 |

INSTRUCTION CROSS REFERENCE (Contd)

| <u>Op Code</u> | | <u>Page</u> | <u>Format</u> | <u>Time</u> |
|----------------|-------|-------------|---------------|----------------|
| R,NOT | 30.08 | 6-56 | 9 | 1.12 |
| R,XFR | 30.09 | 6-57 | 9 | 1.12 |
| NBR | 30.0A | 6-58 | 9 | 1.008 |
| SBR | 30.0B | 6-58 | 9 | 1.008 |
| CBR | 30.0C | 6-58 | 9 | 1.008 |
| R,S* | 30.0D | 6-57 | 9 | |
| R,S/ | 30.0E | 6-57 | 9 | |
| RMS | 30.0F | 6-58 | 9 | 1.456 |
| ILLEGAL | 30.10 | | | |
| ILLEGAL | 30.11 | | | |
| ILLEGAL | 30.12 | | | ILLEGAL SUB-OP |
| ILLEGAL | 30.13 | | | ILLEGAL SUB-OP |
| RF,+ | 30.14 | 6-59 | 9 | |
| RF,- | 30.15 | 6-59 | 9 | |
| RF,* | 30.16 | 6-59 | 9 | |
| RF,/ | 30.17 | 6-59 | 9 | |
| RFD,+ | 30.18 | 6-60 | 9 | |
| RFD,- | 30.19 | 6-60 | 9 | |
| RFD,* | 30.1A | 6-60 | 9 | |
| RFD,/ | 30.1B | 6-60 | 9 | |
| RD,+ | 30.1C | 6-61 | 9 | 2.128 |
| RD,- | 30.1D | 6-61 | 9 | 2.128 |
| RD,XFR | 30.1E | 6-61 | 9 | 2.296 |
| RJD | 30.1F | 6-62 | 9 | |
| AAL | 31.00 | 6-63 | 9 | |
| ILLEGAL | 31.01 | | 9 | ILLEGAL SUB-OP |
| ILLEGAL | 31.02 | | 9 | ILLEGAL SUB-OP |
| F,F | 31.03 | 6-63 | 9 | |
| F,UF | 31.04 | 6-64 | 9 | |
| FD,F | 31.05 | 6-64 | 9 | 1.736 -- 6.888 |
| FD,UF | 31.06 | 6-64 | 9 | 2.296 -- 7.504 |
| F,STD | 31.07 | 6-65 | 9 | 1.456 |

INSTRUCTION CROSS REFERENCE (Contd)

| <u>Op Code</u> | | <u>Page</u> | <u>Format</u> | <u>Time</u> |
|----------------|-------|-------------|---------------|----------------------|
| F,DTS | 31.08 | 6-65 | 9 | 1.624 |
| F,AB | 31.09 | 6-65 | 9 | 1.456 -- 1.512 |
| F,ABD | 31.0A | 6-65 | 9 | 1.68 -- 2.016 |
| F,ABI | 31.0B | 6-66 | 9 | 1.176 -- 1.232 |
| DLD | 32 | 6-13 | 5 | 1.736 |
| LDP | 33 | 6-13 | 5 | 1.736 |
| STP | 34 | 6-18 | 5 | 1.568 |
| MPS | 35 | 6-22 | 5 | 1.736 - 8.68 - 1.576 |
| DVS | 36 | 6-22 | 5 | 11.592 -- 11.7 |
| TBIT | 37.00 | 6-39 | 8 | 1.456 |
| SBIT | 37.01 | 6-39 | 8 | 1.456 |
| CBIT | 37.02 | 6-40 | 8 | 1.456 |
| LDF | 37.03 | 6-14 | 9 | |
| STF | 37.04 | 6-18 | 9 | |
| AABL | 37.05 | 6-93 | 9 | |
| AABR | 37.06 | 6-93 | 9 | |
| ILLEGAL | 37.07 | | | |
| ILLEGAL | 37.08 | | | |
| ILLEGAL | 37.09 | | | |
| MOVE | 37.10 | 6-67 | 9 | 2.016 + .952(N) |
| MOVC | 37.11 | 6-68 | 9 | 2.016 + .952(N) |
| FILL | 37.12 | 6-70 | 9 | |
| MOVT | 37.13 | 6-70 | 9 | |
| MOVA | 37.14 | 6-72 | 9 | |
| MOVU | 37.15 | 6-73 | 9 | |
| MOV P | 37.16 | 6-74 | 9 | |
| MOV N | 37.17 | 6-76 | 9 | |
| ILLEGAL | 37.18 | | | ILLEGAL SUB-OP |
| ILLEGAL | 37.19 | | | ILLEGAL SUB-OP |
| ILLEGAL | 37.1A | | | ILLEGAL SUB-OP |
| ILLEGAL | 37.1B | | | ILLEGAL SUB-OP |
| ILLEGAL | 37.1C | | | ILLEGAL SUB-OP |

INSTRUCTION CROSS REFERENCE (Contd)

| <u>Op Code</u> | | <u>Page</u> | <u>Format</u> | <u>Time</u> |
|----------------|-------|-------------|---------------|----------------------|
| PTHD | 37.1D | 6-94 | 5 | 2.464 -- 1.568(N) |
| UTHD | 37.1E | 6-95 | 5 | 2.464 -- 4.032 |
| ILLEGAL | 38 | | | ILLEGAL OPEN OP CODE |
| ILLEGAL | 39 | | | ILLEGAL OPEN OP CODE |
| IN | 3A.00 | 6-78 | 9 | CONTROLLER DEPENDENT |
| OUT | 3A.01 | 6-78 | 9 | CONTROLLER DEPENDENT |
| NIO | 3A.02 | 6-79 | 9 | 2.856 |
| SPS | 3A.03 | 6-79 | 9 | 2.688 |
| SMIO | 3A.04 | 6-79 | 9 | 1.96 |
| RMIO | 3A.05 | 6-80 | 9 | 2.128 |
| WPF | 3A.06 | 6-80 | 9 | 2.184 |
| RPF | 3A.07 | 6-80 | 9 | 1.792 |
| SPF | 3A.08 | 6-80 | 9 | 22.456 |
| WSR | 3A.09 | 6-81 | 9 | 1.456 |
| RSR | 3A.0A | 6-81 | 9 | 1.456 |
| SSRM | 3A.0B | 6-82 | 11 | 1.512 |
| SCRM | 3A.0C | 6-82 | 11 | 1.512 |
| RIM | 3A.0D | 6-82 | 11 | 1.512 |
| RRM | 3A.0E | 6-82 | 11 | 1.512 |
| ILLEGAL | 3A.0F | | | ILLEGAL SUB-OP |
| SJD | 3A.10 | 6-83 | 11 | |
| ILLEGAL | 3B | | | ILLEGAL OPEN OP CODE |
| ILLEGAL | 3C | | | ILLEGAL OPEN OP CODE |
| ILLEGAL | 3D | | | ILLEGAL OPEN OP CODE |
| ILLEGAL | 3E | | | ILLEGAL OPEN OP CODE |
| CONT | 3F.03 | 6-84 | 11 | 6.496 -- 6.944 |
| LXPA | 3F.04 | 6-85 | 11 | 2.24 |
| EINT | 3F.05 | 6-85 | 11 | 1.512 |

INSTRUCTION CROSS REFERENCE (Contd)

| <u>Op Code</u> | <u>Page</u> | <u>Format</u> | <u>Time</u> |
|----------------|-------------|---------------|------------------------|
| SPG | 3F.06 | 6-86 | 9 3.808 |
| RPG | 3F.07 | 6-86 | 9 3.404 |
| SSIM | 3F.08 | 6-86 | 11 1.512 |
| SCIM | 3F.09 | 6-86 | 11 1.512 |
| ILLEGAL | 3F.0A | | ILLEGAL SUB-OP |
| ILLEGAL | 3F.0B | | ILLEGAL SUB-OP |
| SRTC | 3F.0C | 6-87 | 11 1.456 |
| SIT | 3F.0D | 6-87 | 11 1.456 |
| TRC | 3F.0E | 6-87 | 11 1.456 |
| DINT | 3F.0F | 6-87 | 11 1.68 |
| SST | 3F.10 | 6-88 | 11 2.128 |
| SOPR | 3F.11 | 6-88 | 9 21.784 |
| LOPR | 3F.12 | 6-88 | 9 25.368 |
| DST | 3F.13 | 6-88 | 11 X |
| LPIR | 3F.14 | 6-89 | 9 23.296 |
| LMM | 3F.15 | 6-89 | 11 X |
| ILLEGAL | 3F.16 | | ILLEGAL SUB-OP |
| ILLEGAL | 3F.17 | | ILLEGAL SUB-OP |
| ILLEGAL | 3F.18 | | ILLEGAL SUB-OP |
| PAUS | 3F.19 | 6-90 | 11 VARIABLE TIME DELAY |
| SCPN | 3F.1A | 6-90 | 11 2.296 |
| ILLEGAL | 3F.1B | | ILLEGAL SUB-OP |
| ILLEGAL | 3F.1C | | ILLEGAL SUB-OP |
| OST | 3F.1D | 6-90 | 5 2.464 -- 1.568(N) |
| OSU | 3F.1E | 6-92 | 5 2.464 -- 4.032 |
| ILLEGAL | 3F.1F | | ILLEGAL SUB-OP |

COMMENT SHEET

TITLE: MP-60 Emulation Reference Manual

PUBLICATION NUMBER: 17329120

REVISION: C

NAME:

COMPANY:

STREET ADDRESS:

CITY:

STATE:

ZIP CODE:

Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND TAPE

FOLD

FOLD



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Systems Technology Division

215 Moffett Park Drive
Sunnyvale, California 94088



CUT ALONG LINE

FOLD

FOLD

| | <u>MIN</u> | <u>MAX</u> | <u>TYP</u> | |
|-------------------|------------|------------|------------|---------------------|
| UJP | | | 1.50 | |
| UJI | | | 2.41 | |
| RTJ | 1.87 | 2.24 | 2.06 | |
| JSX | | | 1.50 | |
| XJP | 1.18 | 1.81 | 1.81 | |
| TST,GE/GT/LE/LT | 1.35 | 2.86 | 1.35 | (JUMP is exception) |
| TST,EQ/NE | 1.12 | 2.80 | 1.35 | (JUMP is exception) |
| TSTD,GE/GT/LE/LT | 1.91 | 3.53 | 2.24 | (JUMP is exception) |
| TSTD,EQ/NE | 1.68 | 3.20 | 1.91 | (JUMP is exception) |
| TSTF,GE/GT/LE/LT | 1.35 | 3.60 | 1.35 | (JUMP is exception) |
| TSTF,EQ/NE | 1.12 | 2.80 | 1.35 | (JUMP is exception) |
| TSTDF,GE/GT/LE/LT | 1.91 | 4.70 | 1.91 | (JUMP is exception) |
| TSTDF,EQ/NE | 1.68 | 3.20 | 1.91 | (JUMP is exception) |
| BSK | 1.46 | 2.69 | 1.46 | (JUMP is exception) |
| BJPT | 1.29 | 2.19 | 1.29 | (JUMP is exception) |
| BJPF | 1.29 | 2.19 | 1.29 | (JUMP is exception) |
| FSK,GE/GT/LE/LT | 1.40 | 2.36 | 1.88 | |
| FSK,EQ/NE | 1.35 | 2.30 | 1.83 | |
| XSK | 1.57 | 2.52 | 1.57 | (SKIP is exception) |
| FAD/FSB | 5.30 | 7.20 | 5.50 | |
| RF,+/RF,- | 4.80 | 6.90 | 5.60 | |
| FMP | 9.35 | 11.10 | 9.95 | |
| RF,* | 9.20 | 10.95 | 8.80 | |
| FDV | 12.25 | 12.95 | 12.45 | |
| RF,/ | 11.95 | 12.65 | 12.15 | |
| FADD/FSBD | 6.45 | 11.65 | 7.90 | |
| RFD,+/RFD,- | 6.45 | 11.65 | 7.90 | |
| FXPD/RFD,* | 17.45 | 38.00 | 29.75 | |
| FDVD/RFD,/ | 19.35 | 47.10 | 36.75 | |
| F, IABS | 1.18 | 1.24 | 1.21 | |
| F,SQ | | | 24.36 | |

BSC/MP60 INSTRUCTION TIMES

| | <u>MIN</u> | <u>MAX</u> | <u>TYP</u> |
|----------------|------------|------------|---------------------------|
| NOP | | | .75 |
| LDBA | | | 1.01 |
| LDCA/LDHA | | | 1.24 |
| LDA | | | .90 |
| STBA/STCA/STHA | 2.90 | 2.35 | 2.10 |
| STH | | | 1.58 |
| MPI | 4.00 | 5.00 | 4.50 |
| LDI | .90 | 1.07 | .90 |
| LP/LOR/LXR | | | 1.01 |
| ADI | 1.12 | 1.29 | 1.14 |
| SF - LEFT | 1.52 | 3.25 | 1.52 + (.056)n 0 ≤ n ≤ 31 |
| SF - RIGHT | 1.68 | 3.64 | 1.91 + (.056)n 0 ≤ n ≤ 31 |
| SFD - LEFT | 2.02 | 5.55 | 2.02 + (.056)n 0 ≤ n ≤ 63 |
| SFD - RIGHT | 2.35 | 5.94 | 2.41 + (.056)n 0 ≤ n ≤ 63 |
| LDB | 2.50 | 3.10 | 2.85 |
| STB | 2.50 | 3.20 | 3.00 |
| AND/OR | 1.01 | 3.15 | 2.20 |
| XOR | 2.50 | 3.15 | 2.90 |
| NBR/SBR/CBR | | | 1.01 |
| LDC/LDH | | | 1.58 |
| STC/STH | | | 1.24 |
| LD | | | 1.40 |
| ST | | | 1.10 |
| AD/SB | | | 1.45 |
| RAD | | | 2.10 |
| MP | 7.50 | 9.30 | 8.10 |
| DV | 12.05 | 12.72 | 12.40 |
| MPS | 7.28 | 9.02 | 7.90 |
| DVS | 11.54 | 11.70 | 11.60 |
| LDD | | | 1.90 |
| STD | | | 1.80 |
| ADD/SBD | | | 2.30 |
| R, logical | | | .96 |
| R,+/- | | | 1.01 |
| R,XFR/NOT | | | 1.13 |
| RD,+/- | | | 2.14 |
| RD,XFR | | | 2.30 |

152