Publication Number CB29-633R02

METRIC

MODEL 8/16E PROCESSOR

USER'S MANUAL

PERKIN-ELMER

Computer Systems Division
2 Crescent Place
Oceanport, N.J. 07757

 July 1980

PAGE REVISION STATUS SHEET

PUBLICATION NUMBER CB29-633

TITLE Model 8/16E Processor User's Manual

REVISION R02 DATE July 1980

PAGE	REV.	DATE
i/ii	R00	1/78
iii	R00	1/78
vi	R01	10/78
v/vi	R01	10/78
1-1	R01	10/78
1-2 thru 1-3/ 1-4	R00	1/78
2-1 thru 2-9/ 2-10	R00	1/78
3-1 thru 3-31/ 3-32	R00	1/78
4-1 thru 4-23/ 4-24	R00	1/78
5-1 thru 5-16	R00	1/78
6-1 thru 6-36	R00	1/78
7-1 thru 7-7/ 7-8	R00	1/78
8-1	R00	1/78
8-2	R00	1/78
8-3	R00	1/78
8-4	R02	7/80
8-5 thru 8-11/ 8-12	R00	1/78
9-1 thru 9-5/ 9-6	R02	7/80
9-7 thru 9-25/ 9-26	R00	1/78
10-1 thru 10-23/ 10-24	R00	1/78
11-1 thru 11-11/ 11-12	R00	1/78
A1-1/ A1-2	R01	10/78
A2-1 thru A2-3/ A2-4	R00	1/78
A3-1 thru A3-3/ A3-4	R00	1/78
A4-1	R00	1/78
A4-2	R00	1/78
A5-1 thru A5-6	R00	1/78
A6-1	R00	1/78
A6-2	R00	1/78
A7-1 thru A7-4	R00	1/78
A8-1 thru A8-5/ A8-6	R00	1/78
Index-1	R02	7/80
Index-2	R00	1/78
Index-3	R02	7/80
Index-4	R02	7/80
Index-5	R02	7/80
Index-6	R02	7/80

A2254

PREFACE

The Model 8/16E User's Manual provides information for the programmer and operater in the use of the Model 8/16E computer system. The Programmer is given detailed information on the 16-bit system architecture and the unique memory management scheme as well as a description of each instruction in the Model 8/16E repertoire. The instruction descriptions include valuable system related information presented in the form of programming notes and instruction examples. Information on the Hexadecimal display panel is given to facilitate program preparation and execution for the programmer and operator of the system.

Table of Contents

APPENDICES

FIGURES

FIGURES (Continued)

TABLES

CHAPTER 1
INTRODUCTION

The Perkin-Elmer Model 8/16E Processor allows addressing of more than 64KB of main memory. As much as 256KB of main memory may be added to the Model 8/16E. Included in the basic configuration are the 8/16E Processor with 98 separate instructions (optionally expandable). 8K bytes of memory, 16 general-purpose registers, hardware interrupt vectoring for up to 255 external devices, high-speed Direct Memory Access (DMA) channel, and a power supply. A variety of standard, off-the-shelf options permit the user to tailor the system to meet both present needs and future requirements. The overall efficiency of the Perkin-Elmer 8/16E makes it well-suited for a wide variety of applications from small, dedicated processors to large multi-user systems.

System Architecture

The Perkin-Elmer Model 8/16E Processor is designed around the powerful third generation architecture. The advantages inherent in this type of architecture greatly simplify system design, programming, and debugging. The large, task-oriented instruction set allows the programmer to concentrate on system programming instead of difficult coding to accomplish such basic functions as EXCLUSIVE OR, multiple shifts, or byte processing.

The multi-accumulator architecture provides 16 general-purpose registers for increased programming flexibility, and eliminates accumulator housekeeping that is characteristic of machines with fewer general registers. All 16 registers are available for use at the programmer's discretion. None of the 16 registers is dedicated to any specific purpose, such as index registers, stack pointers, program counter, or subroutine return pointers. The programmer may use the 16 registers for storage of partial results, frequently used constants, loop management constants, etc.

Of the 16 general-purpose registers, 15 are available for indexing. Register Zero (R0) cannot be used as an index register as a zero placed in the index register select field implies no indexing is to be performed. The architectural design also provides 64KB of 100% directly addressable memory, eliminating time-consuming design problems associated with paging and indirect addressing. Programmers can write simple, in-line code without having to be concerned with running out of base pages, and without having to waste memory with indirect address references. A simple address translation scheme allows the 64KB of memory the program can access at any one time to reside in any two 32KB segments of actual available memory.

Memory

The main memory is built around core modules available in 32KB and 64KB versions. All three modules are available with parity as an option for those critical applications requiring the increased functional reliability and data integrity that parity provides. Each of the modules is contained on a single 15-inch printed circuit board, and occupies a single subassembly slot. The 32KB and 64KB core memories are available with a 750 nanosecond cycle time. Memory can be expanded by plugging in additional modules, to a maximum of 256KB (total).

Instruction Set

The instruction set is built around a basic set of 98 individual instructions, designed to provide the programmer with the tools he needs to write programs in as few steps as possible. These are 98 separate and individual instructions. All Perkin-Elmer indexing and masking instructions are counted exclusive of the mask value or indexing register field. To add extra meaning to the condition codes, Perkin-Elmer has provided 14 extended branch mnemonics which are interpreted by the assembler. This brings the total number of discrete mnemonics available to the programmer in the basic set to 112.

The basic instruction set uses both 16 and 32-bit instruction formats. They permit operations between any two general registers, between a general register and memory, or between a general register and a four bit data constant contained in the instruction word.

The Perkin-Elmer 16-bit Processor includes a complete set of Arithmetic and Logical instructions. A complete set of Conditional Branch instructions permits branching to any location in memory. A full set of byte-processing instructions simplifies the handling of byte strings and provides for more efficient and effective use of available memory. The Input/Output instructions permit operations between peripheral devices and general registers, or between peripheral devices and memory.

Input/Output System

The Perkin-Elmer input/output system for 16-bit Processors is built around a dual bus structure capable of handling 255 peripheral devices. High speed devices can operate up to 2,000,000 bytes per second over the optional Selector Channel (DMA). Medium and low speed devices are connected to the standard multiplexor channel that can operate up to 66,000 bytes per second. Both channels operate on a request-response basis to allow simple, reliable peripheral device controller design.

Under program control, automatic hardware interrupt vectoring is provided via the Interrupt Service Pointer (ISP) Table in memory. This allows for a separate driver in memory to be available for each device in the system Perkin-Elmer offers a broad line of competitively priced peripherals that are both program and interface compatible with all members of the Perkin-Elmer family. Perkin-Elmer also offers standard low-cost interface modules to aid the user in interface design.

Software

Standard software available for Perkin-Elmer 16-bit Processors includes: a symbolic Assembler, an interactive Text Editor, an interactive Debug Package, extended FORTRAN IV, FORTRAN V Level I, interactive BASIC, utility programs, and the following operating systems:

Basic Operating System (BOSS-PLUS)
OS/16 Multi-Task Operating System (OS/16 MT2)

In addition, the Perkin-Elmer user's group, INTERCHANGE, has a large software library of its own that is available to the 16-bit Processor user.

Processor Options and Peripherals

The 16-bit Processor provides a flexible hardware system that can be expanded to meet the end user's requirements quickly and easily. As system demands and complexity increase, the Processor can be field-expanded to provide the precise computational capability required.

Processor Options:

Memory Parity provides complete data and instruction protection.

Power Fail Detection/Auto Restart provides an early power fail interrupt and a power up interrupt.

Programmable Memory Protect provides read, write, or execution protection of 2K-byte blocks of memory under software control. The module generates an interrupt to the Processor when it detects a memory protection violation.

Binary Display Panel provides complete user control of the system. It includes long-life Light Emitting Diodes (LED), binary read out and a hexadecimal input keyboard.

Hexadecimal Display Panel provides hexadecimal LED read out in addition to the features of the Binary Display Panel.

Display Controller provides an interface for the Binary and Hexadecimal Display Panels.

Automatic Loader (ALO) provides a simple, single switch bootstrap load capability. ability.

Turnkey Console provides switch control for power, initializing, and execution for the Processor in dedicated systems.

Signed Multiply/Divide option minimizes execution time of mathematical routines, and eliminates the necessity for additional code to generate properly signed quotients and products.

The High Speed floating point option provides both single precision and double precision floating point operations.

Peripheral Products include:

- Selector Channel (SELCH)
- Teletype (ASR 33, 35)
- Carousel (Model 15, 30, 35, 300)
- CRT (Non-Editing, Editing, and Graphic)
- Intertape Cassette System
- Digital Multiplexor System
- Mini Input/Output System (D/A and A/D)
- Real Time Analog System
- Clock Modules (Line Frequency and Precision Interval)
- Universal Logic Interfaces
- I/O Bus Switch
- Line Printers (60, 200, 600 LPM)
- Card Readers (400, 1000 CPM)
- Paper Tape Reader/Punch
- Industry Compatible Magnetic Tapes
  - 9 Track, 45IPS, 800BPI
  - 9 Track, 45IPS, 1600BPI
  - 7 Track, 45IPS, 200BPI
  - 7 Track, 45IPS, 556BPI
  - 7 Track, 45IPS, 800BPI
  - Floppy Media Disc System
  - Disc Systems (2.5, 10, 40, 67, 256 MB)
  - Synchronous Data Set Interfaces (201, 301)
  - Quad Synchronous Adapter (BISYNC or ZBID)
  - Asynchronous Data Set Interfaces (103, 202)
  - IBM 360/370 Interfaces

CHAPTER 2
SYSTEM DESCRIPTION

The unique design characteristics of the Perkin-Elmer 16-bit Processors allow for a fully integrated system in which the relationships between Processor and memory, memory and peripherals, and peripherals and Processor are precisely balanced to provide the utmost in hardware reliability, software simplicity, and total system throughput.

Figure 2-1 illustrates how the various elements of a 16-bit Processor system are combined.

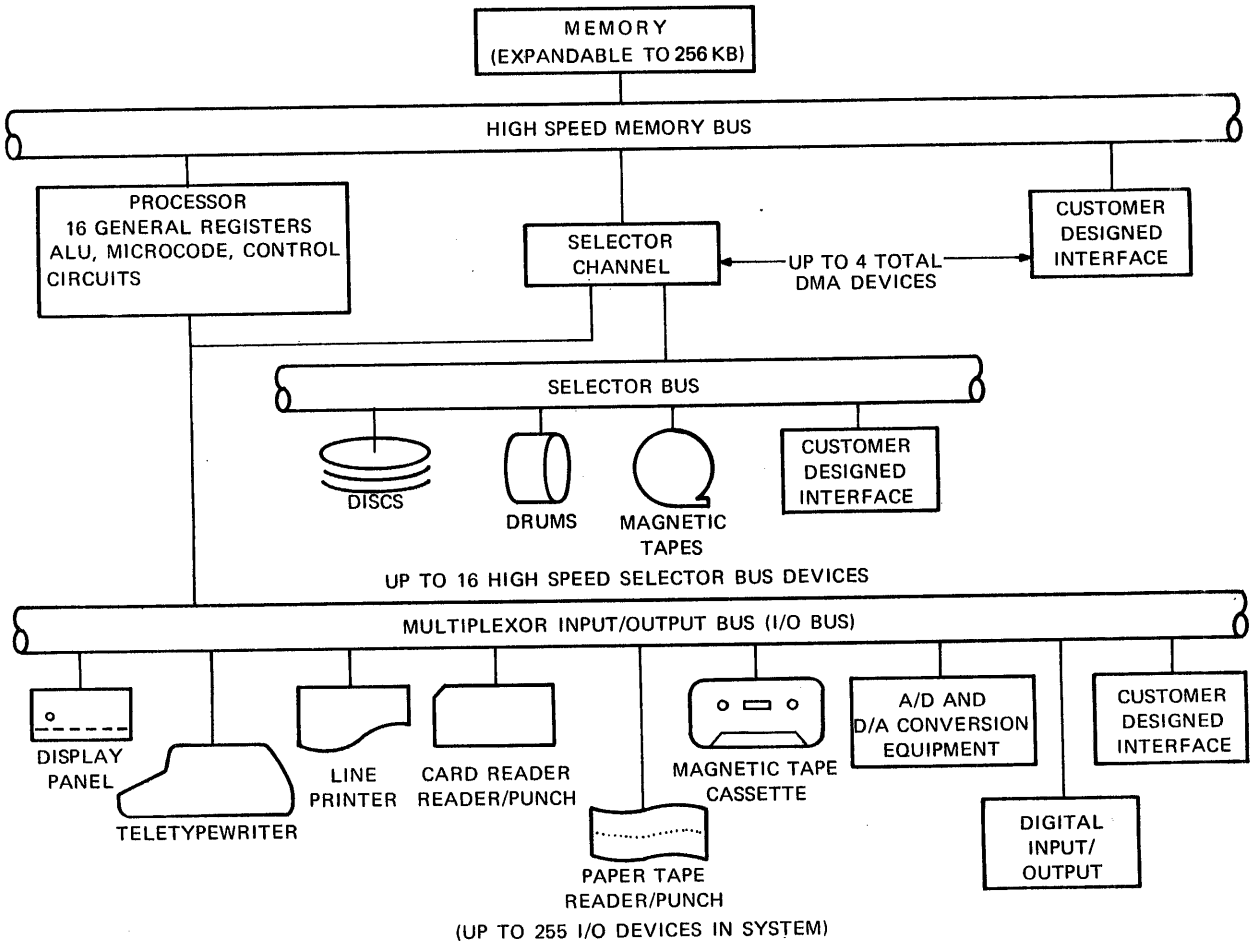


Figure 2-1. System Block Diagram

PROCESSOR

The following is a general overview of the Model 8/16E Processors.

The Central Processing Unit (CPU), or Processor, controls activities in the system. It executes instructions in a specific sequence, and performs arithmetic and logical functions. Included in the Processor's components are:

Program Status Word Register
16 General Registers
Signed multiply/divide hardware (optional)
Floating Point Hardware (optional)
- 8 Single Precision Floating Point Registers
- 8 Double Precision Floating Point Registers

Program Status Word

The 32-bit Program Status Word (PSW), shown in Figure 2-2, defines the state of the Processor at any given time.

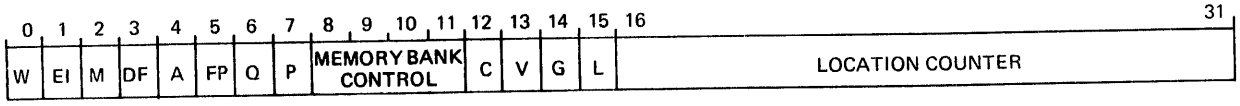


Figure 2-2. Program Status Word Format

Bits 0:15 are reserved for status information and for in interrupt masks. Bits 16:31 contain the Location Counter. Status information and interrupt mask bits are defined as follows:

Mask Bit	Status Information
Bit 0	Wait state (W)
Bit 1	External interrupt mask (EI)
Bit 2	Machine malfunction interrupt mask (M)
Bit 3	Fixed point divide fault interrupt mask (DF)
Bit 4	Automatic I/O and immediate interrupt mask (A)
Bit 5	Floating point fault interrupt mask (FP)
Bit 6	System queue service interrupt mask (Q)
Bit 7	Protect mode (P)
Bits 8:11	Memory Bank Control
Bits 12:15	Condition Code (C,V,G, & L)

Wait State (W)

When Bit 0 of the Program Status Word is set, the Processor halts normal program execution and assumes an idle state. It is still responsive to machine malfunction, external, and immediate interrupts, and to automatic I/O, if these are enabled by other PSW bits.

External Interrupt Mask (EI)

Bit 1 of the Program Status Word controls requests for service from devices on the Multiplexor Bus, including the Selector Channel. If this bit is set, the Processor responds to the requests. If this bit is reset, the requests are queued. This bit also controls the Automatic I/O Operation.

Machine Malfunction Interrupt Mask (M)

Bit 2 of the Program Status Word controls interrupts generated when power fails, when power returns, and when parity checking indicates a memory parity error.

Fixed Point Divide Fault Interrupt Mask (DF)

Bit 3 of the Program Status Word controls interrupts generated when a fixed point divide operation results in quotient overflow, or when division by zero is attempted. If this bit is set, the interrupt is taken. If this bit is reset, the interrupt condition is ignored.

Automatic I/O and Immediate Interrupt Mask (A)

Bit 4 of the Program Status Word controls automatic I/O operations and the vectored immediate interrupt. If this bit is set, along with Bit 1, these functions are enabled (see Chapter 9), and the normal PSW swap from location X'40' is disabled.

Floating Point Fault Interrupt Mask (FP)

Bit 5 of the Program Status Word controls interrupts generated on floating point overflow or underflow, or division by zero. If this bit is set, these conditions cause an interrupt. If this bit is reset, the interrupt condition is ignored (See Chapter 6).

System Queue Service Interrupt Mask (Q)

Bit 6 of the Program Status Word controls the operation of the system queue interrupt. If this bit is set, and if the queue requires service, the interrupt is taken. This bit is also used in connection with Automatic I/O (See Chapter 9).

Protect Mode (P)

Bit 7 of the Program Status Word controls the execution of privileged instructions. If this bit is reset, any legal instruction may be executed. If this bit is set, only non-privileged instructions may be executed. When set, attempts to execute privileged instructions cause an Illegal Instruction Interrupt.

Memory Bank Controller

Bits 8:11 of the Program Status Word control the translation from 16-bit Program Addresses to 18-bit Physical Addresses. A Program address is the 16-bit Location Counter for fetching instructions, or it is the 16-bit effective address of an operand. The 64KB range of program addresses is divided into two 32KB segments. Segment 0 represents program addresses X'0000' through X'7FFF'. Segment 1 represents program addresses X'8000' through X'FFFF'.

The 256KB range of physical address is divided into eight 32KB segments. Depending on the particular combination in PSW bits 8:11, program addresses in segment 0 canbe steered to physical segment 0 or 1; and program addresses in segment 1 can be steered to physical segment 0 through 7. Refer to Chapter 7 for details.

Condition Code (C, V, G, & L)

Bits 12:15 of the Program Status Word are the Condition Code bits. These bits are set by the Processor to indicate the results of instruction execution (See Chapter 4). The usual interpretation of these bits is:

Bit 12	C	Carry or borrow
Bit 13	V	Overflow
Bit 14	G	Greater than zero
Bit 15	L	Less than zero

Processor Interrupts

Interrupt conditions cause the entire Program Status Word to be replaced by a new Program Status Word, thus breaking the usual sequential flow of instruction execution. When an interrupt condition arises, the Processor saves its current Program Status Word in a memory location unique to the type of interrupt condition. It loads a new Program Status Word from a corresponding memory location and vectors to the address specified by the new PSW.

General Registers

There are 16 general purpose registers numbered 0 through 15. Each register is 16 bits wide. None of these registers has a preset use. All may be used at the programmer's discretion for accumulators and for the storing of temporary data. Registers 1 through 15 may be used as index registers. If register 0 is specified as an index register, no indexing occurs.

Single Precision Floating Point Registers

There are eight single precision floating point registers, each 32 bits wide. The registers are identified by the even numbers 0 through 14. Floating point operations must always specify the registers with even numbers. The results are undefined if odd numbers are used.

Double Precision Floating Point Registers

There are eight double precision floating point registers, each 64 bits wide. The registers are identified by the even numbers 0 through 14. Floating point operations must always specify the registers with even numbers. The results are undefined if odd numbers are used. Double precision floating point registers are entirely separate from the single precision floating point registers.

Reserved Memory Locations

The following memory locations are reserved for interrupt pointers, Program Status Words, and system constants.

Location			Use
X'0000'	–	X'001F'	Reserved (Single Precision Floating Point Register Save)
X'0020'	–	X'0021'	Reserved (Display Status)
X'0022'	–	X'0023'	Power Fail Register Save Pointer
X'0024'	–	X'0027'	Power Fail PSW Save Area
X'0028'	–	X'002B'	Floating Point Fault Interrupt Old PSW
X'002C'	–	X'002F'	Floating Point Fault Interrupt New PSW
X'0030'	–	X'0033'	Illegal Instruction Interrupt Old PSW
X'0034'	–	X'0037'	Illegal Instruction Interrupt New PSW
X'0038'	–	X'003B'	Machine Malfunction Interrupt Old PSW
X'003C'	–	X'003F'	Machine Malfunction Interrupt New PSW
X'0040'	–	X'0043'	External Interrupt Old PSW
X'0044'	–	X'0047'	External Interrupt New PSW
X'0048'	–	X'004B'	Fixed Point Divide Fault Interrupt Old PSW
X'004C'	–	X'004F'	Fixed Point Divide Fault Interrupt New PSW
X'0050'	–	X'007F'	Bootstrap Loader and Device Definition Table
X'0080'	–	X'0081'	System Queue Pointer
X'0082'	–	X'0085'	Automatic I/O Channel Termination Interrupt Old PSW
X'0086'	–	X'0089'	Automatic I/O Channel Termination Interrupt New PSW
X'008A'	–	X'008B'	System Queue Overflow Pointer
X'008C'	–	X'008F'	System Queue Overflow Interrupt Old PSW
X'0090'	–	X'0093'	System Queue Overflow Interrupt New PSW
X'0094'	–	X'0095'	Supervisor Call Argument Pointer
X'0096'	–	X'0099'	Supervisor Call Interrupt Old PSW
X'009A'	–	X'009B'	Supervisor Call Interrupt New Status (PSW)
X'009C'	–	X'00BB'	Supervisor Call Interrupt New Location Counters
X'00BC'	–	X'00CF'	Reserved (not used, must be zero)
X'00D0'	–	X'02CF'	Interrupt Service Pointer Table

These reserved locations play an important role in both interrupt and input/output processing. For a detailed description refer to Chapters 8 and 9.

Processor Operations

The Processor performs logical and fixed point arithmetic operations between:

The contents of two registers.
The contents of a register and the contents of a halfword located in memory.

Where the second operand is contained in memory, it may be located in the instruction stream (immediate operand), or it may be located in indexed storage.

Floating point operations take place between the contents of two floating point registers, or between the contents of a floating point register and a floating point operand contained in a fullword in memory.

DATA FORMATS

The Processor performs logical and arithmetic operations on 8-bit bytes, 16-bit halfwords, and 32-bit fullwords. This data may represent a fixed point number, a floating point number, or logical information.

Fixed Point Data

Fixed point arithmetic operands are 16-bit halfwords or 32-bit fullwords. In both of these formats, the most significant bit is the Sign bit, and the remaining bits represent the magnitude. Positive quantities are expressed in true binary form with a Sign bit of zero. Negative quantities are expressed in two's complement form with a Sign bit of one. The numerical value of zero is represented with all bits zero.

Floating Point Data

A floating point number consists of a signed exponent and a signed fraction. The quantity expressed by this notation is the product of the fraction and the number 16 raised to the power of the exponent. Each floating point value requires a 32-bit fullword or a 64-bit doubleword, of which 8 bits are used for the sign and the exponent, and the remaining bits are used for the fraction.

Logical Data

Logical operations manipulate 8-bit bytes, 16-bit halfwords, or 32-bit fullwords. All bits participate in logical operations. The Sign bit has no particular significance.

INSTRUCTION FORMATS

The Perkin-Elmer instruction formats provide a concise method of representing required operations for easy interpretation by the Processor. There are four basic formats, shown in Figure 2-5. The abbreviations used in the figure have the following meanings:

OP	Operation code
R1	First operand register
R2	Second operand register
N	A four bit immediate value
X2	Second operand single index register
A	Second operand 16-bit address
I	Second operand 16-bit immediate value

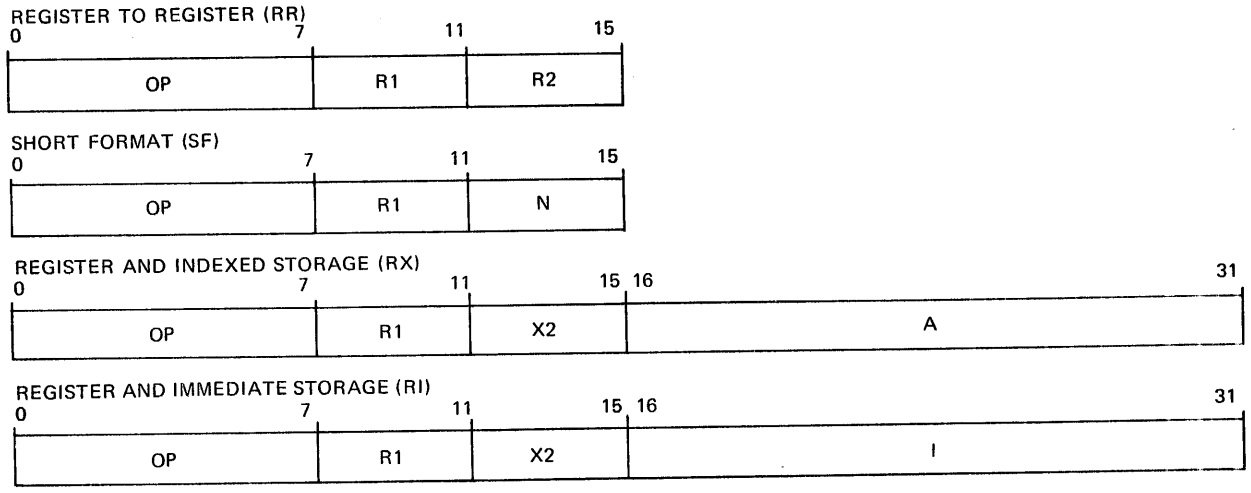


Figure 2-3. Instruction Formats

Many instructions may be expressed in two or more formats. This feature provides flexibility in data organization and instruction sequencing.

When working with the Perkin-Elmer Common Assembler Language (CAL) assembler, it is not necessary to specify the instruction format explicitly. The assembler chooses the most economical format and supplies the required bits in the machine code.

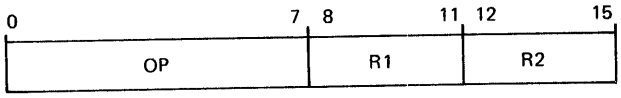
Branch Instruction Formats

The Branch instructions use the RR, SF, and the RX formats. However, in the Conditional Branch instructions, the R1 field does not specify a register. Instead, it contains a mask value (labeled M1 in the instruction descriptions), which is tested against the Condition Code. The Perkin-Elmer CAL assembler provides a series of Extended Branch Mnemonics which make it possible to specify a Conditional Branch without specifying the mask value explicitly. For a summary of the Extended Branch Mnemonics, see Appendix 4.

Programming Examples

Each of the following programming examples refers to the sample assembly language program shown in Figure 2-4. Note the use of symbolic equates for general registers. Machine code generated and the result of each instruction are dependent upon the physical and logical placement of the instructions, respectively.

Register to Register (RR) Format

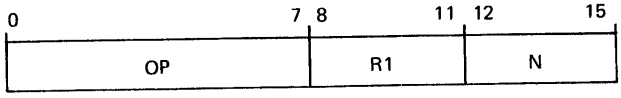


In this 16-bit format, Bits 0:7 contain the operation code. Bits 8:11 contain the R1 field, and Bits 12:15 contain the R2 field. In most RR instructions, the register specified by R1 contains the first operand, and the register specified by R2 contains the second operand. For example:

Machine Code	Label	Assembler Notation	
0865	RR	LHR	R6,R5

0865
- 5 — Second Operand
- 6 — First Operand
- 08 — 'LHR' Instruction Op-Code

Short Form (SF) Format

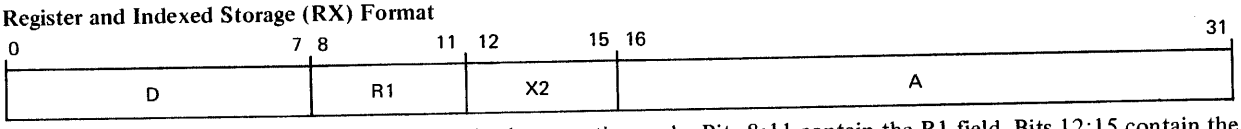


This 16-bit format provides space economy when working with small values. Bits 0:7 contain the operation code. Bits 8:11 contain the R1 field. Bits 12:15 contain the N field. In arithmetic and logical operations, the register specified by R1 contains the first operand. The N field contains a four bit immediate value (12:15) used as the second operand. For example:

Machine Code	Label	Assembler Notation	
245E	SF	LIS	R5,14

245E
- 5 — Second Operand
- 4 — First Operand
- 2E — 'LIS' Instruction Op-Code

Register and Indexed Storage (RX) Format



This is a 32-bit format in which Bits 0:7 contain the operation code, Bits 8:11 contain the R1 field, Bits 12:15 contain the X2 field, and Bits 16:31 contain the A field. In general, the register specified by R1 contains the first operand. The second operand is located in memory at the address obtained by adding the contents of the second operand index register, specified by X2, and the 16-bit absolute address contained in the A field. For example:

Machine Code	Label	Assembler Notation	
4050 1000	RX.EX1	STH	R5,X'1000'

4050 1000
- 1000 — Defines Second Operand Address
- 0 — No Index Register Specified
- 5 — First Operand
- 40 — 'STH' Instruction Op-Code

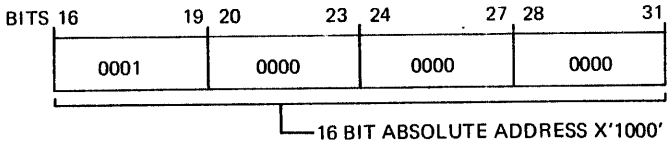
```
16-BIT INSTRUCTION FORMAT EXAMPLES                                  PAGE    1

PROG= EXAMPL      ASSEMBLED BY CAL 03-066R04(16-BIT)

                          1             SCRAT
                          2             TARGT   16
                          3             WIDTH   120
                          5  *
          0000 0005       6  R5         EQU     5                  GENERAL REGISTER 5
          0000 0006       7  R6         EQU     6                  GENERAL REGISTER 6
          0000 0009       8  R9         EQU     9                  GENERAL REGISTER 9
                          9  *
                         10  *
0000R     245E           11  SF         LIS     R5,14              (R5) = X'000E'
                         12  *
0002R     0865           13  RR         LHR     R6,R5              (R6) = X'000E'
                         14  *
0004R     4050 1000      15  RX.EX1     STH     R5,X'1000'         (X'1000') = X'000E'
                         16  *
0008R     4056 0FF2      17  RX.EX2     STH     R5,X'0FF2'(R6)     (X'1000') = X'000E'
                         18  *
000CR     2302           19             BS      RI.EX1
                         20  *
000ER     0000           21  LOC1       DC      H'0'
                         22  *
0010R     C890 8000      23  RI.EX1     LHI     R9,X'8000'         (R9) = X'8000'
                         24  *
0014R     C895 8000      25  RI.EX2     LHI     R9,X'8000'(R5)     (R9) = X'800E'
                         26  *
0018R     4300 0000R     27             B       SF
                         28  *
001CR                    29             END
```

Figure 2-4. 16-Bit Instruction Format Examples

The Second Operand address is calculated as follows:

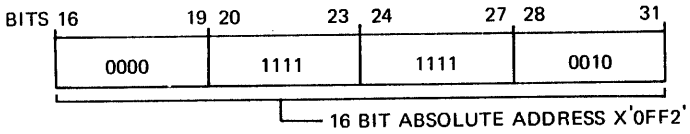


No indexing is specified. Therefore, the second operand address is X'1000'.

Machine Code	Label	Assembler Notation
4056 0FF2	RX.EX2	STH R5,X'0FF2'(R6)

- 0FF2 — Defines Second Operand Address
- 56 — Register 6 to be used for Indexing
- 4 — First Operand
- 40 — 'STH' Instruction Op-Code

The Second Operand address is calculated as follows:



Second Operand Address

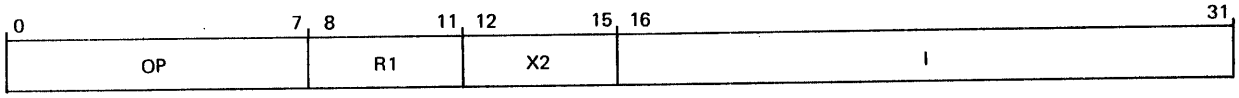
= contents of A field + contents of the Index Register 6 (see Figure 2-4)

= X'0FF2' + X'000E'

= X'1000'

The formation of the second operand address has absolutely no affect on the Condition Code. No flags are generated.

Register and Immediate Data (RI) Format



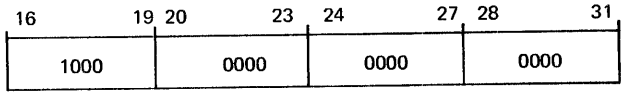
This format represents a 32-bit instruction word. Bits 0:7 contain the operation code. Bits 8:11 contain the R1 specification. Bits 12:15 contain the X2 specification. Bits 16:31 contain the 16-bit immediate value, I.

In this format, the register specified by R1 contains the first operand. The 16-bit effective second operand is obtained by adding together the 16-bit value contained in the I field, and the contents of the register specified by X2. For example:

Machine Code	Label	Assembler Notation
C890 8000	RI.EX1	LHI R9,X'8000'

- 8000 — 16-Bit Immediate Value
- 8 — No Index Register Specified
- 9 — First Operand
- C8 — 'LHI' Instruction Op-Code

The Second Operand is calculated as follows:



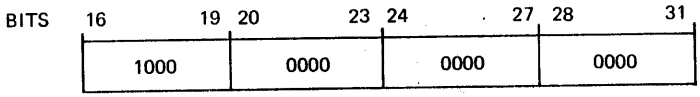
Second Operand

= X'8000'

Machine Code	Label	Assembler Notation
C895 8000	RI.EX2	LHI R9,X'8000'(R5)

- 8000 — 16-Bit Immediate Value
- 8 — Index Register 5 Specified
- 9 — First Operand
- C8 — 'LHI' Instruction Op-Code

The Second Operand is calculated as follows:



Second Operand

= X'8000' + the contents of Index Register 5 (See Figure 2-4).

= X'8000' + X'000E'

= X'800E'

The formation of the second operand address has absolutely no affect on the Condition Code. No flags are generated.

CHAPTER 3
LOGICAL OPERATIONS

This set of logical instructions provides a means for the manipulation of binary data. Many of the instructions grouped with the logical set may also be used in arithmetic and other operations. These instructions include loads, stores, compares, shifts, and list processing.

INTRODUCTION

The instruction repertoire has been grouped by function. The use and operation of each instruction is presented in the following format:

1. An instruction word chart for each instruction including: Mnemonic operation code, and first and second operand designations in the correct assembler format. The format type is designated by SF, RR, RI, or RX.

2. A description of instruction operation.

   An example of a diagrammatic representation of instruction operation is shown below.

   SIS: (R1) ← (R1) − N
   SHR: (R1) ← (R1) − (R2)
   SH: (R1) ← (R1) − [A + (X2)]
   SHI: (R1) ← (R1) − I + (X2)

3. A chart illustrating the possible variations of the Condition Code in the Current Program Status Word as a result of performing the instructions: a one indicates set, a zero indicates reset. It is important to note that any instruction which changes the Condition Code can change all four bits. The conditions listed on the chart are only those conditions which are meaningful after a particular instruction. Other bits may be changed, but their condition is not meaningful, for example:

   Resulting Condition Code:

12	13	14	15	
C	V	G	L	
0	0	0	0	DIFFERENCE IS ZERO
X	X	0	1	DIFFERENCE IS LESS THAN ZERO
X	X	1	0	DIFFERENCE IS GREATER THAN ZERO
X	1	X	X	ARITHMETIC OVERFLOW
1	X	X	X	BORROW

4. A programming note to provide additional pertinent or clarifying information. All privileged instructions and those instructions which may cause a memory protect violation are so noted.

5. Examples.

The symbols and abbreviations used in the instruction diagrams are defined as follows:

( )
[ ] Parentheses or Brackets. Read as "the content of . . .."

← Arrow. Read as "is replaced by . . ." or "replaces . . .."

A The 16-bit halfword address which is a part of the RX instructions.

I The 16-bit halfword immediate field of RI instructions.

R1 The address of a General Register containing the first operand.

M1 Mask of four bits specifying Branch on Condition testing.

R2 The address of a General Register containing the second operand of an RR instruction.

X2 The address of a General Register containing an index value.

N The 4-bit second operand used with Short Format Immediate and Short Format Branch instructions.

(0:7) A bit grouping within a byte, a halfword, or a fullword. Read as "0 through 7 inclusive."
(8:15) "Bits 8 through 15 inclusive", etc.
(16:31)

PSW Program Status Word of 32 bits containing the Status, Condition Code, and current instruction address.

CC Condition Code of 4-bits contained in the PSW.

C Carry Bit contained in the Condition Code (Bit 12 of PSW).

V Overflow Bit contained in the Condition Code (Bit 13 of PSW).

G Greater Than Bit contained in the Condition Code (Bit 14 of PSW).

L Less Than Bit contained in the Condition Code (Bit 15 of PSW).

+ Arithmetic operations — Add,
- Subtract,
* Multiply,
/ and Divide respectively.

: Logical comparison, (e.g., R1:R2).

DATA FORMATS

Logical data may be organized as bytes, halfwords, or fullwords as shown in Figure 3-1.

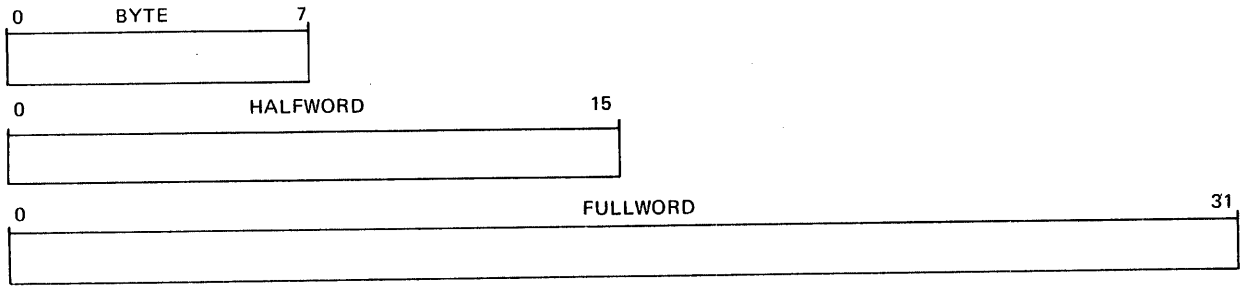


Figure 3-1. Logical Data

Boolean Operations

The Boolean operators AND, OR, and Exclusive OR (XOR) operate on halfword quantities. All bits in both operands participate individually. The Boolean functions are defined as follows:

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0 (logical product)
1 AND 1 = 1

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1 (logical sum)
1 OR 1 = 1

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1 (logical difference)
1 XOR 1 = 0

List Processing

The list processing instructions manipulate a circular list as defined in Figure 3-2.

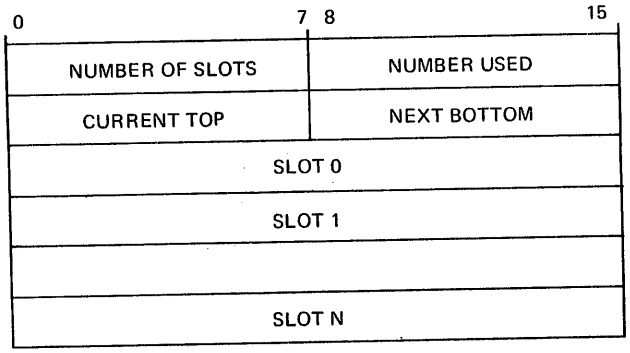


Figure 3-2. Circular List Definition

The first two halfwords contain the list parameters. Immediately following the parameter block is the list itself. The first halfword in the list is designated Slot 0. The remaining slots are designated 1, 2, 3, etc., up to a maximum slot number which is equal to the number in the list minus one. An absolute maximum of 255 halfword slots may be specified. (Slots are designated 0 through X'FE'.)

The first parameter byte indicates the number of slots (halfwords) in the entire list. The second parameter byte indicates the current number of slots being used. When this byte equals zero, the list is empty. When this byte equals the number of slots in the list, the list is full. Once initialized, this byte is maintained automatically. It is incremented when elements are added to the list and decremented when elements are removed.

The third and fourth bytes of the list parameter block specify the current top of the list and the next bottom of the list respectively. These pointers are also updated automatically. See Figure 3-3.

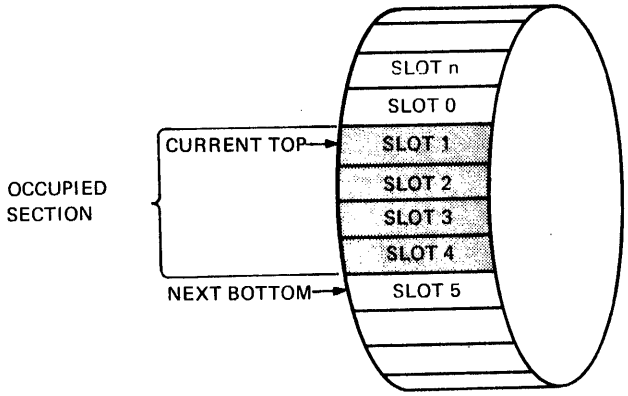


Figure 3-3. Circular List

LOGICAL INSTRUCTION FORMATS

The logical instructions use the Register to Register (RR), the Register and Indexed Storage (RX), and the Register and Immediate Storage (RI) instruction formats.

LOGICAL INSTRUCTIONS

The instructions described in this section are:

LIS	Load Immediate Short
LCS	Load Complement Short
LH	Load Halfword
LHI	Load Halfword Immediate
LHR	Load Halfword Register
LM	Load Multiple
LB	Load Byte
LBR	Load Byte Register
EXBR	Exchange Byte Register
STH	Store Halfword
STM	Store Multiple
STB	Store Byte
STBR	Store Byte Register
CLHR	Compare Logical Halfword Register
CLH	Compare Logical Halfword
CLHI	Compare Logical Halfword Immediate
CLB	Compare Logical Byte
NHR	AND Halfword Register
NH	AND Halfword
NHI	AND Halfword Immediate
OHR	OR Halfword Register
OH	OR Halfword
OHI	OR Halfword Immediate
XHR	Exclusive OR Halfword Register
XH	Exclusive OR Halfword
XHI	Exclusive OR Halfword Immediate
THI	Test Halfword Immediate
SLL	Shift Left Logical
SLLS	Shift Left Logical Short
SRL	Shift Right Logical
SRLS	Shift Right Logical Short
SLHL	Shift Left Halfword Logical
SRHL	Shift Right Halfword Logical
RLL	Rotate Left Logical
RRL	Rotate Right Logical
ATL	Add to Top of List
ABL	Add to Bottom of List
RTL	Remove from Top of List
RBL	Remove from Bottom of List

INSTRUCTIONS

Load Halfword Register (LHR)
Load Immediate Short (LIS)
Load Complement Short (LCS)

Assembler Notation		Op-Code	Format
LHR	R1,R2	08	RR
LIS	R1,N	24	SF
LCS	R1,N	25	SF

Operation

The second operand replaces the contents of the register specified in R1.

LHR: (R1) ← (R2)
LIS: (R1) ← N
LCS: (R1) ← -N

Condition Code

C	V	G	L	
0	0	0	0	Value is ZERO
0	0	0	1	Value is not ZERO
0	0	1	0	Value is not ZERO

Programming Note

These instructions may be used to preset a register with an index value, load a register with the first operand for a subsequent arithmetic operation (e.g., add, multiply), or set the Condition Code for supplemental testing by a Branch on Condition instruction.

The Load Immediate Short instruction causes the 4-bit second operand to be expanded to a 16-bit halfword with high order bits forced to ZERO. This halfword replaces the contents of the register specified by R1.

The load complement short instruction causes the two's complement value of the 4-bit second operand to be expanded to a 16-bit halfword with high order bits forced to one. This value replaces the contents of the register specified by R1.

When the Load instructions operate on fixed point data, the Condition Code indicates ZERO (no flags), negative (L flag), or positive (G flag) value.

In the RR format, if R1 equals R2, the Load instruction functions as a test on the contents of the register.

Example: LCS

Assembler Notation		Machine Code	Comments
LCS	REG8,7	2587	LOAD -7 INTO REG8

Result of LCS Instruction

(REG8) = FFF9
Condition Code = 0001 (L = 1)

Load Halfword (LH)
Load Halfword Immediate (LHI)

Assembler Notation		Op-Code	Format
LH	R1,A (X2)	48	RX
LHI	R1,I (X2)	C8	RI

Operation

The halfword second operand replaces the contents of the register specified by R1.

LH: (R1) ← [A + (X2)]
LHI: (R1) ← I + (X2)

Condition Code

C	V	G	L	
0	0	0	0	Value is ZERO
0	0	0	1	Value is not ZERO
0	0	1	0	Value is not ZERO

Programming Note

These instructions may be used to preset a register with an index value, load a register with the first operand for a subsequent arithmetic operation (e.g., add, multiply), or set the Condition Code for supplemental testing by a Branch on Condition instruction.

When the Load Halfword instructions operate on fixed point data, the Condition Code indicates zero (no flags), negative (L flag), or positive (G flag) value.

In the RX format, the second operand must be located on a halfword boundary.

INSTRUCTION

Load Multiple (LM)

Assembler Notation		Op-Code	Format
LM	R1,A (X2)	D1	RX

Operation

Successive registers, starting with the register specified by R1, are loaded from successive memory locations, starting with the location specified as the effective address of the second operand. Each register is loaded with a halfword from memory. The process stops when Register 15 has been loaded.

1. (R1) ← [A + (X2)]
2. R1: X'F'
   if R1 = X'F', then the instruction is finished.
   if R1 ≠ X'F', then:
3. R1 ← R1 + 1
4. A ← A+2, return to Step 1.

Condition Code

Unchanged

Programming Note

The second operand must be located on a halfword boundary.

Load Byte (LB)
Load Byte Register (LBR)

Assembler Notation		Op-Code	Format
LB	R1,A (X2)	D3	RX
LBR	R1,R2	93	RR

Operation

The 8-bit second operand replaces the least significant bits (Bits 8:15) of the register specified by R1. Bits 0:7 of the register are forced to ZERO.

LB: R1(8:15)←[A + (X2)]
R1(0:7) ←ZERO
LBR: R1(8:15)←R2 (8:15)
R1(0:7) ←ZERO

Condition Code

Unchanged

Programming Note

In the Load Byte Register instruction, the second operand is taken from the least significant eight bits (Bits 8:15) of the register specified by R2.

INSTRUCTION

Exchange Byte Register (EXBR)

Assembler Notation		Op-Code	Format
EXBR	R1,R2	94	RR

Operation

The two 8-bit bytes contained in the register specified by R2 are exchanged and loaded into the register specified by R1. The register specified by R2 is unchanged.s,

EXBR: R1 (0:7) ⟵ R2 (8:15)
R1 (8:15) ⟵ R2 (0:7)

Condition Code

Unchanged

Programming Note

R1 and R2 may specify the same register. In this case, the two bytes in the register specified by R2 are exchanged.

Example: EXBR

Assembler Notation		Machine Code	Comments
LHI	REG7,X'3C4D'	C870 3C4D	(REG7) = 3C4D
LHI	REG3,X'1234'	C830 1234	(REG3) = 1234
EXBR	REG7,REG3	9473	

Result of EXBR Instruction

(REG7) = 3412
(REG3) = 1234
Condition Code = Unchanged

INSTRUCTION

Store Halfword (STH)

Assembler Notation		Op-Code	Format
STH	R1,A(X2)	40	RX

Operation

The 16-bit contents of the register specified by R1 replace the contents of the halfword memory location specified by the effective address of the second operand.

STH: (R1)⟶[A + (X2)]

Condition Code

Unchanged

Programming Note

The second operand location must be on a halfword boundary.

This instruction is subject to Memory Protect.

INSTRUCTION

Store Multiple (STM)

Assembler Notation		Op-Code	Format
STM	R1,A (X2)	D0	RX

Operation

The halfword contents of registers, starting with the register specified by R1, replace the contents of successive memory locations, starting with the location specified by the effective address of the second operand. The process stops when Register 15 has been stored.

STM:
1. (R1) ⟶ [A + (X2)]
2. R1: X'F'
   if R1 = X'F', then the instruction is finished.
   if R1 ≠ X'F', then:
3. R1 ⟵ R1 + 1
4. A ⟵ A + 2, return to Step 1.

Condition Code

Unchanged

Programming Note

The second operand location must be on a halfword boundary.

This instruction is subject to Memory Protect.

The Store Multiple (STM) instruction, in conjunction with the Load Multiple (LM) instruction, is an aid to subroutine execution. They permit the easy saving and restoring of the registers required by the subroutine. The Store Multiple instruction can be used upon entering the subroutine, and the Load Multiple can be the last instruction executed before returning from the subroutine.

Store Byte (STB)
Store Byte Register (STBR)

Assembler Notation		Op-Code	Format
STB	R1,A (X2)	D2	RX
STBR	R1,R2	92	RR

Operation

The least significant eight bits (Bits 8:15) of the register specified by R1 are stored in the second operand location.

STB: [R1(8:15)] ⟶ [A + (X2)]
STBR: [R1(8:15)] ⟶ R2(8:15)

Condition Code

Unchanged

Programming Note

The Store Byte (RX) instruction is subject to memory protect.

In the Store Byte Register instruction, the 8-bit quantity is stored in Bits 8:15 of the register specified by R2. Bits 0:7 of the register are unchanged.

Example: STBR

Assembler Notation		Machine Code	Comments
LHI	REG4, X'7531'	C840 7531	(REG4) = 7531
LHI	REG3, X'8642'	C830 8642	(REG3) = 8642
.			
.			
.			
STBR	REG4, REG3	9243	

Result of STBR Instruction

(REG4) = 7531
(REG3) = 8631
Condition Code = Unchanged

Compare Logical Halfword (CLH)
Compare Logical Halfword Register (CLHR)
Compare Logical Halfword Immediate (CLHI)

Assembler Notation		Op-Code	Format
CLH	R1,A (X2)	45	RX
CLHR	R1,R2	05	RR
CLHI	R1,I (X2)	C5	RI

Operation

The first operand, the contents of the register specified by R1, is compared logically to the second operand. The result is indicated by the Condition Code setting. Neither operand is changed.

CLH: (R1): [A + (X2)]
CLHR: (R1): (R2)
CLHI: (R1): I + (X2)

Condition Code

C	V	G	L	
0	X	0	0	First operand equal to second
1	X	0	1	First operand less than second
1	X	1	0	First operand less than second
0	X	0	1	First operand greater than second
0	X	1	0	First operand greater than second

Programming Note

In the RX format, the second operand must be located on a halfword boundary.

The state of the V flag is undefined.

It is meaningful to check the following condition code mask (M1) after a logical comparison:

Mask	True/False*	Inference
3	False	First operand equal to second
3	True	First operand not equal to second
8	False	First operand not less than second
8	True	First operand less than second

*Refer to Chapter 4, Branching, for True/False concept in branch instructions.

INSTRUCTION

Compare Logical Byte (CLB)

Assembler Notation		Op-Code	Format
CLB	R1,A(X2)	D4	RX

Operation

The byte quantity, contained in Bits 8:15 of the register specified by R1, is compared with the 8-bit second operand. The result is indicated by the Condition Code setting. Neither operand is changed.

CLB: R1(8:15) : [A + (X2)]

Condition Code

C	V	G	L	
0	0	0	0	First operand equal to second
1	0	0	1	First operand less than second
0	0	1	0	First operand greater than second

Programming Note

It is meaningful to check the following condition code mask (M1) after a logical comparison:

Mask	True/False*	Inference
3	False	First operand equal to second
3	True	First operand not equal to second
8	False	First operand not less than second
8	True	First operand less than second

*Refer to Chapter 4, Branching, for True/False concept in branch instructions.

AND Halfword (NH)
AND Halfword Register (NHR)
AND Halfword Immediate (NHI)

Assembler Notation		Op-Code	Format
NH	R1,A (X2)	44	RX
NHR	R1,R2	04	RR
NHI	R1,I (X2)	C4	RI

Operation

The logical product of the 16-bit second operand and the contents of the register specified by R1 replace the contents of the register specified by R1. The 16-bit logical product is formed on a bit-by-bit basiss.

NHR: (R1) ← (R1) AND (R2)
NH: (R1) ← (R1) AND [A + (X2)]
NHI: (R1) ← (R1) AND I + (X2)

Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

In the RX formats, the second operand must be located on a halfword boundary.

When operating on fixed-point data, the Condition Code indicates ZERO (no flags), negative (L flag), or positive (G flag) result.

OR Halfword (OH)
OR Halfword Register (OHR)
OR Halfword Immediate (OHI)

Assembler Notation		Op-Code	Format
OH	R1,A (X2)	46	RX
OHR	R1,R2	06	RR
OHI	R1,I (X2)	C6	RI

Operation

The logical sum of the 16-bit second operand and the contents of the register specified by R1 replace the contents of the register specified by R1. The 16-bit sum is formed on a bit-by-bit basis.

OH: (R1) ← (R1) OR [A + (X2)]
OHR: (R1) ← (R1) OR (R2)
OHI: (R1) ← (R1) OR I + (X2)

Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

In the RX format, the second operand must be located on a halfword boundary.

When operating on fixed-point data, the Condition Code indicates ZERO (no flags), negative (L flag), or positive (G flag) result.

Exclusive OR Halfword (XH)
Exclusive OR Halfword Register (XHR)
Exclusive OR Halfword Immediate (XHI)

Assembler Notation		Op-Code	Format
XH	R1,A (X2)	47	RX
XHR	R1,R2	07	RR
XHI	R1,I (X2)	C7	RI

Operation

The logical difference of the 16-bit second operand and the contents of the register specified by R1 replace the contents of the register specified by R1. The 16-bit difference is formed on a bit-by-bit basis.

XH: (R1) ← (R1) XOR [A + (X2)]
XHR: (R1) ← (R1) XOR (R2)
XHI: (R1) ← (R1) XOR I + (X2)

Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

In the RX formats, the second operand must be located on a halfword boundary.

When operating on fixed-point data, the Condition Code indicates ZERO (no flags), negative (L flag), or positive (G flag) result.

INSTRUCTION

Test Halfword Immediate (THI)

Assembler Notation		Op-Code	Format
THI	R1,I (X2)	C3	RI

Operation

Each bit in the 16-bit second operand is logically ANDed with the corresponding bit contained in the register specified by R1. Neither operand is changed.

THI: (R1) AND I + (X2)

Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

When operating on fixed-point data, the Condition Code indicates ZERO (no flags), negative (L flag), or positive (G flag) result.

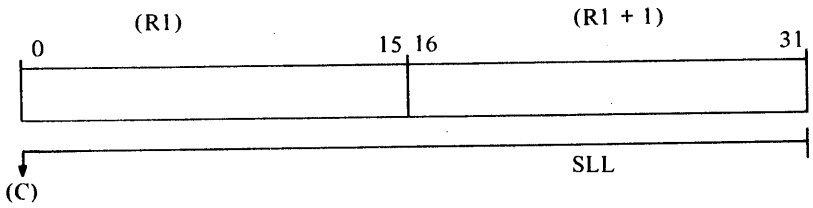
INSTRUCTION

Shift Left Logical (SLL)

Assembler Notation		Op-Code	Format
SLL	R1,I (X2)	ED	RI

Operation

In this instruction, the register specified by R1 and the register implied by the value of R1+1 are linked together to form a fullword operand. This operand is shifted left the number of binary places specified by the second operand. Bits shifted out of Position 0 in the register specified by R1 are shifted through the carry flag of the Condition Code, and then lost. The last bit shifted remains in the carry flag. Bits shifted from Position 0 of the second register move into Position 15 of the first. Zeros are moved into Position 15 of the second register.



Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is not ZERO
X	0	1	0	Result is not ZERO
1	0	X	X	Carry

Programming Note

The shift count is specified by the least significant five bits of the second operand.

The state of the C flag indicates the state of the last bit shifted out of Position 0 of register R1.

If the second operand specifies a shift of zero places, the Condition Code is set in accordance with the value contained in the registers. The state of the C flag is undefined.

The register specified by R1 must be an even numbered register.

When the registers R1 and R1+1 contain fixed point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

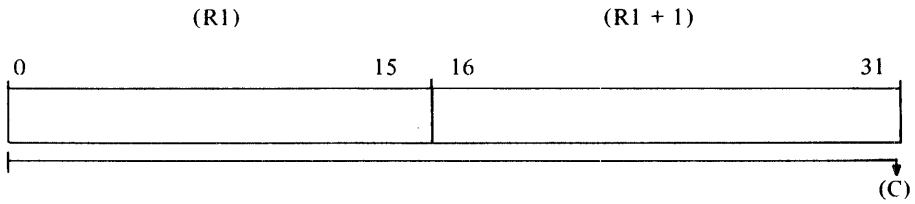
INSTRUCTION

Shift Right Logical (SRL)

Assembler Notation	Op-Code	Format
SRL R1,I (X2)	EC	RI

Operation

In this instruction, the register specified by R1 and the register implied by the value of R1+1 are linked together to form a fullword operand. This operand is shifted right the number of binary places specified by the second operand. Bits shifted out of Position 15 of the second register are shifted through the carry flag of the Condition Code, and then lost. The last bit shifted remains in the carry flag. Bits shifted from Position 15 of the first register move into Position 0 of the second. Zeros are moved into Position 0 of the first register.



Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is not ZERO
X	0	1	0	Result is not ZERO
1	0	X	X	Carry

Programming Note

The shift count is specified by the least significant five bits of the second operand.

The state of the C flag indicates the state of the last bit shifted out of Position 15 of register R1+1.

When the first operand contains fixed point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

If the second operand specifies a shift of zero places, the Condition Code is set in accordance with the value contained in the registers. The state of the C flag is undefined.

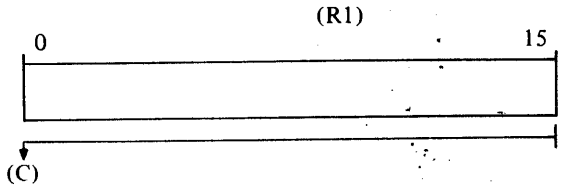
The register specified by R1 must be an even numbered register.

Shift Left Halfword Logical (SLHL)
Shift Left Logical Short (SLLS)

Assembler Notation		Op-Code	Format
SLHL	R1,I (X2)	CD	RI
SLLS	R1,N	91	SF

Operation

Bits 0:16 of the register specified by R1 are shifted left the number of places specified by the second operand. Bits shifted out of Position 0 are shifted through the carry flag and lost. The last bit shifted remains in the carry flag. Zeros are shifted into Position 15.



Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is not ZERO
X	0	1	0	Result is not ZERO
1	0	X	X	Carry

Programming Note

In the RI format, the shift count is specified by the least significant four bits of the second operand.

In either format, the maximum shift count is 15.

The state of the C flag indicates the state of the last bit shifted out of Position 0.

When the register specified by R1 contains fixed point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

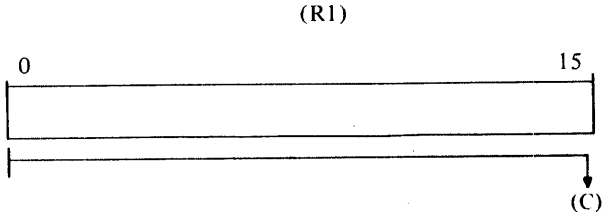
If the second operand specified a shift of zero places, the Condition Code is set in accordance with the value contained in the register. The state of the C flag is undefined.

Shift Right Halfword Logical (SRHL)
Shift Right Logical Short (SRLS)

Assembler Notation		Op-Code	Format
SRHL	R1,I (X2)	CC	RI
SRLS	R1,N	90	SF

Operation

Bits 0:15 of the register specified by R1 are shifted right the number of places specified by the second operand. Bits shifted out of Position 15 are shifted through the carry flag and lost. The last bit shifted remains in the carry flag. Zeros are shifted into Position 0.



Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is not ZERO
X	0	1	0	Result is not ZERO
1	0	X	X	Carry

Programming Note

In the RI format, the shift count is specified by the least significant four bits of the second operand.

In either format, the maximum shift count is 15.

The state of the C flag indicates the state of the last bit shifted out of Position 15.

When the register specified by R1 contains fixed point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

If the second operand specifies a shift of zero places, the Condition Code is set in accordance with the value contained in the register. The state of the C flag is undefined.

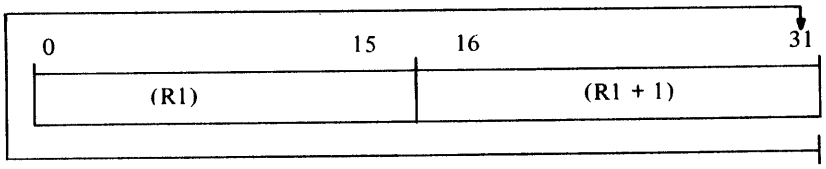
INSTRUCTION

Rotate Left Logical (RLL)

Assembler Notation		Op-Code	Format
RLL	R1,I (X2)	EB	RI

Operation

In this instruction, the register specified by R1 and the register implied by the value of R1+1 are linked together to form a fullword operand. This operand is rotated left the number of binary places specified by the second operand. Bits moved from Position 0 of the first register move into Position 15 of the second register.



Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

The register specified by R1 must be an even numbered register.

The shift count is specified by the least significant five bits of the second operand.

If the second operand specifies a shift of zero places, the Condition Code is set in accordance with the value contained in the registers.

When the register specified by R1 contains fixed point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

Example: RLL

1.

Assembler Notation		Machine Code	Comments
LHI	REG8,X'5678'	C880 5678	(REG8) = 5678
LHI	REG9,X'9ABC'	C890 9ABC	(REG9) = 9ABC
RLL	REG8,X'0004'	EB80 0004	

Result of RLL Instruction

(REG8) = 6789 (REG9) = ABC5
Condition Code = 0010 (G = 1)

2.

Assembler Notation		Machine Code	Comments
LHI	REG8, X'8888'	C880 8888	(REG8) = 8888
LHI	REG9, X'8888'	C890 8888	(REG9) = 8888
RLL	REG8, X'0003'	EB80 0003	

Result of RLL Instruction

(REG9) = 4444 (REG8) = 4444
Condition Code = 0010 (G = 1)

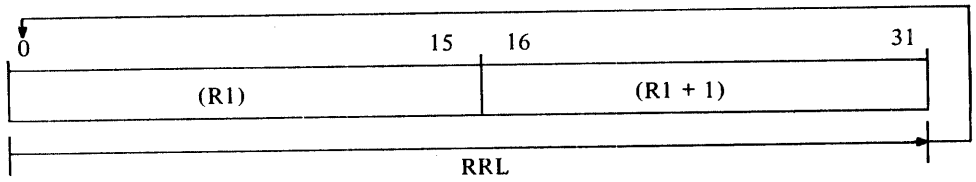
INSTRUCTION

Rotate Right Logical (RRL)

Assembler Notation		Op-Code	Format
RRL	R1,I (X2)	EA	RI

Operation

In this instruction, the register specified by R1 and the register implied by the value of R1+1 are linked together to form a fullword operand. This operand is rotated right the number of binary places specified by the second operand. Bits moved from Position 15 of the second register move into Position 0 of the first register.



Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

The register specified by R1 must be an even numbered register.

The shift count is specified by the least significant five bits of the second operand.

If the second operand specifies a shift of zero places, the Condition Code is set in accordance with the value contained in the registers.

When the register specified by R1 contains fixed point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

Example: RRL

1.

Assembler Notation		Machine Code	Comments
LHI	REG4, X'1234'	C840 1234	(REG4) = 1234
LHI	REG5, X'5678'	C850 5678	(REG5) = 5678
RRL	REG4, X'0004'	EA40 0004	

Result of RRL Instruction

(REG4) = 8123 (REG5) = 4567
Condition Code = 0001 (L = 1)

2.

Assembler Notation		Machine Code	Comments
LHI	REG4, X'1111'	C840 1111	(REG4) = 1111
LHI	REG5, X'1111'	C850 1111	(REG5) = 1111
RRL	REG4, X'0001'	EA40 0001	

Result of RRL Operation

(REG4) = 8888 (REG5) = 8888
Condition Code = 0001 (L = 1)

INSTRUCTION

Add to Top of List (ATL)
Add to Bottom of List (ABL)

Assembler Notation		Op-Code	Format
ATL	R1,A(X2)	64	RX
ABL	R1,A(X2)	65	RX

Operation

The register specified by R1 contains the halfword element to be added to the list. The list is located in memory at the address specified by the second operand. The number of slots used tally is compared with the number of slots in the list. If the number of slots used equals the number of slots in the list, an overflow condition exists. The element is not added to the list and the overflow flag in the Condition Code is set. If the number of slots used tally is less than the number of slots in the list, it is incremented by one, the appropriate pointer (current top or next bottom) is changed, and the element is added to the list.

Condition Code

C	V	G	L	
0	0	0	0	Element added successfully
0	1	0	0	List overflow

Programming Note

These instructions manipulate circular lists as described in the introduction to this chapter.

The second operand location must be on a halfword boundary.

These instructions are subject to memory protect.

The Add to Top of List instruction manipulates the current top pointer in the list. If no overflow occurs, the current top pointer, which points to the last element added to the top of the list, is decremented by one and the element is inserted in the slot pointed to by the new current top pointer. If the current top pointer was zero on entering this instruction, the current top pointer is set to the maximum slot number in the list. This condition is referred to as list wrap.

The Add to Bottom of List instruction manipulates the next bottom pointer. If no overflow occurs, the element is inserted in the slot pointed to by the next bottom pointer, and the next bottom pointer is incremented by one. If the incremented next bottom pointer is greater than the maximum slot number in the list, the next bottom pointer is set to zero. This condition is referred to as list wrap.

Examples:

See examples for next instruction.

Remove from Top of List (RTL)
Remove from Bottom of List (RBL)

Assembler Notation		Op-Code	Format
RTL	R1,A(X2)	66	RX
RBL	R1,A(X2)	67	RX

Operation

The list is located at the address specified by the second operand. The halfword element removed from the list replaces the contents of the register specified by R1. If, at the start of the instruction execution, the number of slots used tally is ZERO, the list is already empty and the instruction terminates with the overflow flag set in the Condition Code. This condition is referred to as list underflow; in this case, (R1) is undefined. If underflow does not occur, the number of slots used tally is decremented by one, the appropriate pointer is changed, and the element is extracted and placed in the register specified by R1.

Condition Code

C	V	G	L	
0	0	0	0	List is now empty
0	0	1	0	List is not yet empty
0	1	0	0	List was already empty

Programming Note

These instructions manipulate circular lists as described in the introduction to this chapter.

The second operand location must be on a halfword boundary.

In the case of list underflow, the contents of the register specified by R1 are undefined.

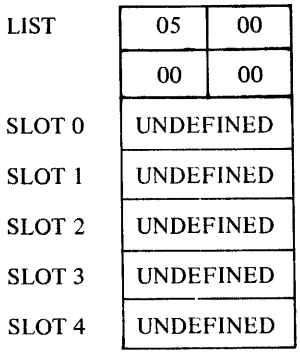
The Remove from Top of List instruction manipulates the current top pointer. If no underflow occurs, the current top pointer points to the element to be extracted. The element is extracted, and placed in the register specified by R1. The current top pointer is incremented by one and compared to the maximum slot number. If the current top pointer is greater than the maximum slot number, the current top pointer is set to ZERO. This condition is referred to as list wrap.

The Remove from Bottom of List instruction manipulates the next bottom pointer. If no underflow occurs, and the next bottom pointer is ZERO, it is set to the maximum slot number (list wrap); otherwise, it is decremented by one, and the element now pointed to is extracted and placed in the register specified by R1.

Examples: List Instructions (ATL, ABL, RTL, RBL)

The following are examples of the use of the four list processing instructions.

The original list is normally set up as shown in Figure 3-4.



where bytes at

LIST = number of total slots
= 5 (in this example)

LIST + 1 = number of entries used
= 0

LIST + 2 = current top of list
= slot 0

LIST + 3 = next bottom of list
= slot 0

Current Top Pointer = Slot 0
Next Bottom Pointer = Slot 0

Figure 3-4. List Processing Instructions

Labels	Assembler Notation		Results and Comments
	LIS	REG0,0	
	STB	REG0,LIST+1	INITIALIZE NO. OF ENTRIES USED TO 0
	STH	REG0,LIST+2	INITIALIZE POINTERS TO 0
	LIS	REG1,1	REGISTERS 1 THRU 6 CONTAIN
	LIS	REG2,2	1 THRU 6 RESPECTIVELY
	LIS	REG3,3	
	LIS	REG4,4	
	LIS	REG5,5	
	LIS	REG6,6	
	STB	REG5,LIST	TOTAL NO. OF ENTRIES = 5

```
REF1    ATL    REG1  LIST
```

LIST	05	01
	04	00
SLOT 0	UNDEFINED	
SLOT 1	UNDEFINED	
SLOT 2	UNDEFINED	
SLOT 3	UNDEFINED	
SLOT 4	0001	

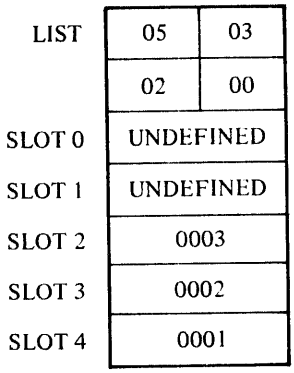
Condition Code = 0000
Current Top Pointer = Slot 4
Next Bottom Pointer = Slot 0

```
REF2    ATL    REG2, LIST
```

LIST	05	02
	03	00
SLOT 0	UNDEFINED	
SLOT 1	UNDEFINED	
SLOT 2	UNDEFINED	
SLOT 3	0002	
SLOT 4	0001	

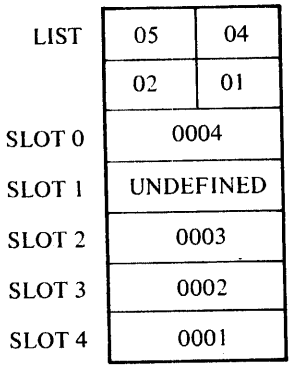
Condition Code = 0000
Current Top Pointer = Slot 3
Next Bottom Pointer = Slot 0

REF3 ATL REG3, LIST



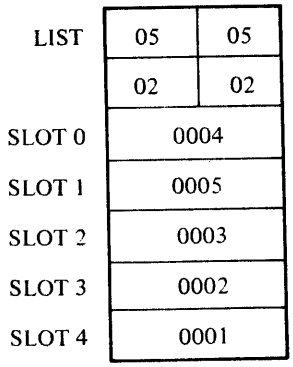
Condition Code = 0000
Current Top Pointer = Slot 2
Next Bottom Pointer = Slot 0

REF4 ABL REG4, LIST



Condition Code = 0000
Current Top Pointer = Slot 2
Next Bottom Pointer = Slot 1

REF5 ABL REG5, LIST



Condition Code = 0000
Current Top Pointer = Slot 2
Next Bottom Pointer = Slot 2

REF6 ABL REG6, LIST

LIST	05	05
	02	02
SLOT 0	0004	
SLOT 1	0005	
SLOT 2	0003	
SLOT 3	0002	
SLOT 4	0001	

Condition Code = 0100
Condition Code = 0100 (List Overflow)
Next Bottom Pointer = Slot 2

REF7 RTL REG7, LIST

LIST	05	04
	03	02
SLOT 0	0004	
SLOT 1	0005	
SLOT 2 X	0003	
SLOT 3	0002	
SLOT 4	0001	

(REG 7) = 0003
Condition Code = 0010
Current Top Pointer = Slot 3
Next Bottom Pointer = Slot 2

REF8 RBL REG8, LIST

LIST	05	03
	03	01
SLOT 0	0004	
SLOT 1 X	0005	
SLOT 2 X	0003	
SLOT 3	0002	
SLOT 4	0001	

(REG 8) = 0005
Condition Code = 0010
Current Top Pointer = Slot 3
Next Bottom Pointer = Slot 1

NOTE
X = Entry removed from list, and is not accessible through further manipulation of list instructions.

REF9 RTL REG9, LIST

LIST		05	02
		04	01
SLOT 0		0004	
SLOT 1	X	0005	
SLOT 2	X	0003	
SLOT 3	X	0002	
SLOT 4		0001	

(REG 9) = 0002
Condition Code = 0010
Current Top Pointer = Slot 4
Next Bottom Pointer = Slot 1

REF10 RBL REG10, LIST

LIST		05	01
		04	00
SLOT 0	X	0004	
SLOT 1	X	0005	
SLOT 2	X	0003	
SLOT 3	X	0002	
SLOT 4		0001	

(REG 10) = 0004
Condition Code = 0010
Current Top Pointer = Slot 4
Next Bottom Pointer = Slot 0

REF11 RTL REG11, LIST

LIST		05	00
		00	00
SLOT 0	X	0004	
SLOT 1	X	0005	
SLOT 2	X	0003	
SLOT 3	X	0002	
SLOT 4	X	0001	

(REG 11) = 0001
Condition Code = 0000 (List is now empty)
Current Top Pointer = Slot 0
Next Bottom Pointer = Slot 0

NOTE

X =Entry removed from list, and is not accessible through further manipulation of list instructions.

REF12 RTL REG12, LIST

LIST		05	00
		00	00
SLOT 0	X	0004	
SLOT 1	X	0005	
SLOT 2	X	0003	
SLOT 3	X	0002	
SLOT 4	X	0001	

(REG 12) = UNDEFINED
Condition Code = 0100 (List Was Already Empty)
Current Top Pointer = Slot 0
Next Bottom Pointer = Slot 0

NOTE

X = Entry removed from list, and is not accessible through further manipulation of list instructions.

CHAPTER 4
BRANCHING

In normal operations, the Processor executes instructions in sequential order. The Branch instructions allow this sequential mode of operation to be varied, so that programs can loop, transfer control to subroutines, or make decisions based on the results of previous operations.

OPERATIONS

The second operand in Branch instructions is the address of the memory location to which control is transferred. The address may be contained in a register, or it may be specified in the instruction as the second operand address.

Decision Making

The Conditional Branch insturctions permit the program to make decisions based on previous results. In these instructions, the R1 field contains a 4-bit mask, M1, which is tested against the Condition Code. The result of the test determines whether the branch is taken, or the next sequential instruction is executed.

The following examples show current Condition Code, mask specified in a Branch instruction, and the result of the test on which a branch or no branch decision is made.

Current Condition Code	Mask (M1)	Result of Test	(True/False)
0000	0010	0000	(False)
0001	1010	0000	(False)
1001	1000	1000	(True)
0100	0100	0100	(True)
1010	0010	0010	(True)
0010	0011	0010	(True)
0010	0000	0000	(False)

Subroutine Linkage

The Branch and Link instructions allow branching to subroutines in such a way that a return address is passed to the subroutine. In these instructions, the address of the instruction immediately following the Branch instruction is saved in the register specified by R1.

BRANCH INSTRUCTION FORMATS

The Branch instructions use the Register to Register (RR), the Short Form (SF), and the Register and Indexed Storage (RX) format.

BRANCH INSTRUCTIONS

The instructions described in this section are:

BFC	Branch on False Condition
BFCR	Branch on False Condition Register
BFBS	Branch on False Condition Backward Short
BFFS	Branch on False Condition Forward Short
BTC	Branch on True Condition
BTCR	Branch on True Condition Register
BTBS	Branch on True Condition Backward Short
BTFS	Branch on True Condition Forward Short
BAL	Branch and Link
BALR	Branch and Link Register
BXLE	Branch on Index Low or Equal
BXH	Branch on Index High

Branch on True Condition (BTC)
Branch on True Condition Register (BTCR)
Branch on True Condition Backward Short (BTBS)
Branch on True Condition Forward Short (BTFS)

Assembler Notation		Op-Code	Format
BTC	M1,A(X2)	42	RX
BTCR	M1,R2	02	RR
BTBS	M1,N	20	SF
BTFS	M1,N	21	SF

Operation

The Condition Code of the Program Status Word is tested for the conditions specified by the mask field, M1. If any of the conditions tested are found to be true, a branch is executed to the second operand location. If none of the conditions tested is found to be true, the next sequential instruction is executed.

Tested Condition True:

BTC: [PSW (16:31)] ← A + (X2)
BTCR: [PSW (16:31)] ← (R2)
BTBS: [PSW (16:31)] ← [PSW (16:31)] -2N
BTFS: [PSW (16:31)] ← [PSW (16:31)] +2N

Tested Condition False:

BTC: [PSW (16:31)] ← [PSW (16:31)] +4

BTBS:
BTFS:
BTCR: } [PSW (16:31)] ← [PSW (16:31)] +2

Condition Code

Unchanged

Programming Note

In the RR format, the branch address is contained in the register specified by R2.

In the SF format, the N field contains the number of *halfwords* to be added or subtracted from the current Location Counter to obtain the branch address.

In the RR and RX formats, the branch address must be located on a halfword boundary.

Example: BTC

Assembler Notation		Machine Code	Comments
LH	R1,X 100	4810 0100	Load halfword (X'1234') located at X'100'. Condition Code is set to CVGL = 0010. Mask is 3, i.e., M1 = 0011. Perform logical AND between CVGL and M1, i.e., 0010 AND 0011. The result is 0010, i.e., true; therefore, a branch is taken to LOC.
BTC	3, LOC	4230 ABC0	

Branch on False Condition (BFC)
Branch on False Condition Register (BFCR)
Branch on False Condition Backward Short (BFBS)
Branch on False Condition Forward Short (BFFS)

Assembler Notation		Op-Code	Format
BFC	M1,A(X2)	43	RX
BFCR	M1,R2	03	RR
BFBS	M1,N	22	SF
BFFS	M1,N	23	SF

Operation

The Condition Code of the Program Status Word is tested for the conditions specified in the mask field, M1. If all conditions tested are found to be false, a branch is executed to the second operand location. If any of the conditions tested is found to be true, the next sequential instruction is executed.

Tested Condition False

BFC: [PSW (16:31)] ← A+(X2)
BFCR: [PSW (16:31)] ← (R2)
BFBS: [PSW (16:31)] ← [PSW (16:31)] -2N
BFFS: [PSW (16:31)] ← [PSW (16:31)] +2N

Tested Condition True

BFC: [PSW (16:31)] ← [PSW (16:31)] +4

BFCR:
BFBS:
BFFS: } [PSW (16:31)] ← [PSW (16:31)] +2

Condition Code

Unchanged

Programming Note

In the RR format, the branch address is contained in the register specified by R2.

In the SF format, the N field contains the number of *halfwords* to be added to or subtracted from the current Location Counter to obtain the branch address.

In the RR and RX formats, the branch address must be located on a halfword boundary.

Branch on False Condition with a mask of 0 is an Unconditional Branch.

Example: BFC

Assembler Notation		Machine Code	Comments
LCS	R1,2	2512	(R1) = X'FFFE'. Condition Code, CVGL = 0001 Mask is 1001. Perform logical AND between mask and CVGL, i.e., 1001 AND 0001. The result is 0001, i.e., true, therefore, a branch is not taken to LOC.
BFC	9, LOC	4390 ABC0	

Branch and Link (BAL)
Branch and Link Register (BALR)

Assembler Notation		Op-Code	Format
BAL	R1,A(X2)	41	RX
BALR	R1,R2	01	RR

Operation

The address of the next sequential instruction is saved in the register specified by R1, and a branch is taken to the second operand address.

BAL: R1 ← [PSW (16:31)] + 4
[PSW (16:31)] ← A + (X2)

BALR: R1 ← [PSW (16:31)] + 2
[PSW (16:31)] ← (R2)

Condition Code

Unchanged

Programming Note

The second operand location must be on a halfword boundary.

The branch address is calculated before the register specified by R1 is changed. R1 may specify the same register as X2 or R2.

Example: BAL

The following example illustrates the use of the BAL instruction. The instruction causes control to be transferred to a subroutine called SUBROUT. After completion of the subroutine, the linking register is used to branch back to the next sequential instruction after the BAL, i.e., the instruction labeled RETURN.

	Labels	Assembler Notation		Comments
MAIN PROG	BEGIN	BAL	REG4,SUBROUT	TRANSFER TO SUBROUT
	RETURN	XHR	R6,R6	
		STH	R6,LAB+4	
		:		
SUBROUTINE	SUBROUT	LH	R8,LOC	THE RETURN ADDRESS OF THE SUBROUTINE IS IN REG4
		AHI	R8,10	
		:		
	RTNEND	BR	REG4	RETURN TO XHR INST

NOTE

Within the subroutine, the linking register (REG4 in the example) should not be used unless stored and reloaded within the subroutine.

Result of BAL Instruction

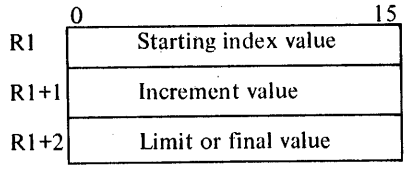
Condition Code = Unchanged

INSTRUCTION

Branch on Index Low or Equal (BXLE)

Assembler Notation		Op-Code	Format
BXLE	R1,A(X2)	C1	RX

Set Up



Prior to execution of this instruction, the register specified by R1 must contain a starting index value. The register specified by R1+1 must contain an increment value. The register specified by R1+2 must contain a comparand (limit or final value). All values may be signed.

Operation

Execution of this instruction causes the increment value to be added to the index value. The result is logically compared to the limit or final value. If the index value is less than or equal to the limit value, a branch is executed to the second operand location. If the index value is greater than the limit value, the next sequential instruction is executed.

BXLE: (R1) ←——— (R1) + (R1+1)
(R1): (R1+2)
If (R1) ≤ (R1+2), then [PSW (16:31)] ← A + (X2)
If (R1) > (R1+2), then [PSW (16:31)] ← [PSW (16:31)] + 4

Condition Code

Unchanged

Programming Note

The incremented index value replaces the contents of the register specified by R1.

The register specified by R1 must not be greater than 13.

The second operand location must be on a halfword boundary.

The branch address is calculated before incrementing the starting index value contained in the register specified by R1.

The register specified by R1 may be the same as X2.

Example: BXLE

Transfer 10 bytes in memory starting at Memory Location Labelled BUF0 to memory location labelled BUF1.

Labels	Assembler Notation		Comments
	LIS	REG3,0	(REG 3) = STARTING INDEX VALUE = 0
	LIS	REG4,1	(REG 4) = INCREMENT VALUE = 1
	LIS	R5,9	(REG 5) = FINAL VALUE = 9
AGAIN	LB	REG0, BUF0(R3)	(REG 0) = 1 BYTE FROM BUF0
	STB	REG0, BUF1(R3)	COPY 1 BYTE TO BUF1
	BXLE	R3,AGAIN	IF (REG 3) = (REG 5), DONE
	.		
	.		
	.		
BUF0	DS	10	
BUF1	DS	10	

Result of BXLE Instruction

Condition Code = Unchanged by BXLE Instruction
(REG1) = 000A
(REG2) = 0001
(REG3) = 0009

INSTRUCTION

Branch on Index High (BXH)

Assembler Notation		Op-Code	Format
BXH	R1,A(X2)	C0	RX

Set Up

	0	15
R1	Starting index value	
R1+1	Increment value	
R1+2	Limit or final value	

Prior to execution of this instruction, the register specified by R1 must contain a starting index value. The register specified by R1+1 must contain an increment value. The register specified by R1+2 must contain a comparand (limit or final value). All values may be signed.

Operation

Execution of this instruction causes the increment value to be added to the index value. The result is logically compared to the limit or final value. If the index value is greater than the limit value, a branch is executed to the second operand location. If the index value is equal to or less than the limit value, the next sequential instruction is executed.

BXH: (R1) ← (R1) + (R1 + 1)
(R1): (R1 + 2)
if (R1) ≤ (R1 + 2), then [PSW (16:31)] ← [PSW(16:31)] + 4
if (R1) > (R1 + 2), then [PSW (16:31)] ← A + (X2)

Condition Code

Unchanged

Programming Note

The incremented index value replaces the contents of the register specified by R1.

The second operand location must be on a halfword boundary.

The branch address is calculated before incrementing the starting index value contained in the register specified by R1.

The register specified by R1 may be the same as X2.

The register specified by R1 must not be greater than 13.

Example: BXH

The following example shows how to set up a counter (1 - 9) using the BXH instruction.

Label	Assembler Notation		Comment
	LIS	REG1,1	(REG 1) = 0001 (INDEX)
	LIS	REG2,1	(REG 2) = 0001 (INCREMENT)
	LIS	REG3,9	(REG 3) = 0009 (COMPARAND)
BEGIN	BXH	REG1, LABEL	COMPARE INDEX WITH COMPARAND
	LH	R6,COUNT	
	.		
	.		
	.		
	B	BEGIN	BRANCH TO BXH INSTRUCTION
LABEL	LH	R8,RTN	EXIT FROM BXH
	ST	R8,MEM	

Result of BXH Instruction

The code between the instructions labelled BEGIN and LABEL is executed 9 times.

Condition Code = Unchanged by BXH instruction
(REG1) = 000A
(REG2) = 0001
(REG3) = 0009

EXTENDED BRANCH MNEMONICS

The CAL Assembler supports 14 extended branch mnemonics that generate the branch op-code (true or false conditional) and the condition code mask required. The programmer must supply the second operand address (symbolic or absolute). In the case of short format (SF) branch instructions, the second operand branch address must be within ± 15 halfwords of the current location counter. The CAL Assembler determines the backward or forward relationship of the second operand address and generates the appropriate operation code.

Examples of extended branch mnemonic:

```
                .
                .
                .
                LH      R5,LOOP1
                BNZ     LOERR
LAP             SRLS    R6,1
                BNCS    LAP
                BS      CONTIN
LOERR           LIS     R6,0
ERROR1          AIS     R6,1
                SIS     R5,4
                BPS     ERROR1
                SIS     R8,1
                BO      ERROR2
CONTIN          LH      R1,TIME
                .
                .
                .
```

Appendix 4 lists the extended branch mnemonics and the proper operand form to be used with each mnemonic. The actual machine code generated is also listed.

The instructions described in this section are:

BC	Branch on Carry	BP	Branch on Plus
BCR	Branch on Carry Register	BPR	Branch on Plus Register
BCS	Branch on Carry Short	BPS	Branch on Plus Short
BNC	Branch on No Carry	BNP	Branch on Not Plus
BNCR	Branch on No Carry Register	BNPR	Branch on Not Plus Register
BNCS	Branch on No Carry Short	BNPS	Branch on Not Plus Short
BE	Branch on Equal	BO	Branch on Overflow
BER	Branch on Equal Register	BOR	Branch on Overflow Register
BES	Branch on Equal Short	BOS	Branch on Overflow Short
BNE	Branch on Not Equal	BNO	Branch on No Overflow
BNER	Branch on Not Equal Register	BNOR	Branch on No Overflow Register
BNES	Branch on Not Equal Short	BNOS	Branch on No Overflow Short
BL	Branch on Low	BZ	Branch on Zero
BLR	Branch on Low Register	BZR	Branch on Zero Register
BLS	Branch on Low Short	BZS	Branch on Zero Short
BNL	Branch on Not Low	BNZ	Branch on Not Zero
BNLR	Branch on Not Low Register	BNZR	Branch on Not Zero Register
BNLS	Branch on Not Low Short	BNZS	Branch on Not Zero Short
BM	Branch on Minus	B	Branch (Unconditional)
BMR	Branch on Minus Register	BR	Branch Register (Unconditional)
BMS	Branch on Minus Short	BS	Branch Short (Unconditional)
BNM	Branch on Not Minus		
BNMR	Branch on Not Minus Register	NOP	No Operation
BNMS	Branch on Not Minus Short	NOPR	No Operation Register

INSTRUCTION

Branch on Carry (BC)
Branch on Carry Register (BCR)
Branch on Carry Short (BCS)

Assembler Notation		Op-Code + M1	Format
BC	A(X2)	428	RX
BCR	R2	028	RR
BCS	A	208 (Backward) 218 (Forward)	SF

Operation

If the Carry (C) flag in the Condition Code is set, a branch is executed to the second operand location. If the Carry flag is not set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

Example: BCS

Assembler Notation			Machine Code	Comments
SHIFT	SLLS	R9,1	9191	Register 9 is shifted left until the first zero bit is shifted out (left).
	BCS	SHIFT	2081	

INSTRUCTION

Branch on No Carry (BNC)
Branch on No Carry Register (BNCR)
Branch on No Carry Short (BNCS)

Assembler Notation		Op-Code + M1	Format
BNC	A(X2)	438	RX
BNCR	R2	038	RR
BNCS	A	228 (Backward) 238 (Forward)	SF

Operation

If the Carry (C) flag in the Condition Code is not set, a branch is executed to the second operand location. If the Carry flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Equal (BE)
Branch on Equal Register (BER)
Branch on Equal Short (BES)

Assembler Notation		Op-Code + M1	Format
BE	A(X2)	433	RX
BER	R2	033	RR
BES	A	223 (Backward) 233 (Forward)	SF

Operation

If the G flag and the L flag are both reset in the Condition Code, a branch is executed to the second operand location. If either flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

Example: BE

Assembler Notation		Machine Code	Comments
CLHI	R4,X'23'	C540 0023	If R4 contains X'23', a branch is executed to location X'A00'. Otherwise the next sequential instruction is executed.
BE	OPTIN	4330 0A00	

INSTRUCTION

Branch on Not Equal (BNE)
Branch on Not Equal Register (BNER)
Branch on Not Equal Short (BNES)

Assembler Notation		Op-Code + M1	Format
BNE	A(X2)	423	RX
BNER	R2	023	RR
BNES	A	203 (Backward) 213 (Forward)	SF

Operation

If the G flag or the L flag is set in the Condition Code, a branch is executed to the second operand location. If neither flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Low (BL)
Branch on Low Register (BLR)
Branch on Low Short (BLS)

Assembler Notation		Op-Code + M1	Format
BL	A(X2)	428	RX
BLR	R2	028	RR
BLS	A	208 (Backward) 218 (Forward)	SF

Operation

When two operands are compared, the C flag is set in the Condition Code of the PSW if the first operand is less than the second operand. If the Carry (C) flag in the Condition Code is set, a Branch on Low is executed to the second operand address. If the Carry flag is not set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

Example: BL

Assembler Notation		Machine Code	Comments
CLHI	R1,X'FF'	C510 00FF	R1 is compared to X'00FF'.
BL	RESTART	4280 0A00	If R1 is less than X'FF', a branch is taken to memory location X'0A00'.

INSTRUCTION

Branch on Not Low (BNL)
Branch on Not Low Register (BNLR)
Branch on Not Low Short (BNLS)

Assembler Notation		Op-Code + M1	Format
BNL	A(X2)	438	RX
BNLR	R2	038	RR
BNLS	A	228 (Backward) 238 (Forward)	SF

Operation

When two operands are compared, the C flag is not set in the Condition Code of the PSW if the first operand is not less than the second operand. If the Carry (C) flag in the Condition Code is not set, a Branch is executed to the second operand address. If the Carry flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Minus (BM)
Branch on Minus Register (BMR)
Branch on Minus Short (BMS)

Assembler Notation		Op-Code + M1	Format
BM	A(X2)	421	RX
BMR	R2	021	RR
BMS	A	201 (Backward) 211 (Forward)	SF

Operation

If the Less-Than (L) flag in the Condition Code is set, a branch is executed to the second operand location. If the L flag is not set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

Example: BM

Assembler Notation		Machine Code	Comments
SIS	R3,1	2631	If R3 is less than 0 after
BM	CONTINUE	4210 10A0	the subtraction, a branch is taken to X'10A0'.

INSTRUCTION

Branch on Not Minus (BNM)
Branch on Not Minus Register (BNMR)
Branch on Not Minus Short (BNMS)

Assembler Notation		Op-Code + M1	Format
BNM	A(X2)	431	RX
BNMR	R2	031	RR
BNMS	A	221 (Backward) 231 (Forward)	SF

Operation

If the Less-Than (L) flag in the Condition Code is not set, a branch is executed to the second operand location. If the L flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Plus (BP)
Branch on Plus Register (BPR)
Branch on Plus Short (BPS)

Assembler Notation		Op-Code + M1	Format
BP	A(X2)	422	RX
BPR	R2	022	RR
BPS	A	202 (Backward) 212 (Forward)	SF

Operation

If the Greater-Than (G) flag in the Condition Code is set, a branch is executed to the second operand location. If the G flag is not set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Not Plus (BNP)
Branch on Not Plus Register (BNPR)
Branch on Not Plus Short (BNPS)

Assembler Notation		Op-Code + M1	Format
BNP	A(X2)	432	RX
BNPR	R2	032	RR
BNPS	A	222 (Backward) 232 (Forward)	SF

Operation

If the Greater-Than (G) flag in the Condition Code is reset, a branch is executed to the second operand location. If the G flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Overflow (BO)
Branch on Overflow Register (BOR)
Branch on Overflow Short (BOS)

Assembler Notation		Op-Code + M1	Format
BO	A(X2)	424	RX
BOR	R2	024	RR
BOS	A	204 (Backward) 214 (Forward)	SF

Operation

If the Overflow (V) flag in the Condition Code is set, a branch is executed to the second operand location. If the V flag is not set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on No Overflow (BNO)
Branch on No Overflow Register (BNOR)
Branch on No Overflow Short (BNOS)

Assembler Notation		Op-Code + M1	Format
BNO	A(X2)	434	RX
BNOR	R2	034	RR
BNOS	A	224 (Backward) 234 (Forward)	SF

Operation

If the Overflow (V) flag in the Condition Code is not set, a branch is executed to the second operand location. If the V flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Zero (BZ)
Branch on Zero Register (BZR)
Branch on Zero Short (BZS)

Assembler Notation		Op-Code + M1	Format
BZ	A(X2)	433	RX
BZR	R2	033	RR
BZS	A	223 (Backward) 233 (Forward)	SF

Operation

If the G and L flag are both reset in the Condition Code, a branch is executed to the second operand location. If the G or L flag is set, the next sequential instruction is executed.

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Not Zero (BNZ)
Branch on Not Zero Register (BNZR)
Branch on Not Zero Short (BNZS)

Assembler Notation		Op-Code + M1	Format
BNZ	A(X2)	423	RX
BNZR	R2	023	RR
BNZS	A	203 (Backward) 213 (Forward)	SF

Operation

If the G or L flag is set in the Condition Code, a branch is executed to the second operand address. If the G and L flags are both reset, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch (Unconditional) (B)
Branch Register (Unconditional) (BR)
Branch Short (Unconditional) (BS)

Assembler Notation		Op-Code + M1	Format
B	A(X2)	430	RX
BR	R2	030	RR
BS	A	220 (Backward) 230 (Forward)	SF

Operation

A branch is unconditionally executed to the second operand address.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

Example: B

Assembler Notation		Machine Code	Comments
B	OPTIN	4300 0A00	An unconditional branch is executed.

INSTRUCTION

No Operation (NOP)
No Operation Register (NOPR)

Assembler Notation		Op-Code + M1	Format
NOP	A(X2)	420	RX
NOPR	R2	020	RR

Operation

After a short delay (instruction decode time), the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

A(X2) and R2 are meaningless and usually equal ZERO (0).

Example: NOP, NOPR

Assembler Notation		Machine Code	Comments
NOP	0	4200 0000	No Operation
NOPR	0	0200	No Operation

CHAPTER 5
FIXED POINT ARITHMETIC

Fixed Point Arithmetic instructions provide a complete set of operations for calculating addresses and indexes, for counting, and for general purpose fixed point arithmetic.

DATA FORMATS

Figure 5-1 shows the two formats for fixed point data: halfword and fullword. In each of these formats, the most significant bit (Bit 0) is the Sign bit. The remaining bits, either 15 or 31, represent the magnitude.

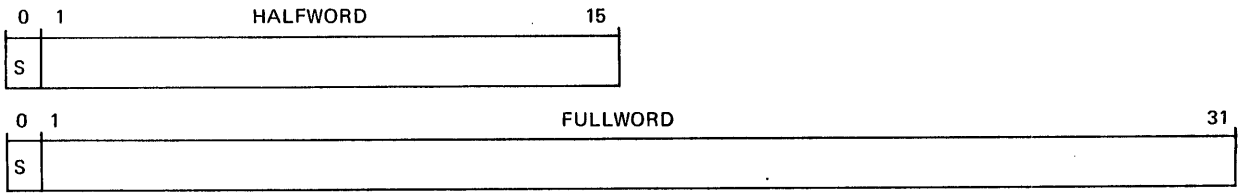


Figure 5-1. Fixed Point Data Words Format

Positive values are represented in true binary form with a Sign bit of ZERO. Negative values are represented in two's complement form with a Sign bit of ONE. To change the sign of a number, the two's complement of the number is produced as follows:

1. Change all zeros to ones, and all ones to zeros.
2. Add one.

FIXED POINT NUMBER RANGE

Fixed point numbers represent integers. Table 5-1 shows the relation between different formats along with decimal values.

TABLE 5-1. FIXED POINT FORMAT RELATIONS

FULLWORD	HALFWORD	DECIMAL
80000000 (MOST NEGATIVE)		-21474 83648
	8000 (MOST NEGATIVE)	-32768
FFFFFFFF	FFFF (LEAST NEGATIVE)	-1
00000000	0000	0
00000001	0001	1
	7FFF (MOST POSITIVE)	32767
7FFFFFFF (MOST POSITIVE)		21474 83647

CONDITION CODE

Most Fixed Point Arithmetic Instructions affect the Condition Code. (The exceptions are Multiply and Divide.) The Condition Code indicates the effect of the operation on the 16-bit result.

In fixed point Add and Subtract operations, because the arguments are represented in two's complement form, all bits, sign included, participate in forming the result. Consequently, the occurrence of a carry or borrow has no real arithmetic significance.

For example, an Add operation between a minus one (FFFF) and a plus two (0002) produces the correct result of plus one (0001) and a carry. The Condition Code is set to 1010 (C = 1 and G = 1). "Carry only" means that the complete result, which in this case would have been 10001, would not fit in 16 bits.

An overflow occurs when the result does not fit in 15 bits. Note that bit "zero" must be reserved for the sign of the result, e.g., adding one to the largest positive fixed point value produces an overflow:

```
 7FFF
+0001
=8000
```

The condition code is 0101 (V = 1 and L = 1)

The result, 8000, is logically correct, but because the sign bit is negative when the result should be positive, the overflow condition exists.

The columns of the Condition Code table show the state of the C, V, G, and L flags for the specific result.

The 'X' in the Condition Code column means that particular flag is not defined, i.e., the flag can be 0 or 1. Hence, no inference should be drawn by testing that particular flag.

FIXED POINT INSTRUCTION FORMATS

The fixed point instructions use the Register to Register (RR), the Short Form (SF), the Register and Indexed Storage (RX), and the Register and Immediate (RI) instruction formats.

FIXED POINT INSTRUCTIONS

The fixed point instructions described in this section are:

AHR	Add Halfword Register
AIS	Add Immediate Short
AH	Add Halfword
AHI	Add Halfword Immediate
AHM	Add Halfword to Memory
SHR	Subtract Halfword Register
SIS	Subtract Immediate Short
SH	Subtract Halfword
SHI	Subtract Halfword Immediate
CHR	Compare Halfword Register
CH	Compare Halfword
CHI	Compare Halfword Immediate
MH	Multiply Halfword
MHR	Multiply Halfword Register
DH	Divide Halfword
DHR	Divide Halfword Register
SLA	Shift Left Arithmetic
SLHA	Shift Left Halfword Arithmetic
SRA	Shift Right Arithmetic
SRHA	Shift Right Halfword Arithmetic
ACH	Add With Carry Halfword
ACHR	Add With Carry Halfword Register
SCH	Subtract with Carry Halfword
SCHR	Subtract with Carry Halfword Register
MHU	Multiply Halfword Unsigned
MHUR	Multiply Halfword Unsigned Register

Add Halfword (AH)
Add Halfword Register (AHR)
Add Halfword Immediate (AHI)
Add Immediate Short (AIS)

Assembler Notation		Op-Code	Format
AH	R1,A(X2)	4A	RX
AHR	R1,R2	0A	RR
AHI	R1,I(X2)	CA	RI
AIS	R1,N	26	SF

Operation

The second operand is added algebraically to the contents of the register specified by R1. The result replaces the contents of the register specified by R1.

AH	(R1) ← (R1) + [A + (X2)]
AHR	(R1) ← (R1) + (R2)
AHI	(R1) ← (R1) + I + (X2)
AIS	(R1) ← (R1) + N

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

Programming Note

In the RX format, the second operand must be located on a halfword boundary.

The second operand for the Add Immediate Short instruction is obtained by expanding the 4-bit data field, N, to a 16-bit halfword by forcing the high order bits to zero.

Example: AH

This example adds the halfword at memory location labeled LAB to the contents of Register 4.

1. Register 4 contains X'0002'
   Halfword at memory location LAB contains X'FFFF'

Assembler Notation	Comments
AH REG4,LAB	ADD (LAB) TO (REG4)

Result of AH Instruction

(REG4) = 0001
(LAB) = unchanged by this instruction
Condition Code = 1010 (C=1, G=1)

2. Register 5 contains X'FFF5'
   LAB contains X'FFF2'

Assembler Notation	Comments
AH REG5, LAB	ADD (LAB) TO (REG5)

Result of AH Instruction

(REG5) = FFE7
(LAB) = unchanged by this instruction
Condition Code = 1001 (C=1, L=1)

INSTRUCTION

Add Halfword to Memory (AHM)

Assembler Notation	Op-Code	Format
AHM R1,A(X2)	61	RX

Operation

The contents of the register specified by R1 is added algebraically to the contents of the memory location specified by the effective address of the second operand. The 16-bit result replaces the contents of the memory location specified by the effective address of the second operand. The content of the register specified by R1 is not changed.

AHM [A + (X2)] ◄——— (R1) + [A + (X2)]

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

Programming Note

The second operand must be located on a halfword boundary.

This instruction is subject to Memory Protect.

Example: **AHM**

This example adds the contents of Register 5 to the contents of memory location LAB.

1. Register 5 contains X'0002'
   Halfword in memory at LAB contains X'FFFF'

Assembler Notation	Comments
AHM REG5,LAB	ADD (REG5) TO (LAB)

Result of AHM Instruction

(REG5) = unchanged by this instruction
(LAB) = 0001
Condition Code = 1010 (C=1,G=1)

2. Register 6 contains X'FFF5'
   LAB contains X'FFF2'

Assembler Notation	Comments
AHM REG6,LAB	ADD (REG6) TO (LAB)

Result of AHM Instruction

(REG6) = unchanged by this instruction
(LAB) = FFE7
Condition Code = 1001 (C=1,L=1)

INSTRUCTIONS

Subtract Halfword (SH)
Subtract Halfword Register (SHR)
Subtract Halfword Immediate (SHI)
Subtract Immediate Short (SIS)

Assembler Notation		Op-Code	Format
SH	R1,A(X2)	4B	RX
SHR	R1,R2	0B	RR
SHI	R1,I(X2)	CB	RI
SIS	R1,N	27	SF

Operation

The halfword second operand is subtracted from the contents of the register specified by R1. The result replaces the contents of the register specified by R1. The second operand is unchanged.

SH (R1) ← (R1)–[A+(X2)]
SHR (R1) ← (R1)–(R2)
SHI (R1) ← (R1)–I–(X2)
SIS (R1) ← (R1)–N

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO
X	1	X	X	Arithmetic overflow
1	X	X	X	Borrow

Programming Note

The second operand for the Add Immediate Short instruction obtained by expanding the 4-bit data field, N, to a 16-bit halfword by forcing the high order bits to zero.

In the RX format, the second operand must be located on a halfword boundary.

Example: SH

This example subtracts the halfword at memory location LOC from the contents of register 9.

1. Register 9 contains X'3456'
   LOC contains X'FFF4'

Assembler Notation	Comments
SH REG9,LOC	Subtract contents of LOC from (REG9)

Result of SH Instruction

(REG9) = 3462
(LOC) = FFF4
Condition Code = 1010

2. Register 9 contains X'4567'
   LOC contains X'2345'

Assembler Notation	Comments
SH REG9,LOC	Subtract contents of LOC from (REG9)

Result of SH Instruction

(REG9) = 2222
(LOC) = 2345
Condition Code = 0010

Add with Carry Halfword (ACH)
Add with Carry Halfword Register (ACHR)

Assembler Notation		Op-Code	Format
ACH	R1,A(X2)	4E	RX
ACHR	R1,R2	0E	RR

Operation

The 16-bit second operand and the carry of the previous operation are added algebraically to the contents of the register specified by R1. The result replaces the contents of the register specified by R1 and is reflected by the setting of the Condition Code. The second operand is unchanged.

ACH (R1) ← (R1) + [A + (X2)] + C
ACHR (R1) ← (R1) + (R2) + C

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

Programming Note

Multiple precision addition operations require a carry forward from the least significant operands to the most significant. To accomplish this, the locations containing the least significant portions of the two operands are summed, using the Add Halfword instruction. A carry forward, if it occurs, is retained in the carry bit of the Condition Code. The locations containing the next least significant portions of the two operands are then summed, using the Add with Carry instruction. The carry bit contained in the Condition Code, set from the previous operation, participates in this sum. The carry bit is then set to reflect the new result. The Add with Carry instruction is used on succeeding pairs of operands until the most significant operands of the multiple precision words have been summed. The resulting Condition Code is valid for testing the multiple precision word.

Subtract with Carry Halfword (SCH)
Subtract with Carry Halfword Register (SCHR)

Assembler Notation		Op-Code	Format
SCH	R1,A(X2)	4F	RX
SCHR	R1,R2	0F	RR

Operation

The 16-bit second operand and the borrow from the previous operation are subtracted from the contents of the register specified by R1. The result replaces the contents of the register specified by R1 and is reflected by the setting of the Condition Code. The second operand is unchanged.

SCH (R1) ← (R1) - [A + (X2)] - C
SCHR (R1) ← (R1) - (R2) - C

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO
X	1	X	X	Arithmetic overflow
1	0	X	X	Carry (Borrow)

Programming Note

Multiple precision subtraction operations require a carry forward from the least significant operands to the most significant. To accomplish this, the locations containing the least significant portions of the two operands are subtracted, using the Subtract Halfword instruction. A carry forward, if it occurs, is retained in the carry bit of the Condition Code. The locations containing the next least significant portions of the two operands are then subtracted, using the Subtract with Carry instruction. The carry bit contained in the Condition Code, set from the previous operation, participates in this operation. The carry bit is then set to reflect the new result. The Subtract with Carry instruction is used on succeeding pairs of operands until the most significant operands of the multiple precision words have been subtracted. The resulting Condition Code is valid for testing the multiple precision word.

Compare Halfword (CH)
Compare Halfword Register (CHR)
Compare Halfword Immediate (CHI)

Assembler Notation		Op-Code	Format
CH	R1,A(X2)	49	RX
CHR	R1,R2	05	RR
CHI	R1,I(X2)	C9	RI

Operation

The Halfword second operand is compared algebraically with the first operand, the contents of the register specified by R1. The result is indicated by the Condition Code setting. Neither operand is changed.

Condition Code

C	V	G	L	
0	X	0	0	First operand is equal to second operand
1	X	0	1	First operand is less than second operand
0	X	1	0	First operand is greater than second operand

Programming Note

In the RX format, the second operand must be located on a halfword boundary.

The state of the V flag is undefined.

Example: CH

This example compares the contents of REG8 to the halfword at LAB.

Register 8 contains X'7891'
Halfword at LAB contains X'3123'

Assembler Notation	Comments
CH REG8,LAB	Compare (REG8) to (LAB)

Result of CH Instruction

(REG8) = unchanged by this instruction
(LAB) = unchanged by this instruction
Condition Code = 0010 (G=1)

Multiply Halfword (MH)
Multiply Halfword Register (MHR)

Assembler Notation		Op-Code	Format
MH	R1,A(X2)	4C	RX
MHR	R1,R2	0C	RR

Operation

The signed (halfword) first operand, contained in the register specified by R1+1, is multiplied by the signed (halfword) second operand. The 32-bit result replaces the contents of the registers specified by R1 and R1+1.

MH (R1,R1 + 1) ⟵ (R1 + 1) * [A + (X2)]
MHR (R1,R1 + 1) ⟵ (R1 + 1) * (R2)

Condition Code

Unchanged

Programming Note

After multiplication, the most significant 15 bits with sign bit are contained in R1. The least significant 16 bits are contained in R1+1. The sign of the result is determined by the rules of algebra.

In the RX format, the second operand must be located on a halfword boundary.

The R1 field of these instructions must specify an even numbered register.

Example: MH

This example multiplies the halfword contents of Register 9 by the halfword in memory location LAB.

Register 9 contains X'0045'
Halfword at memory location LAB contains X'8674'

Assembler Notation	Comments
MH REG8,LAB	Multiply (REG9) by (LAB)

Result of MH Instruction

(REG8) = FFDF (REG9) = 3D44
(LAB) = unchanged by this instruction
Condition Code = unchanged by this instruction

Multiply Halfword Unsigned (MHU)
Multiply Halfword Unsigned Register (MHUR)

Assembler Notation		Op-Code	Format
MHU	R1,A(X2)	DC	RX
MHUR	R1,R2	9C	RR

Operation

The 16-bit second operand is multiplied by the contents of the register specified by R1 + 1. All 16 bits of both operands are considered magnitude. The resulting 32-bit product is contained in the registers specified by R1 and R1 + 1.

MHU (R1,R1 + 1) ◄—— (R1 + 1)*[A + (X2)]
MHUR (R1,R1 + 1) ◄—— (R1 + 1)*(R2)

Condition Code

Unchanged

Programming Note

The R1 field must specify an even numbered register.

In the RR format, R2 may specify any register.

This instruction is most useful in applications requiring multiple precision multiply capability.

Divide Halfword (DH)
Divide Halfword Register (DHR)

Assembler Notation		Op-Code	Format
DH	R1,A(X2)	4D	RX
DHR	R1,R2	0D	RR

Operation

The 32-bit signed dividend contained in the register specified by R1 and R1+1 is divided by the 16-bit signed second operand (divisor). The 16-bit signed remainder is stored in the register specified by R1. The 16-bit signed quotient is stored in the register specified by R1+1.

DH	(R1 + 1) ←	(R1,R1 + 1)/[A + (X2)]
	(R1) ←	REMAINDER
DHR	(R1 + 1) ←	(R1,R1 + 1)/(R2)
	(R1) ←	REMAINDER

Condition Code

Unchanged

Programming Note

The R1 field of these instructions must specify an even-numbered register. Otherwise, the results are undefined.

In the RX formats, the second operand must be located on a halfword boundary.

If the divisor is equal to zero, the instruction is not executed, the operand registers are unchanged, and the Divide Fault Interrupt is taken, if enabled by bit 3 of the current program status word. If the interrupt is not enabled, the next sequential instruction is executed.

If the value of the quotient is greater than X'7FFF' or less than (more negative than) X'8000', quotient overflow is said to occur.

If quotient overflow occurs, the operand registers are not changed, and the Divide Fault Interrupt is taken, if enabled by bit 3 of the current program status word. If the interrupt is not enabled, the next sequential instruction is executed.

The sign of the quotient is determined by rules of algebra.

The sign of the remainder is the same as the sign of the dividend.

Example: DH

In this example, the contents of Registers 8 and 9 are divided by the halfword contents of memory location LOC.

1. Register 8 contains X'0000' = Dividend
   Register 9 contains X'0054' = Divisor
   LOC contains X'0008'

Assembler Notation	Comments
DH REG8,LOC	Divide (REG8, REG9) by (LOC)

Result of DH Instruction

(REG8) = 0004 = Remainder
(REG9) = 000A = Quotient
(LOC) = 0008
Condition Code = unchanged by this instruction

2. Register 8 contains X'0000' = Dividend
   Register 9 contains X'1234' = Divisor
   LOC contains X'0000'

Assembler Notation	Comments
DH REG8,LOC	Divide (REG8, REG9) by (LOC)

Result of DH Instruction

Division by zero causes arithmetic fault to be taken if bit 3 of PSW is enabled.

Operands and Condition Code remain unchanged by this instruction.

3. Register 8 contains X'FFFF'
   Register 9 contains X'8002' = Dividend
   LOC contains X'0001'

Assembler Notation	Comments
DH REG8,LOC	Divide (REG8, REG9) by (LOC)

Result of DH Instruction

Quotient overflow causes arithmetic fault to be taken if bit 3 of PSW is enabled.

Operands and Condition Code remain unchanged by this instruction.

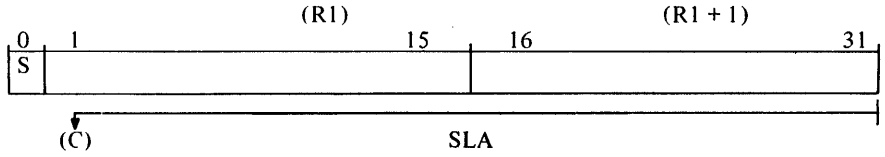
INSTRUCTION

Shift Left Arithmetic (SLA)

Assembler Notation		Op-Code	Format
SLA	R1,I(X2)	EF	RI

Operation

In this instruction, the register specified by R1 and the register implied by the value R1+1 are linked together to form a fullword operand. Bit 0 of the register specified by R1 is the Sign bit. Bits 1:15 of the register specified by R1 and Bits 0:15 of the register specified by R1+1 are shifted left the number of binary places specified by the second operand. The Sign bit is not shifted. Bits shifted out of Position 1 of the first register are shifted into the carry flag of the PSW and then lost. Zeros are moved into Position 15 of the second register.



Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO

Programming Note

R1 must specify an even-numbered register.

The shift count is specified by the least significant five bits of the second operand.

A shift of zero places causes the Condition Code to be set in accordance with the value contained in the register specified by R1. The state of the C flag is undefined in this case.

The state of the C flag indicates the state of the last bit shifted.

Example: SLA

This example shifts the bits in Registers 4 and 5 left by the number specified by the second operand.

Register 4 contains X'8047'
Register 5 contains X'ABCD'

Assembler Notation		Comments
SLA	REG4,4	Shift Left 4 Places

Result of SLA Instruction

(REG4) = X'847A', (REG5) = X'BCD0'
Condition Code = 0001 (L=1)

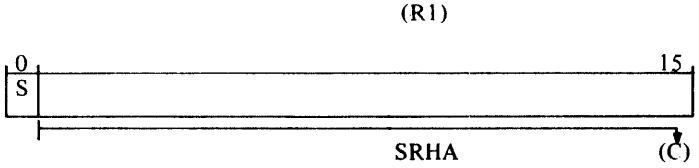
INSTRUCTION

Shift Right Halfword Arithmetic (SRHA)

Assembler Notation		Op-Code	Format
SRHA	R1,I(X2)	CE	RI

Operation

Bits 1:15 of the register specified by R1 are shifted right the number of places specified by the second operand. Bit 0 of the register, the halfword Sign bit, remains unchanged and is propagated right the number of positions specified by the second operand. Bits shifted out of Position 15 are shifted through the carry flag and lost. The last bit shifted remains in the carry flag.



Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO

Programming Notes

The shift count is specified by the low order four bits of the second operand.

The state of the C flag indicates the state of the last bit shifted.

If the second operand specified a shift of zero places, the Condition Code is set in accordance with the value contained in the register. The state of the C flag is undefined.

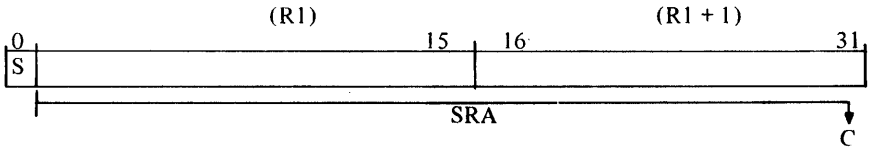
INSTRUCTION

Shift Right Arithmetic (SRA)

Assembler Notation		Op-Code	Format
SRA	R1,I(X2)	EE	RI

Operation

In this instruction, the register specified by R1 and the register implied by the value R1+1 are linked together forming a fullword operand. Bit 0 of the register specified by R1 is the Sign bit. Bits 1:15 of the register specified by R1 and Bits 0:15 of the register specified by R1+1 are shifted right the number of binary places specified by the second operand. The Sign bit remains unchanged and is propagated right as many positions as specified by the second operand. Bits shifted out of Position 15 of the second register are shifted into the carry flag of the PSW, and then lost.



Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO

Programming Note

R1 must specify an even-numbered register.

The state of the C flag indicates the state of the last bit shifted.

The shift count is specified by the least significant five bits of the second operand.

A shift of zero places causes the Condition Code to be set in accordance with the value contained in the registers. The C flag is undefined.

Example: SRA

This example shifts the contents of Registers 8 and 9 right the number of places specified by the second operand.

Register 8 contains X'8ABC'
Register 9 contains X'4256'

Assembler Notation	Comments
SRA REG8,8	Shift (REG9) right 8 bits

Result of SRA Instruction

(REG8) = FF8A (REG9) = BC42
Condition Code = 0001 (L=1)

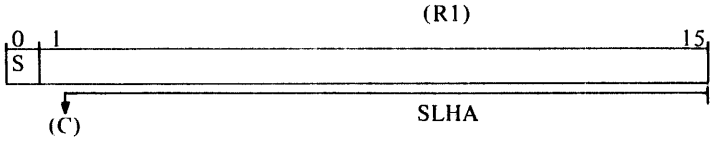
INSTRUCTION

Shift Left Halfword Arithmetic (SLHA)

Assembler Notation		Op-Code	Format
SLHA	R1,I(X2)	CF	RI

Operation

Bits 1:15 of the register specified by R1 are shifted left the number of places specified by the second operand. Bit 0 of the register, the Sign bit, remains unchanged. Bits shifted out of Position 1 are shifted through the carry flag and then lost. The last bit shifted remains in the carry flag. Zeros are shifted into Position 15.



Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO

Programming Note

The state of the C flag indicates the state of the last bit shifted.

The shift count is specified by the least significant four bits of the second operand.

A shift of zero places causes the state of the Condition Code to be set in accordance with the value contained in the register. The C flag is undefined.

CHAPTER 6
FLOATING POINT ARITHMETIC

Floating Point Arithmetic instructions provide a means for rapid manipulation of scientific data expressed as floating point numbers. Single Precision as well as Double Precision Floating Point Instructions are described in this chapter. The comprehensive set of instructions includes load and store floating point numbers; add, subtract, multiply, divide and compare two floating point numbers; convert fixed print to floating point and vice versa.

INTRODUCTION

Floating point is a means of respresenting a quantity in any numbering system. Consider a decimal number (base = 10), 123 which can be represented in the following forms:

$$\begin{aligned} 123.0 &\times 10^0 \\ 1.23 &\times 10^2 \\ 0.123 &\times 10^3 \\ 0.0123 &\times 10^4 \end{aligned}$$

Note that in this example, the decimal point moved. Hence we have a floating point. In actual floating point representation the significant digits are always fractional and are collectively referred to as fraction. The power to which the base number is raised is called the exponent. For example, in the number ".45678 x 10^2", 45678 is the fraction and 2 is the exponent. Both the fraction and the exponent may be signed. If we have a floating point representation as,

(sign of fraction) (exponent) (fraction)

then the following representation applies:

Number		Floating point		
+32.94	= +.3294 x 10^2	+	+2	3294
-23760000.0	= -.2376 x 10^8	-	+8	2376
+0.000059	= +.59 x 10^{-4}	+	-4	59
-0.0000000092073	= .92073 x 10^{-9}	-	-9	92073

The convenience with which extremely large or small numbers can be expressed in floating point makes it ideally suitable for scientific computation. Note the compactness in the above examples.

The Perkin-Elmer floating point representation is similar to the above representation. The differences are as follows:

Hexadecimal, instead of decimal, numbering system is used.
Physical size of the number and hence the magnitude and the precision is limited.

The single precision floating point number fields are shown in Figure 6-1.

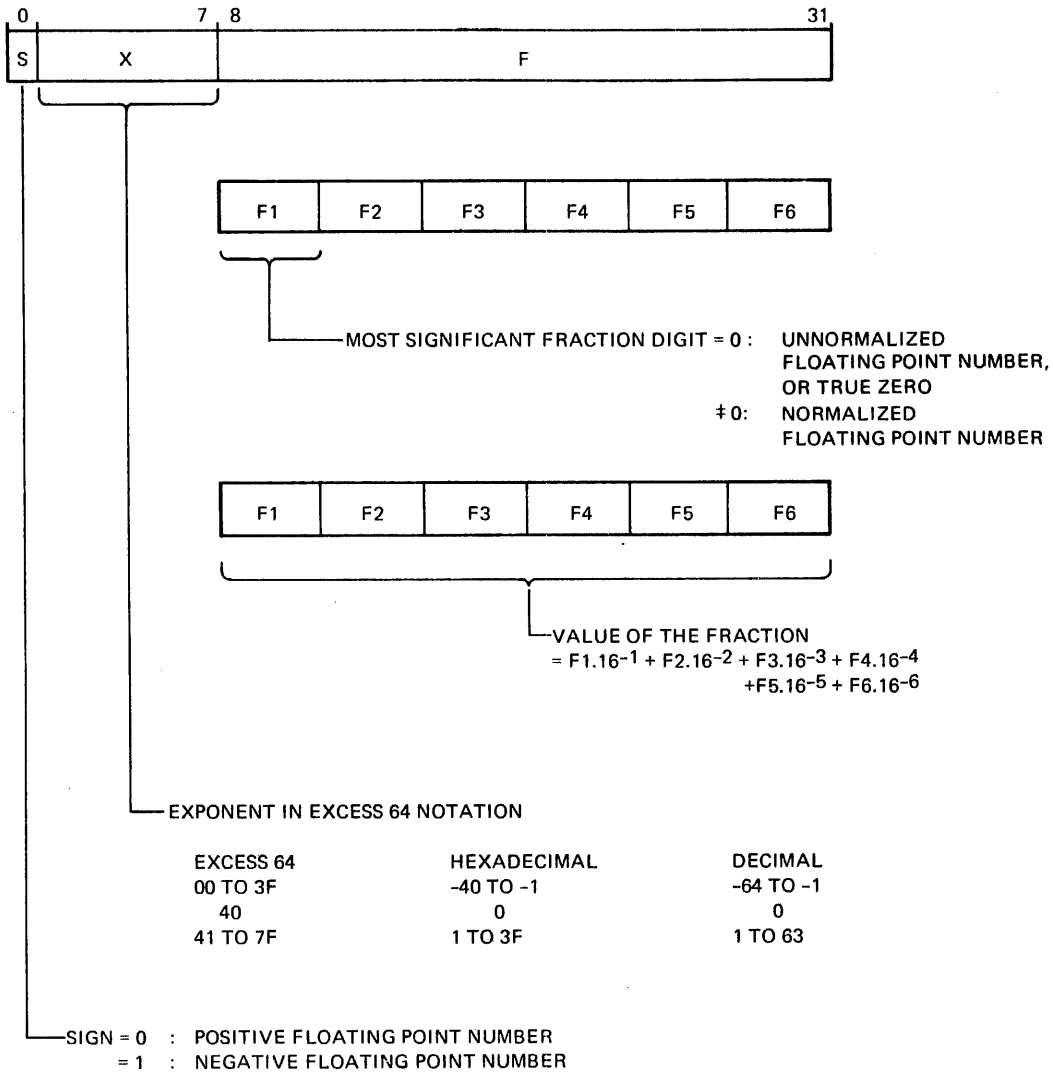
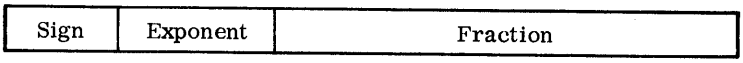


Figure 6-1. Single Precision Floating Point Number Fields

FLOATING-POINT NUMBER

A floating point number is represented in the following form:



Sign — The most significant bit of a floating point number is a sign bit. The sign bit is zero for positive numbers and one for negative numbers. The floating point value of zero always has a positive sign.

Exponent — The 7-bit fields, bits 1:7, is designated as the exponent field. The exponent field contains the true value of the exponent plus X'40' (decimal 64). This helps to represent very small magnitudes between 0 and 1. The exponent is said to be expressed in excess 64 notation. Some of the exponent values are as follows:

Exponent in Excess 64 notation	True exponent in hexadecimal	True exponent in decimal	Multiply fraction by
00	-40	-64	16^{-64}
3F	-1	-1	16^{-1}
40	0	0	1
41	1	1	16
7F	3F	63	16^{+63}

The exponent field for true zero is always 00.

Fraction — The fraction field is six hexadecimal digits for single precision floating point numbers (thus limiting the precision) and 14 hexadecimal digits for double precision floating point numbers. As in any other fraction, the floating point fraction is expressed with most precision when the most significant digit (not necessarily the most significant bit) is non-zero. The floating point number with such a fraction is called a normalized floating point number. Normalized numbers are always used to obtain maximum possible precision. For hexadecimal fraction conversion, refer to Appendix 5.

Examples: The following examples illustrate the sign, exponent and fraction concept of a floating point number.

Numbers in Hex integer-fraction notation	Sign-exponent-fraction shown for clarity S	E	F	Single Precision Floating point numbers
+1.3A25678	0	41	13A25678	4113A256
-6.89F2C	1	41	689F2C	C1689F2C
+1A.C39D21	0	42	1AC39021	421AC39D
-3C1DF.82A3	1	45	3C1DF82A3	C53C1DF8
+ABCDEF12.9AC	0	48	ABCDEF129AC	48ABCDEF
+0.0032A9CF2	0	3E	32A9CF2	3E32A9CF
-0.000002C7B5	1	3B	2C7B5	BB2C7B50

Refer to Appendix 5 for examples of similar conversion to double precision floating point numbers.

Floating Point Number Range

The range of magnitude (M) of a normalized floating point number is as follows.

Single precision:	$16^{-65} \leq M \leq (1 - 16^{-6}) * 16^{63}$
Double precision:	$16^{-65} \leq M \leq (1 - 16^{-14}) * 16^{63}$
Approximately for both:	$5.4 * 10^{-79} < M < 7.2 * 10^{75}$

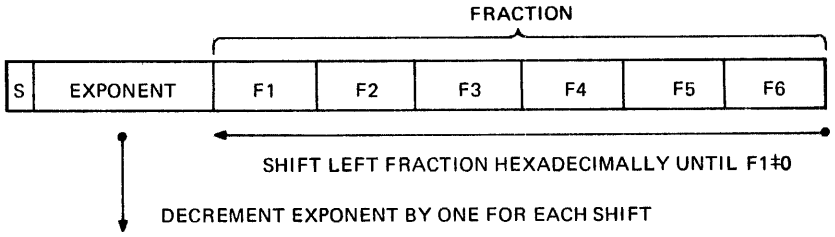
Table 6-1 shows the single precision floating point range in relation to the fixed point range along with the decimal values.

TABLE 6-1 FLOATING/FIXED POINT RANGES

	Floating Point numbers	Fixed Point integer	Decimal numbers
(most negative)	FFFF FFFF		$-7.2 * 10^{75}$
	C880 0000	8000 0000 (most negative)	-2 147 483 648
	C110 0000	FFFF FFFF (least negative)	-1
(least negative)	8010 0000		$-5.4 * 10^{-79}$
(true zero)	0000 0000	0000 0000	0
(least positive)	0010 0000		$+5.4 * 10^{-79}$
	4110 0000	0000 0001 (least positive)	+1
	487F FFFF	7FFF FFFF (most positive)	+2 147 483 647
(most positive)	7FFF FFFF		$+7.2 * 10^{75}$

Normalization

Normalization is a process of making non-zero the most significant digit (F1) of the fraction of a floating point number. In the normalization process, the floating point fraction is shifted left hexadecimally (i.e., four bits at a time), and its exponent is decremented by one for each hexadecimal shift until the most significant digit (not necessarily the most significant bit) of the fraction is non-zero.



Except for LE, LER, LD, LDR instructions, all the floating point operations assume and require normalized operands for consistent results. The LE, LER, LD and LDR instructions normalize an unnormalized operand.

Example:

	Operands	After normalization	
1.	42012345	41123450	
2.	21000ABC	1EABC000	
3.	C900FE12	C7FE1200	
4.	6C000000	00000000	(true zero)
5.	82000A67	00000000	(exponent underflow)

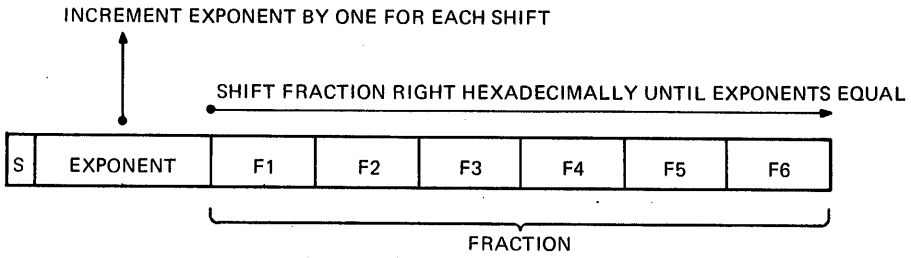
In example 4, the fraction of the operand is zero. During the normalization process, such a fraction is detected and the floating point number is set to true zero.

In example 5, the exponent of the operand is very small. During the normalization process, the exponent is decremented from 00 to 7F. Such a transition results in exponent underflow and the floating point number is set to true zero.

In floating point operations, assuming that the operands are normalized, normalized results are always produced.

Equalization

Equalization is a process of making equal the exponents of two floating point numbers. The fraction of the floating point number with the smaller exponent is shifted right hexadecimally, i.e., four bits at a time, and its exponent is incremented by one for each hexadecimal shift until the two exponents are equal.



During floating point addition and subtraction the two floating point operands are equalized.

Example:

	Floating point operands	After equalization
1.	43123456 3F789ABC	43123456 43000078
2.	C7FE1234 4956789A	C900FE12 4956789A

In this example, normalized floating point numbers are shown because addition and subtraction require normalization. Note that if the exponents differ by 6 or more the significance of the lower exponent floating point number is lost in the process of equalization.

True Zero

A floating point number is said to be true zero when the exponent and the fraction fields are all zeroes. In other words, all data bits must be zero. A value of zero always has a positive sign. In general, zero values participate as normal operands in all floating point operations.

A true zero may be used as an operand or may result from an arithmetic operation that caused an exponent underflow, in which case the entire number is forced to true zero. Secondly, if an arithmetic operation produces a result whose fraction digits are all zeroes (sometimes referred to as loss of significance), the entire number is forced to true zero.

Examples:

Numbers	Operation	Result	
030000AB	Normalize	0000 0000	exponent underflow
41ABCDEF 41ABCDEF	Subtract	0000 0000	loss of significance

Exponent Overflow

In floating point operations, exponent overflow may occur. Exponent overflow occurs when a resulting exponent is greater than +63. If overflow occurs, the exponent and fraction bits of the result are set to all 1s, the largest possible magnitude and therefore the closest possible answer. The sign of the result is not affected by the overflow. Figure 6-2 illustrates exponent overflow using a line representation of numbers.

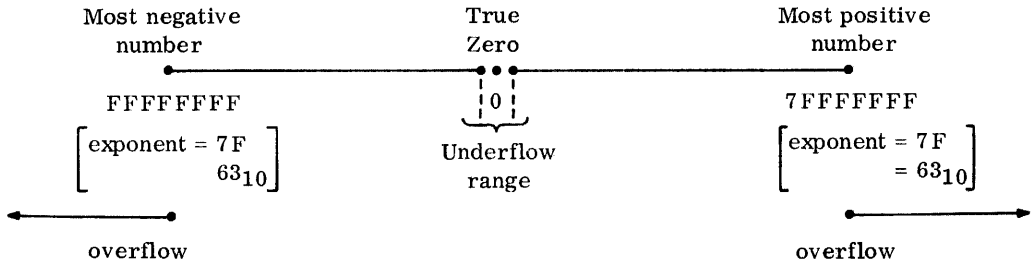


Figure 6-2. Exponent Overflow

If overflow occurs, the V flag in the Condition Code is set, and an arithmetic fault interrupt is taken, if enabled by the current PSW.

Exponent Underflow

The normalization process, during a floating point operation, may produce an exponent underflow. Exponent underflow occurs when a result exponent would be less than -64. If underflow occurs, the entire result is set to true zero, the closest possible answer. Figure 6-3 illustrates exponent underflow using a line representation of numbers.

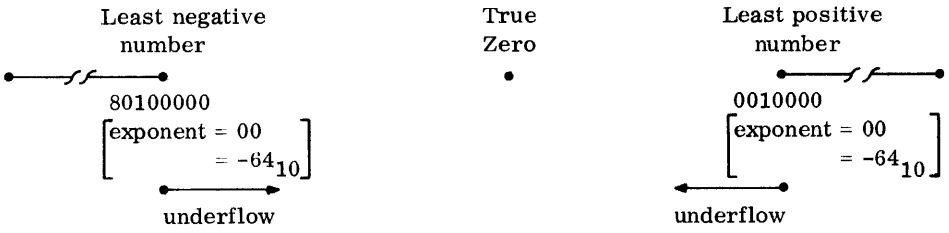
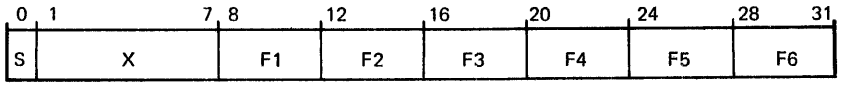


Figure 6-3. Exponent Underflow

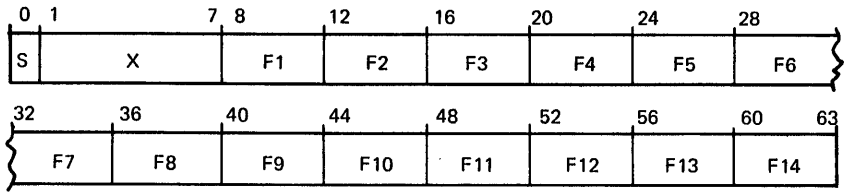
If underflow occurs, the V flag in the Condition Code is set, and an arithmetic fault interrupt is taken, if enabled by the current PSW.

Data Formats

Floating point numbers occur in one of two formats, single precision and double precision. The single precision format requires a fullword (32 bits) in one of the eight single precision floating point registers or on a fullword address boundary, in memory. The sign (s), exponent (x) and fraction (consisting of digits F1, F2, F3, F4, F5 and F6) fields are designated as follows:



The double precision format requires a doubleword (64 bits) in one of the eight double precision floating point registers or on a doubleword address boundary in memory. The sign (S), exponent (X) and fraction (consisting of digits F1 through F14) fields are designated as follows:



The value of a single (and similarly double) precision floating point number can be expressed as follows:

$$\text{sign}(F1.16^{-1} + F2.16^{-2} + F3.16^{-3} + F4.16^{-4} + F5.16^{-5} + F6.16^{-6})\ 16^{X-x'40'}$$

Guard Digit and Rounding

A guard digit is an extra hexadecimal digit provided to the right of the least significant fraction digit of a floating point number. Only single precision floating point numbers can have a guard digit. The guard digit is produced and used during the processing of intermediate results of a floating point operation. The guard digit does not appear in the final result. However, the guard digit helps rounding the final result, thus increasing the precision slightly. In the absence of a guard digit, as is the case in double precision floating point numbers, the final result is simply truncated.

NOTE

In Processors which do not have the double precision floating point option, there is no guard digit for single precision floating point numbers. Hence the results are truncated, not rounded.

A guard digit is produced during the equalization phase of an Add or Subtract single precision floating point operation. Then the operation is performed to obtain an intermediate result. The guard digit participates in the operation. If the guard digit of the intermediate result is 0 through 7, no rounding is done. If it is 8 through F, one (1) is added to the fraction of the intermediate result to obtain the final result fraction, unless such an addition produces a carry into the exponent field. The following example illustrates the rounding procedure.

Operands	After equalization	Guard digit	
42ABCD12	42ABCD12	[0]	
+ 416789AB	+ 4206789A	[B]	
	42B245AC	[B]	intermediate result
	+ 1		
	42B245AD		final result

A guard digit is also produced during the Multiply and Divide single precision floating point operations. The intermediate product or the quotient is rounded as shown here to obtain the final result.

Conversion from Decimal

The process of converting a decimal number into the excess 64 notation used internally by the Processor involves the following steps:

1. Separate the decimal integer from the decimal fraction:

   $182.375_{10} = (182 + .375)_{10}$

2. Convert each part to hexadecimal by referring to the Integer conversion table and the Fraction conversion table in Appendix 5.

   $182_{10} = B6_{16}$ $\qquad .375_{10} = .6_{16}$

3. Combine the hexadecimal integer and fraction:

   $B6.6_{16} = (B6.6 \times 16^{0})_{16}$

4. Shift the radix point:

   $(B6.6 \times 16^{0})_{16} = (.B66 \times 16^{2})_{16}$

5. Add 64, (X'40'), to the exponent

   $40_{16} + 2_{16} = 42_{16}$

6. Convert the exponent field and fraction to binary allowing 1 bit for the sign, 7 bits for the exponent field, and 24 or 56 bits for the fraction.

   42B66 = 0100 0010 1011 0110 0110 0000 0000 0000

CONDITION CODE

Following floating point operations, including load, the Condition Code indicates the result of the operation.

FLOATING POINT INSTRUCTION FORMATS

The Floating Point instructions use the Register to Register (RR), and the Register and Indexed Storage (RX) instruction formats. In all of the RR formats, except for Fix and Float, the R1 and the R2 fields specify one of the floating point registers. There are eight single precision floating point registers, and eight double precision floating point registers numbered 0, 2, 4, 6, 8, 10, 12, and 14. Except FXR and FXDR instructions, the R1 field always specifies a floating point register.

FLOATING POINT INSTRUCTIONS

The floating point arithmetic operations, excluding loads and stores, require normalized operands to insure correct results. If the operands are not normalized, the results of these operations are undefined. Floating point results are normalized. The Floating Point Load instruction normalizes floating point data extracted from memory.

The single precision floating point instructions described in this section are:

LE	Load Floating Point	CE	Compare Floating Point
LER	Load Floating Point Register	CER	Compare Floating Point Register
* LME	Load Floating Point Multiple	ME	Multiply Floating Point
STE	Store Floating Point	MER	Multiply Floating Point Register
*STME	Store Floating Point Multiple	DE	Divide Floating Point
AE	Add Floating Point	DER	Divide Floating Point Register
AER	Add Floating Point Register	* FXR	Fix Register
SE	Subtract Floating Point	* FLR	Float Register
SER	Subtract Floating Point Register		

* Not Available In All Implementations

The double precision floating point instructions described in this section are:

LD	Load DPFP	CD	Compare DPFP
LDR	Load Register DPFP	CDR	Compare Register DPFP
LMD	Load Multiple DPFP	MD	Multiply DPFP
STD	Store DPFP	MDR	Multiply Register DPFP
STMD	Store Multiple DPFP	DD	Divide DPFP
AD	Add DPFP	DDR	Divide Register DPFP
ADR	Add Register DPFP	FXDR	Fix Register DPFP
SD	Subtract DPFP	FLDR	Float Register DPFP
SDR	Subtract Register DPFP		

Double precision Floating point is not available in all implementations.

Load Floating Point (LE)
Load Floating Point Register (LER)

Assembler Notation		Op-Code	Format
LE	R1, A(X2)	68	RX
LER	R1, R2	28	RR

Operation

The floating point second operand is normalized, if necessary, and placed in the floating point register specified by R1.

LER: (R1) ← (R2)
LE: (R1) ← [A+ (X2)]

Condition Code

C	V	G	L	
0	0	0	0	Floating point value is ZERO
0	0	0	1	Floating point value is less than ZERO
0	0	1	0	Floating point value is greater than ZERO
0	1	0	0	Exponent underflow

Programming Note

If the fraction is zero, the result is forced to X'0000 0000'

Normalization may produce exponent underflow. In this event, the result is forced to zero, X'0000 0000', the V flag in the Condition Code is set, the G and L flags are reset and, if enabled by bit 5 of the current PSW, the arithmetic fault interrupt is taken.

In the RX formats, the second operand must be located on a fullword boundary.

Example: LE

This example normalizes the fullword at memory location LOC and places it in Floating Point Register 8.

Floating Point Register 8 = undefined
LOC = X'4200 1000'

Assembler Notation	Comments
LE REG8, LOC	Normalize contents of LOC

Result of LE Instruction

(Floating Point Register 8) = 4010 0000
(LOC) = unchanged by this instruction
Condition Code = 0010

INSTRUCTION

Load Floating Point Multiple (LME)

Assembler Notation		Op-Code	Format
LME	R1, A (X2)	7?	RX

Operation

Successive floating point registers, starting with the register specified by R1, are loaded from successive memory locations starting with the address of the second operand. The process stops when Floating Point Register 14 has been loaded.

1. (R1)←[A + (X2)]
2. R1:X'E'
   if R1 = X'E', the instruction is finished
   if R1 ≠ X'E', then:
3. R1←——R1+2
4. A←——A+4, return to step 1

Condition Code

Unchanged

Programming Note

The second operand must be located on a fullword boundary.

INSTRUCTION

Store Floating Point (STE)

Assembler Notation		Op-Code	Format
STE	R1, A (X2)	60	RX

Operation

The floating point first operand, contained in the floating point register specified by R1, is placed in the memory location specified by the second operand address. The first operand is unchanged.

$[A+(X2)] \leftarrow (R1)$

Condition Code

Unchanged

Programming Note

The second operand must be located on a fullword boundary.

This instruction is subject to memory protect.

INSTRUCTION

Store Floating Point Multiple (STME)

Assembler Notation		Op-Code	Format
STME	R1, A(X2)	71	RX

Operation

The contents of successive floating point registers, starting with the register specified by R1, are stored in successive memory locations, starting with the address of the second operand. The operation stops when the contents of Floating Point Register 14 have been stored.

1. [A+(X2)] ⟵ (R1)
2. R1:X'E'
   if R1 = X'E', the instruction is finished
   if R1 ≠ X'E', then:
3. R1 ⟵ R1+2
4. A ⟵ A+4, return to step 1.

Condition Code

Unchanged

Programming Note

The second operand must be located on a fullword boundary.

This instruction is subject to memory protect.

Add Floating Point (AE)
Add Floating Point Register (AER)

Assembler Notation		Op-Code	Format
AE	R1,A(X2)	6A	RX
AER	R1,R2	2A	RR

Operation

The exponents of the two operands are compared. If the exponents differ, the fraction with the smaller exponent is shifted right hexadecimally (four bits at a time), and its exponent is incremented by one for each hexadecimal shift until the two exponents are equal. The hexadecimal digits (of four bits each) are shifted through the guard digit. The guard digit contains the last shifted hexadecimal digit. If no shift occurs it is zero. The fractions are then added algebraically.

If the addition of fractions produces a carry, the exponent of the result is incremented by one and the fraction of the result is shifted right one hexadecimal digit. The carry bit is shifted back into the most significant hexadecimal digit of the fraction, producing a normalized result. This result replaces the contents of the register specified by R1.

If the addition of fractions does not produce a carry, the result is normalized, if necessary, and replaces the contents of the register specified by R1.

AER: $(R1) \leftarrow (R1) + (R2)$
AE: $(R1) \leftarrow (R1) + [A + (X2)]$

Condition Code

C	V	G	L	
X	0	0	0	Floating point result is ZERO
X	0	0	1	Floating point result is less than ZERO
X	0	1	0	Floating point result is greater than ZERO
X	1	0	1	Exponent overflow, Result is negative
X	1	1	0	Exponent overflow, Result is positive
X	1	0	0	Exponent underflow

Programming Note

When the addition of the fractions produces a carry, incrementing the exponent of the result by one may produce exponent overflow. In this case, the result is forced to the maximum value, $\pm$ X'7FFF FFFF', the V flag, along with the G or L flag is set in the Condition Code and, if enabled by bit 5 of the current PSW, the arithmetic fault interrupt is taken.

Normalization of the result may produce exponent underflow. In this case, the result is forced to zero, X'0000 0000'. The V flag is set in the Condition Code. The G and the L flags are always reset, and if enabled by bit 5 of the current PSW, the arithmetic fault interrupt is taken.

If the guard digit is 0:7, the result is not rounded. If the guard digit is 8:F, the result is rounded by adding 1 to the fraction of the result unless rounding produces a carry into the exponent field.

In the RX formats, the second operand must be located on a fullword boundary.

Example: SE

This example subtracts the contents of LOC from the contents of Floating Point Register 8 and places the result in Floating Point Register 8.

Floating Point Register 8 contains X'7FEF FFFF'
LOC contains X'7A10 0000'

Assembler Notation	Comments
SE REG8, LOC	Subtract (LOC) from (REG8)

Result of SE Instruction

(Floating Point Register 8)	=	7FEF FFFE
(LOC)	=	unchanged by this instruction
Condition Code	=	0010

Subtract Floating Point (SE)
Subtract Floating Point Register (SER)

Assembler Notation		Op-Code	Format
SE	R1, A(X2)	6B	RX
SER	R1, R2	2B	RR

Operation

The exponents of the two operands are compared. If the exponents differ, the fraction with the smaller exponent is shifted right hexadecimally (four bits at a time), and its exponent is incremented by one for each hexadecimal shift until the two exponents are equal. The hexadecimal digits (of four bits each) are shifted through the guard digit. The guard digit contains the last shifted hexadecimal digit. If no shift occurs it is zero. The second operand fraction is then subtracted algebraically from the first operand fraction.

If the subtraction of fractions produces a borrow, the exponent of the result is incremented by one and the fraction of the result is shifted right one hexadecimal digit. The borrow bit is shifted back into the most significant hexadecimal digit of the fraction, producing a normalized result. This result replaces the contents of the register specified by R1.

If the subtraction of fractions does not produce a borrow, the result is normalized. The normalized result replaces the contents of the register specified by R1.

SER: (R1)←—(R1) - (R2)
SE: (R1)←—(R1) - [A + (X2)]

Condition Code

C	V	G	L	
X	0	0	0	Floating point result is ZERO
X	0	0	1	Floating point result is less than ZERO
X	0	1	0	Floating point result is greater than ZERO
X	1	0	1	Exponent overflow, Result is negative
X	1	1	0	Exponent overflow, Result is positive
X	1	0	0	Exponent underflow

Programming Note

When the subtraction of the fractions produces a borrow, incrementing the exponent of the result by one may produce exponent overflow. In this case, the result is forced to the maximum value, ±X'7FFF FFFF', the V flag, along with the G or L flag is set in the Condition Code and, if enabled by bit 5 of the current PSW, the arithmetic fault interrupt is taken.

Normalization of the result may produce exponent underflow. In this case, the result is forced to zero, X'0000 0000'. The V flag is set in the Condition Code. The G and the L flags are always reset and, if enabled by bit 5 of the current PSW, the arithmetic fault interrupt is taken.

The shifted hexadecimal digits (if any) participate in subtraction and produce a guard digit. If the guard digit is 0:7, the result is not rounded. If the guard digit is 8:F, the result is rounded by adding 1 to the fraction of the result unless rounding produces a carry into the exponent field.

In the RX formats, the second operand must be located on a fullword boundary.

Example: AE

This example adds the contents of LOC to the contents of the Floating Point Register 8 and places the answer in Floating Point Register 8.

Floating Point Register 8 contains X'7EFF FFFF'
LOC contains X'7EFF FFFF'

Assembler Notation	Comments
AE REG8, LOC	ADD (REG 8) to (LOC)

Result of AE Instruction

(Floating Point Register 8)	=	7F1F FFFF
(LOC)	=	unchanged by this instruction
Condition Code	=	0010

Compare Floating Point (CE)
Compare Floating Point Register (CER)

Assembler Notation		Op-Code	Format
CE	R1, A(X2)	69	RX
CER	R1, R2	29	RR

Operation

The first operand is compared to the second operand. Comparision is algebraic, taking into account the sign, fraction, and exponent of each number. The result is indicated by the Condition Code setting. Neither operand is changed.

CER: (R1):(R2)
CE: (R1):[A+(X2)]

Condition Code

C	V	G	L
0	X	0	0
1	X	0	1
0	X	1	0

First operand is equal to second operand
First operand is less than second operand
First operand is greater than second operand

Programming Note

The state of the V flag is undefined.

In the RX formats, the second operand must be located on a fullword boundary.

Multiply Floating Point (ME)
Multiply Floating Point Register (MER)

Assembler Notation		Op-Code	Format
ME	R1, A(X2)	6C	RX
MER	R1, R2	2C	RR

Operation

The exponents of each operand, as derived from the excess 64 notation used in floating point representation, are added to produce the exponent of the result. This exponent is converted back to excess 64 notation. The fractions are then multiplied.

If the result is zero, the entire floating point value is forced to zero, X'0000 0000'. If the product is not zero, the result is normalized. The sign of the result is determined by the rules of algebra. The result replaces the contents of the register specified by R1.

MER: (R1) ← (R1) * (R2)
ME: (R1) ← (R1) * [A+(X2)]

Condition Code

C	V	G	L	
X	0	0	0	Floating point result is ZERO
X	0	0	1	Floating point result is less than ZERO
X	0	1	0	Floating point result is greater than ZERO
X	1	0	1	Exponent overflow, Result is negative
X	1	1	0	Exponent overflow, Result is positive
X	1	0	0	Exponent underflow

Programming Note

The addition of exponents may produce exponent overflow. In this case, the result is forced to the maximum value, ±X'7FFF FFFF'. The V flag in the Condition Code is set, along with either the G or the L flag, depending on the sign of the result. An arithmetic fault interrupt is taken, if enabled by bit 5 of the current PSW.

The addition of exponents or the normalization process can produce exponent underflow. In this case, the result is forced to zero, X'0000 0000'. The V flag in the Condition Code is set. The G and L flags are reset, and if enabled by bit 5 of the current PSW, the arithmetic fault interrupt is taken.

Multiplication of two 6-hexadecimal digit fractions effectively produces a result of 6-hexadecimal digits and a guard digit. If the guard digit is 0:7, the result is not rounded. If the guard digit is 8:F, the result is rounded by adding 1 to the fraction of the result, unless rounding produces a carry into the exponent field.

In the RX formats, the second operand must be located on a fullword boundary.

Example: ME

This example multiplies the contents of LOC by the contents of the Floating Point Register 8 and places the result in Floating Pointer Register 8.

Floating Point Register 8 contains X'5FFF FFFF'
LOC contains X'60FF FFFF'

Assembler Notation	Comments
ME REG8, LOC	Multiply (REG 8) by (LOC)

Result of ME Instruction

(Floating Point Register 8)	=	7FFF FFFE
(LOC)	=	unchanged by this instruction
Condition Code	=	0010

Divide Floating Point (DE)
Divide Floating Point Register (DER)

Assembler Notation		Op-Code	Format
DE	R1, A (X2)	6D	RX
DER	R1, R2	2D	RR

Operation

The exponents of each operand, as derived from the excess of 64 notation used in floating point representation, are subtracted to produce the exponent of the result. This exponent is converted back to excess 64 notation.

The first operand fraction is then divided by the second operand fraction. Division continues until the quotient is normalized, adjusting the exponent for each additional division required. No remainder is returned. The sign of the quotient is determined by the rules of algebra. The quotient replaces the contents of the register specified by R1.

DER: (R1) ⟵ (R1) / (R2)
DE: (R1) ⟵ (R1) / [A+(X2)]

Condition Code

C	V	G	L	
0	0	0	0	Floating point result is ZERO
0	0	0	1	Floating point result is less than ZERO
0	0	1	0	Floating point result is greater than ZERO
0	1	0	1	Exponent overflow, Result is negative
0	1	1	0	Exponent overflow, Result is positive
0	1	0	0	Exponent underflow
1	1	0	0	Divisor equal to zero

Programming Note

Before starting the divide operation, the divisor is checked. If it is equal to zero, the operation is aborted. Neither operand is changed. The C and the V flags of the Condition Code are set. The G and L flags are reset. If enabled by bit 5 of the current PSW, the arithmetic fault interrupt is taken.

The subtraction of exponents may produce exponent overflow. In this case, the result is forced to the maximum value, ±X'7FFF FFFF'. The V flag in the Condition Code is set, along with either the G or the L flag, depending on the sign of the result. An arithmetic fault interrupt is taken, if enabled by bit 5 of the current PSW.

The subtraction of exponents or the division process can produce exponent underflow. In this case, the result is forced to zero, X'0000 0000'. The V flag in the Condition Code is set. The G and L flags are always reset, and if enabled by bit 5 of the current PSW, the arithmetic fault interrupt is taken.

The 6-hexadecimal digit first operand fraction is divided by the 6-hexadecimal digit second operand effectively producing the 6-hexadecimal digit quotient along with a guard digit. If the guard digit is 0:7, the quotient is not rounded. If the guard digit is 8:F, the quotient is rounded by adding 1 to the fraction of the result unless rounding produces a carry into the exponent field.

In the RX formats, the second operand must be located on a fullword boundary.

Example: DE

This example divides the contents of Floating Point Register 4 by the contents of memory location LOC and places the result in Floating Pointer Register 4.

Floating Point Register 4 contains X'44FF FFFF' = Dividend
LOC contains X'0611 1111' = Divisor

Assembler Notation	Comments
DE REG4, LOC	Divide (REG4) by (LOC)

Result of DE Instruction

(Floating Point Register 4)	=	7FF0 0000
(LOC)	=	unchanged by this instruction
Condition Code	=	0010

INSTRUCTION

Fix Register (FXR)

Assembler Notation		Op-Code	Format
FXR	R1, R2	2E	RR

Operation

R1 specifies one of the general purpose registers. R2 specifies one of the single precision floating point registers. The floating point number contained in the floating point register is converted to a two's complement notation integer value by shifting and truncating. The 16-bit result is stored in the register specified by R1.

1. (R2): '4880000 ', then:
   if (R2)≥ X'440000', then:

2. (R1)←'7FFF', go to step 9
   if (R2)< X'44000000', then:

3. (R2): Y'41000000'
   if (R2) < Y'41000000'

4. (R1)←X'0000', Go to step 9
   if (R2) ≥ Y'41000000', then:

5. Count← R2 (1:7)

6. Count← X'44' - Count

7. If Count = 0, Go to step 9

8. If Count ≠ 0, then
   Shift (R1) Right four places
   Count←Count-1, go to step 7

9. If (R2) = Positive, end of instruction
   If (R2) = Negative, then:
   (R1)←0-(R1)

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO or underflow
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO
X	1	0	1	Overflow, Result is negative
X	1	1	0	Overflow, Result is positive

Programming Note

The range of floating point magnitudes M that produces a non-zero integral result is:

$$\pm \text{ Y'4480 0000'} > M \geq \pm \text{ Y'4110 0000'}$$

Floating point magnitudes greater than + Y'447F FFFF' cause overflow. The result is forced to X'7FFF' if positive or to X'8001' if negative. The V flag is set in the Condition Code along with either the G or L flag, depending on the sign of the result.

Floating point magnitudes less than +Y'4110 0000' cause underflow and the result is forced to zero.

In the event of overflow or underflow, the Arithmetic Fault Interrupt is not taken, even if enabled in the current PSW.

Example: FXR

This example converts the contents of the Floating Point Register 8 to a fixed point number and places it in Register 2.

Floating Point Register 8 contains Y'42F0 0000'
Register 2 is undefined

Assembler Notation		Comments
FXR	REG2, REG8	Convert (REG 8) to fixed point

Result of FXR Instruction

(REG2)	= 00F0
(Floating Point Register 8)	= unchanged by this instruction
Condition Code	= 0010

INSTRUCTION

Float Register (FLR)

Assembler Notation		Op-Code	Format
FLR	R1,R2	2F	RR

Operation

R1 specifies one of the single precision floating point registers. R2 specifies one of the general purpose registers. The 16-bit integer value contained in the register specified by R2 is converted to a floating point number and stored in the floating point register specified by R1.

1. Temporary A← Y '4600'
2. Temporary B ← (R2)
3. if Temporary B is minus then:
   Temporary B←—0-Temporary B and
   Temporary A← Y'C600'
4. Normalize (Temporary A, Temporary B)
5. (R1)←—Temporary A

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO

Programming Note

The full range of fixed point integer values may be converted to floating point. The fixed point value X'7FFF', the largest positive integer, converts to a floating point value of Y'447F FF00'. The fixed point value X'8000', the most negative integer, converts to a floating point value of Y'C480 0000'. The result in R1 is normalized.

Example: FLR

This example converts the Fixed point contents of Register 4 to a floating point number and places it into Floating Point Register 8.

Register 4 contains X'7FFF'
Floating Point Register 8 is undefined

Assembler Notation		Comments
FLR	REG8, REG4	Convert (REG4) to Floating Point

Result of FLR Instruction

(Floating Point Register 8)	=	447FFF00
(REG4)	=	unchanged by this instruction
Condition Code	=	0010

Load Double Precision Floating Point (LD)
Load Register Double Precision Floating Point (LDR)

Assembler Notation	Op-Code	Format
LD R1, A (X2)	78	RX
LDR R1, R2	38	RR

Operation

The floating point second operand is normalized, if necessary, and placed in the double precision floating point register specified by R1.

LDR: (R1) ← (R2)
LD: (R1) ← [A+(X2)]

Condition Code

C	V	G	L	
0	0	0	0	Double precision value is ZERO
0	0	0	1	Double precision value is less than ZERO
0	0	1	0	Double precision value is greater than ZERO
0	1	0	0	Exponent underflow

Programming Note

If the fraction is zero, the result is forced to X'0000 0000 0000 0000'.

Normalization may produce exponent underflow. In this event, the result is forced to X'0000 0000 0000 0000', the V flag in the Condition Code is set, the G and L flags are reset and, if enabled by bit-19 of the current PSW, the arithmetic fault interrupt is taken.

In the RX formats, the second operand must be located on a double word boundary.

Example: LD

This example normalizes the double word at memory location LOC and places it in Double Precision Floating Point Register 6.

Floating Point Register 6 = undefined
LOC = 4300 0000 0230 0000

Assembler Notation	Comments
LD REGG. LOC	Normalize contents of LOC

Result of LD Instruction

(Floating Point Register 6) = 3C23 0000 0000 0000
(LOC) = unchanged by this instruction
Condition Code = 0010

INSTRUCTION

Load Multiple Double Precision Floating Point (LMD)

Assembler Notation	Op-Code	Format
LMD R1, A (X2)	7F	RX

Operation

Successive double-precision floating point registers, starting with the register specified by R1, are loaded from successive memory locations starting with the address of the second operand. The process stops when Double Precision Floating Point Register 14 has been loaded.

1. (R1)←[A+(X2)]
2. R1:X'E'
   if R1=X'E', the instruction is finished
   if R1≠ X'E', then:
3. R1←R1 + 2
4. A←A + 8, return to step 1

Condition Code

Unchanged

Programming Note

The second operand must be located on a double word boundary.

INSTRUCTION

Store Double Precision Floating Point (STD)

Assembler Notation	Op-Code	Format
STD R1, A (X2)	70	RX

Operation

The floating point first operand, contained in the double precision floating point register specified by R1 is placed in the memory location specified by the second operand address. The first operand is unchanged.

$[A + (X2)] \longleftarrow (R1)$

Condition Code

Unchanged.

Programming Notes

The second operand must be located on a double word boundary.

This instruction is subject to memory protect.

INSTRUCTION

Store Multiple Double Precision Floating Point (STMD)

Assembler Notation	Op-Code	Format
STMD R1, A (X2)	7E	RX

Operation

The contents of successive double precision floating point registers, starting with the register specified by R1, are stored in successive memory locations, starting with the address of the second operand. The operation stops when the contents of Double Precision Floating Point Register 14 have been stored.

1. [A + (X2)] ← (R1)
2. R1:X'E'
   if R1 = X'E', the instruction is finished
   if R1 ≠ X'E', then:
3. R1 ← R1 + 2
4. A ← A + 8, return to step 1

Condition Code

Unchanged

Programming Note

The second operand must be located on a double word boundary.

This instruction is subject to memory protect.

Add Double Precision Floating Point (AD)
Add Register Double Precision Floating Point (ADR)

Assembler Notation		Op-Code	Format
AD	R1, A (X2)	7A	RX
ADR	R1, R2	3A	RR

Operation

The exponents of the two operands are compared. If the exponents differ the fraction with the smaller exponent is shifted right hexadecimally (four bits at a time), and its exponent is incremented by one for each hexadecimal shift until the two exponents are equal. The fractions are then added algebraically.

If the addition of fractions produces a carry, the exponent of the result is incremented by one and the fraction of the result is shifted right one hexadecimal position. The carry bit is shifted back into the most significant hexadecimal digit of the fraction, producing a normalized result. This result replaces the contents of the double precision floating point register specified by R1.

If the addition of fractions does not produce a carry, the result is normalized, if necessary, and placed in the double precision floating point register specified by R1.

ADR: (R1) ⟵ (R1) + (R2)
AD: (R1) ⟵ (R1) + [A + (X2)]

Condition Code

C	V	G	L	
X	0	0	0	Double Precision Result is ZERO
X	0	0	1	Double Precision Result is less than ZERO
X	0	1	0	Double Precision Result is greater than ZERO
X	1	0	1	Exponent Overflow, Result is negative
X	1	1	0	Exponent Overflow, Result is positive
X	1	0	0	Exponent Underflow

Programming Note

When the addition of fractions produces a carry, incrementing the exponent of the result by one may produce exponent overflow. In this case, the result is forced to the maximum value, ±X'7FFF FFFF FFFF FFFF', the V flag, along with the G or L flag is set in the Condition Code and, if enabled by bit 5 of the current PSW, the arithmetic fault interrupt is taken.

Normalization of the result may produce exponent underflow. In this case, the result is forced to zero, X'0000 0000 0000 0000'. The V flag is set in the Condition Code, and the G and L flags are reset, and if enabled by bit 5 of the current PSW, the arithmetic fault interrupt is taken.

In the RX formats, the second operand must be located on a double word boundary.

Subtract Double Precision Floating Point (SD)
Subtract Register Double Precision Floating Point (SDR)

Assembler Notation		Op-Code	Format
SD	R1, A(X2)	7B	RX
SDR	R1, R2	3B	RR

Operation

The exponents of the two operands are compared. If the exponents differ, the fraction with the smaller exponent is shifted right hexadecimally (four bits at a time), and its exponent is incremented by one for each hexadecimal shift until the two exponents are equal. The second operand fraction is then subtracted algebraically from the first operand fraction.

If the subtraction of fractions produces a carry, the exponent of the result is incremented by one and the fraction of the result is shifted right one hexadecimal position. The carry bit is shifted back into the most significant hexadecimal digit of the fraction producing a normalized result. This result replaces the contents of the double precision floating point register specified by R1.

SDR: (R1) ⟵ (R1) - (R2)
SD: (R1) ⟵ (R1) - [A + (X2)]

Condition Code

C	V	G	L	
X	0	0	0	Double Precision Result is ZERO
X	0	0	1	Double Precision Result is less than ZERO
X	0	1	0	Double Precision Result is greater than ZERO
X	1	1	0	Exponent Overflow, Result is positive
X	1	0	1	Exponent Overflow, Result is negative
X	1	0	0	Exponent Underflow

Programming Note

When the subtraction of fractions produces a carry, incrementing the exponent of the result by one may produce exponent overflow. In this case, the result is forced to the maximum value, ±X'7FFF FFFF FFFF FFFF', the V flag, along with the G or L flag is set in the Condition Code, and if enabled by bit 5 of the current PSW, the arithmetic fault interrupt is taken.

Normalization of the result may produce exponent underflow. In this case, the result is forced to zero, X'0000 0000 0000 0000'. The V flag is set in the Condition Code, the G and L flags are reset, and if enabled by bit 5 of the current PSW, the arithmetic fault interrupt is taken.

In the RX formats, the second operand must be located on a double word boundary.

Compare Double Precision Floating Point (CD)
Compare Register Double Precision Floating Point (CDR)

Assembler Notation		Op-Code	Format
CD	R1, A(X2)	79	RX
CDR	R1, R2	39	RR

Operation

The first operand is compared to the second operand. Comparison is algebraic, taking into account the sign, exponent and fraction of each number. The result is indicated by the Condition Code setting. Neither operand is changed.

CDR: (R1):(R2)
CD: (R1): [A + (X2)]

Condition Code

C	V	G	L	
0	X	0	0	First operand is equal to second operand
1	X	0	1	First operand is less than second operand
0	X	1	0	First operand is greater than second operand

Programming Note

The state of the overflow flag is undefined.

In the RX formats, the second operand must be located on a double word boundary.

Multiply Double Precision Floating Point (MD)
Multiply Register Double Precision Floating Point (MDR)

Assembler Notation		Op-Code	Format
MD	R1, A (X2)	7C	RX
MDR	R1, R2	3C	RR

Operation

The exponents of the two operands, as derived from the excess 64 notation used in floating point representation, are added to produce the exponent of the result. This exponent is converted back to excess 64 notation. The fractions are then multiplied.

If the product is zero, the entire double precision value is forced to zero, X'0000 0000 0000 0000'. If the product is not zero, the result is normalized if necessary. After normalization, the product is rounded. The sign of the result is determined by the rules of algebra. The result replaces the contents of the double precision floating point register specified by R1.

MDR: (R1) ⟵ (R1) * (R2)
MD: (R1) ⟵ (R1) * [A + (X2)]

Condition Code

C	V	G	L	
X	0	0	0	Double precision result is ZERO
X	0	0	1	Double precision result is less than ZERO
X	0	1	0	Double precision result is greater than ZERO
X	1	1	0	Exponent overflow, Result is positive
X	1	0	1	Exponent overflow, Result is negative
X	1	0	0	Exponent underflow

Programming Note

The addition of exponents may produce exponent overflow. In this case, the result is forced to the maximum value, ±X'7FFF FFFF FFFF FFFF'. The V flag in the Condition Code is set, along with either the G or L flag, depending on the sign of the result. An arithmetic fault interrupt is taken, if enabled by bit 5 of the current PSW.

The addition of exponents or the normalization process can produce exponent underflow. In this case, the result is forced to zero, X'0000 0000 0000 0000'. The V flag in the Condition Code is set, the G and L flags are reset, and if enabled by bit 5 of the current PSW, the arithmetic fault interrupt is taken.

In the RX formats, the second operand must be located on a double word boundary.

Divide Double Precision Floating Point (DD)
Divide Register Double Precision Floating Point (DDR)

Assembler Notation		Op-Code	Format
DD	R1, A (X2)	7D	RX
DDR	R1, R2	3D	RR

Operation

The exponents of the two operands, as derived from the excess 64 notations used in floating point representation, are subtracted to produce the exponent of the result. This exponent is converted back to excess 64 notation.

The second operand fraction is then divided into the first operand fraction. Division continues until the quotient is normalized, adjusting the exponent for each additional division required.

No remainder is returned. The sign of the quotient is determined by the rules of algebra. The quotient replaces the contents of the double precision floating point register specified by R1.

DDR: (R1) ⟵ (R1) / (R2)
DD: (R1) ⟵ (R1) / [A + (X2)]

Condition Code

C	V	G	L	
0	0	0	0	Double precision result is ZERO
0	0	0	1	Double precision result is less than ZERO
0	0	1	0	Double precision result is greater than ZERO
0	1	0	1	Exponent overflow, Result is negative
0	1	1	0	Exponent overflow, Result is positive
0	1	0	0	Exponent underflow
1	1	0	0	Divisor is zero

Programming Notes

Before starting the divide operation, the divisor is checked. If it is equal to zero, the operation is aborted. Neither operand is changed. The C and V flags in the Condition Code are set, the G and L flags are reset, and if enabled by bit 5 of the current PSW, the arithmetic fault interrupt is taken.

The subtraction of exponents may produce exponent overflow. In this case, the result is forced to the maximum value, ±X'7FFF FFFF FFFF FFFF'. The V flag in the Condition Code is set, along with either the G or L flag, depending on the sign of the result. An arithmetic fault interrupt is taken, if enabled by bit 5 of the current PSW.

The subtraction of exponents or the division process may produce exponent underflow. In this case, the result is forced to zero, X'0000 0000 0000 0000'. The V flag in the Condition Code is set, the G and L flags are reset, and if enabled by bit 5 of the current PSW, the arithmetic fault interrupt is taken.

In the RX formats, the second operand must be located on a double word boundary.

INSTRUCTION

Fix Register Double Precision (FXDR)

Assembler Notation		Op-Code	Format
FXDR	R1, R2	3E	RR

Operation

R1 specifies one of the general purpose registers. R2 specifies one of the double precision floating point registers. The floating point number contained in the floating point register is converted to an integer value by truncating. The 16-bit result is placed in the general register specified by R1.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO or underflow
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO
X	1	0	1	Overflow, Result is negative
X	1	1	0	Overflow, Result is positive

Programming Notes

The range of the floating point magnitude (M) that produces a non-zero integral result is,
$\pm$ X'4480 0000 0000 0000' > M $\geq$ $\pm$ X'4110 0000 0000 0000'

Double precision floating point magnitudes greater than +X'447F FFFF FFFF FFFF' cause overflow. The result is forced to X'7FFF' if positive or to X'8001' if negative. The V flag is set in the Condition Code along with either the G or L flag, depending on the sign of the result.

Double Precision floating point magnitudes less than +X'4110 0000 0000 0000' cause underflow. The result is forced to zero and the Condition Code is set to zero.

In the event of overflow or underflow, the Arithmetic Fault Interrupt is not taken even if enabled in the current PSW.

Example : FXDR

This example converts the contents of the Double Precision Floating Point Register 8 to a fixed point number and places it in Register 3.

Floating Point Register 8 contains X'C410 0000 0000 0000'
Register 3 is undefined.

Assembler Notation		Comments
FXDR	REG3, REG8	Convert (REG8) to fixed point

Result of FXDR Instuction

(REG3) = X'F000'
(Floating Point REG8) = X'C410 0000 0000 0000'
Condition Code = 0001

INSTRUCTION

Float Register Double Precision (FLDR)

Assembler Notation	Op-Code	Format
FLDR R1, R2	3F	RR

Operation

R1 specifies one of the double precision floating point registers. R2 specifies one of the general purpose registers. The 16-bit integer value contained in the register specified by R2 is converted to a floating point number and placed in the double precision floating point register specified by R1.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO

Programming Notes

The full range of fixed point integer values may be converted to double precision floating point. The fixed point value X'7FFF', the largest positive integer, converts to a double precision floating point value of X'44F FF00 0000 0000'. The fixed point value X'8000', the most negative integer, converts to a double precision floating point value of X'C480 0000 0000 0000'.

The result in R1 is normalized.

Example : FLDR

This example converts the Fixed point contents of Register 7 to a Floating Point number and places it into Floating Point Register 8.

Register 7 contains X'FFFE'
Floating Point Register 8 is undefined

Assembler Notation	Comments
FLDR REG8, REG7	Convert (REG7) to Floating Point

Result of FLDR Instruction

(Floating Point Register 8) = C120 0000 0000 0000
(Register 7) = FFFE
Condition Code = 0001

CHAPTER 7
MEMORY MANAGEMENT

The Model 8/16E Processor is unique in the 16-bit product line in that it is the first to allow addressing of more than the customary 64KB of main memory. As much as 256KB of main memory can be added to the Model 8/16E Processor

INTRODUCTION

The maximum address in 256KB is X'3FFFF'. As this number requires 18 bits to express, there must be a translation mechanism to expand the normal 16-bit address. In the 8/16E, the mechanism involves four, previously un-assigned, bits in the Program Status Word (bits 8 through 11) and the restriction that the Processor can only access 64KB of memory at one time.

To understand the address translation mechanism, the difference between a physical address and a program address must be understood. Simply, a physical address is an 18-bit number that allows accessing of 256KB of memory. The program address remains 16 bits, resulting in a 64KB-at-a-time limitation.

The program address is the 16-bit instruction Location Counter if fetching an instruction, or it is the 16-bit effective address of an operand. The 64KB range of program addresses is divided into two 32KB segments. Segment 0 represents program address from X'0000' through X'7FFF'. Segment 1 represents program address from X'8000' through X'FFFF'.

The 256KB range of physical addresses is divided into eight 32KB segments. Based upon the setting in PSW bits 8, 9, 10, and 11, program addresses in segment 0 refer to a corresponding location in one of the physical address segments and program addresses in segment 1 refer to a corresponding location in another physical address segment.

When PSW bits 8 through 11 equal 0000_2 or 1111_2, a program address and the resulting physical address are identical. The top two bits of the physical address remain reset. Program addresses in segment 0 refer to locations in physical segment 0 and program addresses in segment 1 refer to locations in physical segment 1.

When PSW bits 8 through 11 equal 0001_2 through 0110_2, program addresses in segment 0 still refer to locations in physical segment 0, but program addresses in segment 1 refer to locations in physical segment 2, 3, 4, 5, 6, or 7, depending on the actual value in PSW bits 8 through 11. See Figure 7-1.

When PSW bits 8 through 11 equal 0111_2 through 1110_2, program addresses in segment 0 refer to locations in physical segment 1 instead of 0. Program addresses in segment 1 refer to locations in physical segment 0, 1, 2, 3, 4, 5, 6, or 7. See Figure 7-2.

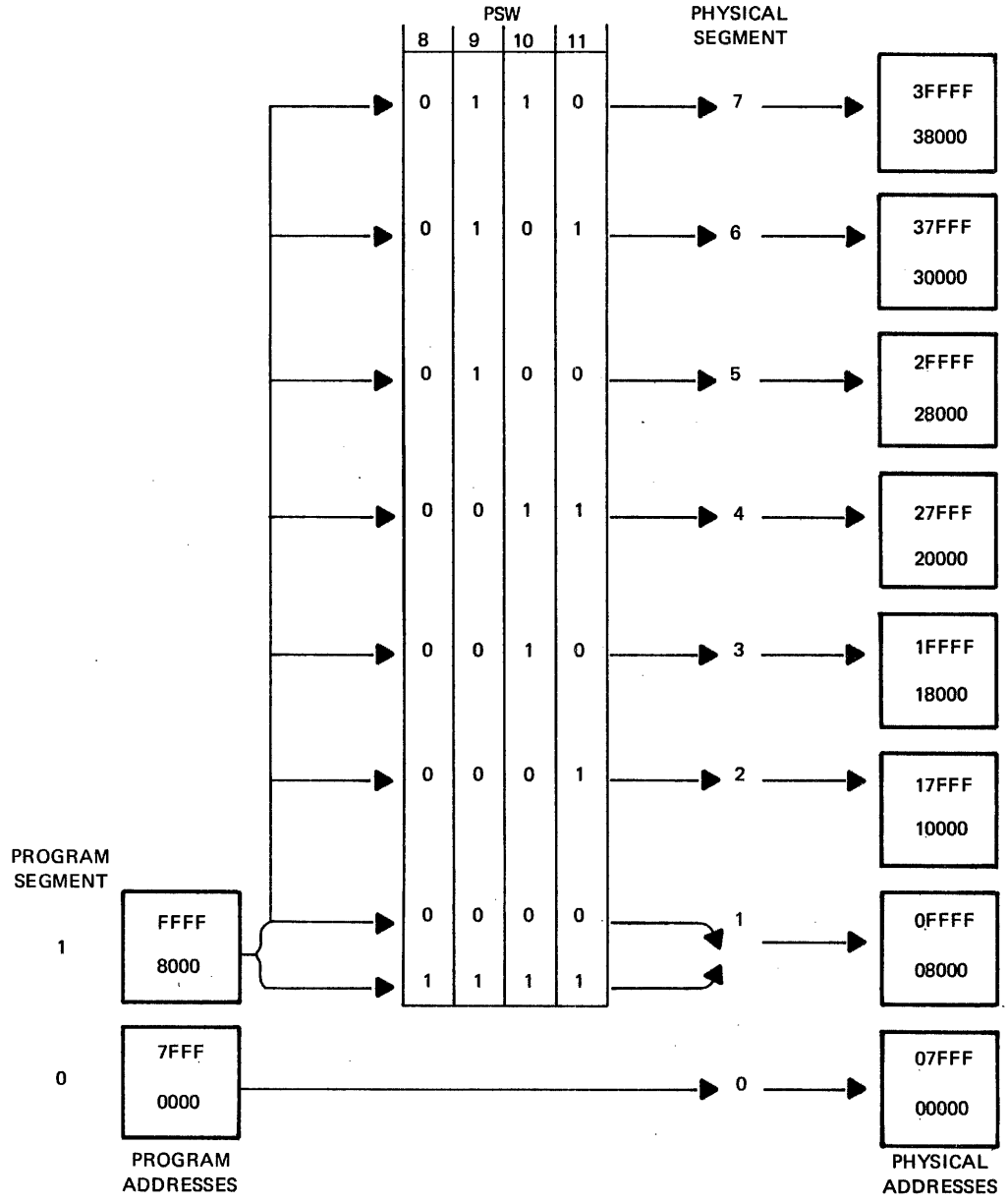


Figure 7-1. Case 1 Translation from Program Address to Physical Address.

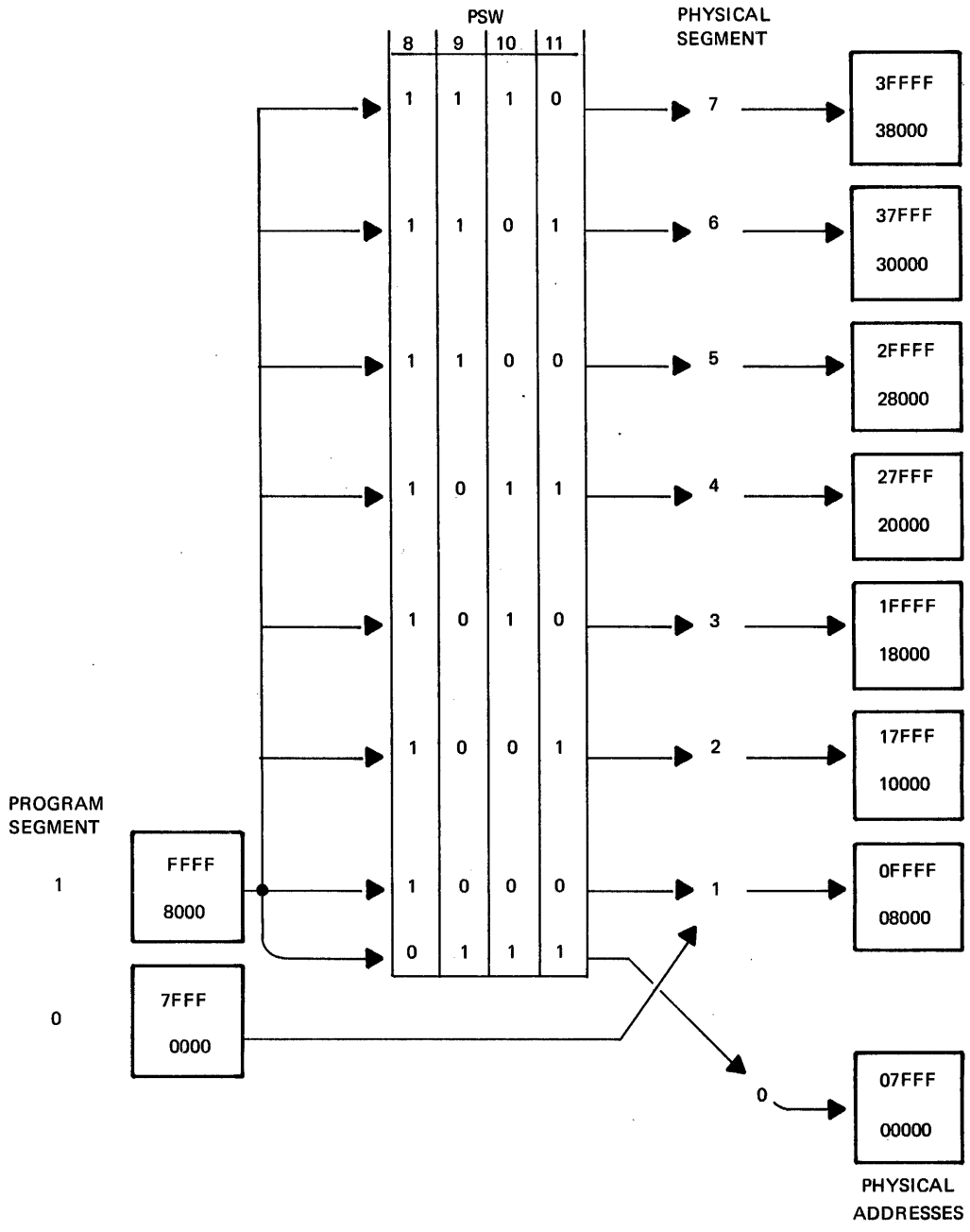


Figure 7-2. Case 2 Translation from Program Address to Physical Address.

Table 7-1 summarizes the relationship between program addresses and physical addresses. In order to change from the currently active 64KB, the current PSW must be modified. Now the PSW can be modified directly by doing an LPSW or EPSR instruction. Each of these instructions has its disadvantages - LPSW modifies the entire PSW, Location Counter included; and EPSR requires two general registers.

TABLE 7-1. RELATIONSHIP BETWEEN PROGRAM ADDRESS AND PHYSICAL ADDRESS

PSW				PROGRAM ADDRESS	PROGRAM ADDRESS
08	09	10	11	0000 - 7FFF	8000 - FFFF
0	0	0	0	00000-07FFF	08000-0FFFF
0	0	0	1	00000-07FFF	10000-17FFF
0	0	1	0	00000-07FFF	18000-1FFFF
0	0	1	1	00000-07FFF	20000-27FFF
0	1	0	0	00000-07FFF	28000-2FFFF
0	1	0	1	00000-07FFF	30000-37FFF
0	1	1	0	00000-07FFF	38000-3FFFF
0	1	1	1	08000-0FFFF	00000-07FFF
1	0	0	0	08000-0FFFF	08000-0FFFF
1	0	0	1	08000-0FFFF	10000-17FFF
1	0	1	0	08000-0FFFF	18000-1FFFF
1	0	1	1	08000-0FFFF	20000-27FFF
1	1	0	0	08000-0FFFF	28000-2FFFF
1	1	0	1	08000-0FFFF	30000-37FFF
1	1	1	0	08000-0FFFF	38000-3FFFF
1	1	1	1	00000-07FFF	08000-0FFFF

MEMORY SEGMENT SELECTION INSTRUCTION FORMATS

To make the selection of different memory segments easier, four new user level instructions have been added to the Model 8/16E instruction repertoire. These instructions are:

LPSR	Load Program Status Field Register
LPS	Load Program Status Field
SETMR	Set Map Register
SETM	Set Map

Load Program Status Field Register (LPSR)

Load Program Status Field (LPS)

Assembler Notation		Op-Code	Format
LPSR	R2	33	RR
LPS	A(X2)	73	RX

Operation

The halfword operand replaces bits 0 through 15 of the Program Status Word.

LPSR: PSW(0:15)←(R2)

LPS: PSW(0:15)←[A+(X2)]

Condition Code

Determined by the new status.

Programming Note

The R1 field of this instruction is not used by the Processor. The Assembler automatically sets the R1 field to zero.

These instructions are priviledged operations.

In the RX format, the second operand must be on a halfword boundary.

Set Map Register (SETMR)

Set Map (SETM)

Assembler Notation		Op-Code	Format
SETMR	R1, R2	13	RR
SETM	R1, A(X2)	53	RX

Operation

The register specified by R1 contains a user's program address. The halfword second operand contains a new PSW status field. This instruction copies bits 8 through 11 of the second operand to bits 8 through 11 of the current PSW. These new bits and the specified program address are adjusted as necessary so that the location can be accessed.

SETM (R):

1. PSW (8:11)←bits 8:11 of second operand.
2. If PSW (8:11) = 0000_2 through 0110_2, or 1111_2, Go to Step 6.
3. If PSW (8:11) = 0111_2, Complement bit 0 of R1 and Set PSW (8:11) = 0000_2 and Go to Step 6.
4. If PSW (8:11) = 1000 through 1110, and if R1 bit 0 is reset, Set bit 0 of R1 and Set PSW (8:11) = 0000_2 and Go to Step 6.
5. If PSW (8:11) = 1000_2 through 1110, and if R1 bit 0 is set, Reset PSW bit 8 and Go to Step 6.
6. Adjust Condition Code of PSW according to the new value of R1 and exit.

Condition Code

C	V	G	L	
0	0	0	0	Value in R1 is zero
0	0	0	X	Value in R1 is Negative
0	0	X	0	Value in R1 is Positive

Programming Note

These instructions are priviledged operations.

In the RX format, the second operand must be located on a halfword boundary.

SUMMARY OF SETM, SETMR

SECOND OPERAND BITS 8:11	ORIGINAL R1 BIT 0	NEW R1 BIT 0	NEW PSW BITS 8:11
0000	0	0	0000
0000	1	1	0000
0001	0	0	0001
0001	1	1	0001
0010	0	0	0010
0010	1	1	0010
0011	0	0	0011
0011	1	1	0011
0100	0	0	0100
0100	1	1	0100
0101	0	0	0101
0101	1	1	0101
0110	0	0	0110
0110	1	1	0110
0111	0	1	0000
0111	1	0	0000
1000	0	1	0000
1000	1	1	0000
1001	0	1	0000
1001	1	1	0001
1010	0	1	0000
1010	1	1	0010
1011	0	1	0000
1011	1	1	0011
1100	0	1	0000
1100	1	1	0100
1101	0	1	0000
1101	1	1	0101
1110	0	1	0000
1110	1	1	0110
1111	0	0	1111
1111	1	1	1111

CHAPTER 8
STATUS SWITCHING AND INTERRUPTS

INTRODUCTION

At any given time, the Processor may be in either the Stop or the Run mode. In the Stop mode, the normal execution of instructions is suspended. The Processor is under control of the operator who can, through the Display Console:

Examine the contents of any memory location

Change the contents of any memory location

Examine the contents of any general register

Examine the contents of any floating point register

Examine the contents of the Program Status Word

Execute instructions singularly

Put the Processor in the Run mode

Once the Processor has been put in the Run mode, the current Program Status Word controls the operation of the Processor. By changing the contents of the current PSW, a running program can:

Put the Processor in the Wait state

Enable or disable various interrupts

Switch between the supervisor and the protect modes

Vary the normal sequential execution of instructions

PROGRAM STATUS WORD

The Program Status Word is a 32-bit fullword as shown in Figure 8-1.

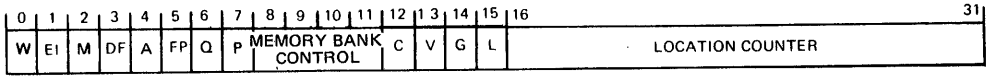


Figure 8-1. Program Status Word Format

Bits 0:15 of the PSW are reserved for status definitions. Bits 12:15 are reserved for the Condition Code. Bits 16:31 are reserved for the Location Counter. The status definition bits are interpreted as follows:

Bit 0	(W)	Wait state
Bit 1	(EI)	External interrupt mask
Bit 2	(M)	Machine malfunction interrupt mask
Bit 3	(DF)	Fixed point fault interrupt mask
Bit 4	(A)	Automatic I/O and immediate interrupt mask
Bit 5	(FP)	Floating point fault interrupt mask
Bit 6	(Q)	Queue service interrupt mask
Bit 7	(P)	Protect mode
Bits 8:11		Memory Bank Control

The current PSW is contained in a hardware register within the Processor. Status switching results when the current PSW, or at least the first half (bits 0:15) of the current PSW is replaced. The occurrence of an interrupt or the execution of a Status Switching instruction can cause the replacement of the current PSW.

Wait State

Replacing the current PSW with one in which bit 0 is set puts the Processor in the Wait state. When the Processor is in the Wait state, program execution is halted. However, the Processor is still responsive to machine malfunction, external, and immediate interrupts, if they are enabled. Automatic I/O channel operations can also temporarily force the Processor out of the Wait state. If the Processor is put in the Wait state with all interrupts disabled, only operator intervention from the Display Console can force the Processor out of the Wait state.

Protect Mode

When bit 7 of the current PSW is set, the Processor is in the Protect mode. A program running in this mode is not allowed to execute Privileged instructions. (Privileged instructions include all I/O instructions, and most of the Status Switching instructions.) If bit 7 of the current PSW is reset, the Processor is in the Supervisor mode. Programs running in this mode may execute any legal instruction. On models not equipped with the protect mode, PSW bit 7 has no significance and there is no privileged instruction detection.

INTERRUPT SYSTEM

The interrupt system of the Processor provides rapid response to external and internal events that require service by special software routines. In the interrupt response procedure, the Processor preserves its current state, and transfers control to the required interrupt handler. This software routine may optionally restore the previous state of the Processor upon completion of the service.

Some interrupts are controlled by bits in the current Program Status Word. That is, they can be enabled or disabled by setting or resetting a bit in the PSW. Other interrupts are not controlled by PSW bits, and are always enabled. The following is a list of Processor interrupts and their controlling PSW bits, if any:

Interrupt	PSW Bit
External	1
Machine Malfunction	2
Fixed Point Fault	3
Automatic I/O	4
Floating Point Fault	5
System Queue Service	6
Protect Mode Violation	7
Supervisor Call	none
Simulated	none
Illegal Instruction	none
System Queue Overflow	none

Interrupts occur at various times during processing. The external, immediate, console, and machine malfunction interrupts occur between the execution of instructions, or after the completion of an automatic I/O channel operation. The system queue service, arithmetic fault, supervisor call, and simulated interrupts occur during the execution of instructions. The system queue overflow interrupt occurs as part of an automatic I/O channel operation. The illegal instruction and protect mode violation interrupts occur before the execution of the improper instruction.

The interrupt procedure is based on the concept of old, current, and new Program Status Words. The current PSW, contained in the hardware register, defines the operating state of the Processor. When this state must be changed, the current PSW becomes the old PSW. The new PSW becomes the current PSW. The current PSW now contains the operating status and the Location Counter for the interrupt service routine.

External Interrupt

This I/O interrupt provides compatibility with previous Perkin-Elmer Processors. Bit 1 of the current PSW controls this interrupt. If this bit is set and bit 4 reset (see immediate interrupt), and an external device requests Processor service, the following action takes place:

The current Program Status Word replaces the contents of memory locations X'0040' - X'0043'.

The new Program Status Word from locations X'0044' - X'0047' becomes the current Program Status Word.

From this point it is up to the software to identify the interrupting device, and take appropriate action.

Machine Malfunction Interrupt

Bit 2 of the current Program Status Word controls the machine malfunction interrupt. This interrupt may occur on a memory parity error, on the detection of primary power failure, or during the restart procedure after power has been restored. When the machine malfunction interrupt occurs, the current Program Status Word is saved in memory locations X'0038' - X'003B'. The new PSW from locations X'003C' - X'003F' becomes the current PSW. The new PSW as stored in memory must have zeros in the Condition Code. When the new PSW becomes the current PSW, the Condition Code indicates the type of machine malfunction. These Condition Code states are:

C	V	G	L	
0	0	0	0	Power restore
0	0	0	1	Power failure

The new Program Status Word for the machine malfunction interrupt must disable this interrupt.

The power fail interrupt occurs when the primary power fail detector senses a low voltage, when the initialize switch of the Display Console is depressed, or when the key operated power switch is turned to the OFF position. Following the PSW exchange, the software has approximately one millisecond to perform any necessary operations before the automatic shut down procedure takes over. During the automatic shut down procedure, the Processor saves the current PSW in memory locations X'0024' - X'0027'. Single precision floating point registers, if equipped, are stored in memory locations X'0' - X'001E'. The contents of the general registers are saved in 16 successive halfword locations starting at the address specified in memory locations X'0022' - X'0023'. Double precision floating point registers, if equipped, are stored in the 16 successive halfwords following the general registers

When power returns, the Processor restores the PSW, the floating point registers, and the general registers from their save areas. If bit 2 of the restored PSW is set, the Processor takes another machine malfunction interrupt, this time with no bits set in the Condition Code.

The user is assumed to have set a flag in his machine malfunction interrupt handler indicating that he has executed his power-down sequence. Testing of this flag on a machine malfunction in conjunction with a test of the L flag in the condition code will allow determination of power restore versus memory error interrupts.

Fixed Point Fault Interrupt

Bit 3 of the current PSW controls this interrupt. If this bit is set, the interrupt is enabled. A fixed point fault interrupt occurs for either of two reasons:

The divisor in a Fixed Point Divide instruction is zero.

The signed quotient resulting from a fixed point divide operation cannot be expressed in 16 bits.

This interrupt is always taken before any operand is changed. The current PSW is saved in memory locations X'48' - X'4B'. The new PSW, contained in memory locations X'4C' - X'4F', becomes the current PSW. The Location Counter of the old PSW contains the address of the instruction following the one that caused the interrupt.

If bit 3 of the current PSW is reset, quotient overflow or attempted division by zero do not cause an interrupt. The operands are unchanged, and the next sequential instruction is executed.

Immediate Interrupt

If both bit 1 and bit 4 of the current Program Status Word are set, an interrupt request from a peripheral device results in an automatic I/O operation. This may be either an automatic I/O channel operation or an immediate interrupt.

When the Processor receives the interrupt request, it automatically acknowledges the request. The device, in turn, responds with its unique device number. The Processor doubles this number, and uses the result as an index into the interrupt service pointer table, which must contain a half-word entry for each of the possible 256 device numbers. The table starts at memory location X'00D0', and extends through location X'02CF'. (Chapter 9, Input/Output Operations, contains detailed descriptions of the make-up of this table and its use in both interrupt driven I/O and automatic I/O channel operations.)

If the location reserved for the interrupting device contains an odd value, the Processor starts an automatic I/O channel operation. Otherwise, the Processor takes the immediate interrupt, and the following events occur:

The current Program Status Word is saved in the location specified by the entry in the table.

The status portion (bits 0:15) of the Program Status Word is loaded with the value contained in the memory location obtained by adding four to the value contained in the table.

The Location Counter of the current Program Status Word is loaded with a value obtained by adding six to the address contained in the table.

The immediate interrupt provides hardware vectoring of external interrupt requests. Each device on the system may have a unique location for the interrupt service routine. If several devices of the same type are included in the system, one service routine may be used for all, if the interrupting device is first identified and then a branch is taken to the common service routine.

Console Interrupt

The console interrupt is also controlled by bit 4 of the current Program Status Word. If this bit is set, and if the operator:

Depresses the console function key, FN, and,

Depresses the hexadecimal 0 key,

the Processor acts as if it had received an interrupt request from device X'01'. The effect may be either an immediate interrupt, or the activation of the automatic I/O channel. If bit 4 of the current Program Status Word is reset, and the operator attempts to generate a console interrupt request, the request is ignored. It is not queued. On models not equipped with a display panel, there is no provision for the console interrupt.

Floating Point Fault Interrupt

The floating point fault interrupt, enabled by bit 5 of the current Program Status Word, occurs on exponent overflow, exponent underflow, or division by zero. On exponent overflow, the result is forced to $\pm$X'7FFF FFFF'. On exponent underflow, the result is forced to X'0000 0000'. On division by zero, the destination register is unchanged.

When this interrupt occurs, the current Program Status Word is saved in memory locations X'0028' - X'002B'. The new Program Status Word from locations X'002C' - X'002F' becomes the current Program Status Word. The Location Counter of the old PSW contains the address of the next instruction location following the one that caused the interrupt.

System Queue Interrupt

The system queue serves both hardware (channel I/O) and software. Whenever the Processor executes a Load Program Status Word or an Exchange Program Status Register instruction, or when it prepares to resume normal program execution after a channel I/O operation, it checks Bit 6 of the current Program Status Word. If this bit is set, and if there is an item in the system queue, the Processor takes the system queue interrupt. Taking this interrupt causes the current Program Status Word to be saved in memory locations X'0082' — X'0085'. The new Program Status Word contained in memory locations X'0086' — X'0089' becomes the current Program Status Word.

Protect Mode Violation Interrupt

The Protect mode violation interrupt is enabled by bit 7 of the current Program Status Word. Setting this bit puts the Processor in the Protect mode. The interrupt occurs when a program, running in the Protect mode, attempts to execute a Privileged instruction. Privileged instructions include all I/O operations and several of the Status Switching instructions. On taking this interrupt, the current Program Status Word is saved in memory locations X'0030' — X'0033'. The new Program Status Word from locations X'0034' — X'0037' becomes the current Program Status Word. The old Location Counter contains the address of the instruction that caused the interrupt. Not all models have the protect mode feature.

Illegal Instruction Interrupt

The illegal instruction interrupt cannot be disabled. The interrupt occurs whenever the Processor reads an instruction word containing an operation code that is not one of those permitted by the system. The Processor saves the current Program Status Word in memory locations X'0030' — X'0033'. The new Program Status Word contained in memory locations X'0034' — X'0037' becomes the current Program Status Word.

When the Processor encounters an illegal instruction, it makes no attempt to execute it. The Location Counter of the old Program Status Word contains the address of the illegal instruction.

Supervisor Call Interrupt

This interrupt occurs as the result of the execution of a Supervisor Call instruction. This instruction provides a means for user level (Protect mode) programs to communicate with system programs. The supervisor call interrupt is always enabled. When the Processor executes a Supervisor Call instruction, it:

Saves the current PSW in memory locations X'0096' - X'0099'.

Places the address of the supervisor call parameter block (address of the second operand) in memory locations X'0094' — X'0095'.

Loads the current PSW status with the value contained in locations X'009A' — X'009B'.

Loads the current PSW Location Counter from one of the supervisor call new PSW Location Counters.

System Queue Overflow Interrupt

The termination of an automatic I/O channel operation may cause an item to be added to the system queue. If, at this time, the queue is full, the Processor takes the system queue overflow interrupt. When this occurs, the Processor:

Saves the current PSW in memory locations X'008C' - X'008F'.

Loads the current PSW from the contents of the locations X'0090' - X'0093'.

Saves the item that could not be added to the queue in memory locations X'008A' - X'008B'.

This action allows the software to clear out the queue before any channel I/O terminations are lost. While clearing the queue, external interrupts should be disabled. The queue overflow interrupt cannot be disabled.

Note that, although software routines may use the system queue, and take advantage of the queue service interrupt described previously, the queue overflow interrupt results only when the automatic I/O channel attempts to add to a full queue.

Simulated Interrupt

The Simulate Interrupt instruction simulates a request for service from an external device. When this instruction is executed, the Processor goes through the automatic I/O procedure, using the device address presented in the instruction word. The effect of this instruction may be either an immediate interrupt or the activation of the automatic I/O channel.

STATUS SWITCHING INSTRUCTION FORMATS

The Status Switching instructions use the Register to Register (RR), and the Register and Indexed Storage (RX) instruction formats. In two cases, Load Program Status Word and Simulate Interrupt, the R1 field of the instruction has no significance, and must be zero.

STATUS SWITCHING INSTRUCTIONS

The Status Switching instructions provide for software control of the interrupt structure of the system. They also allow user level programs to communicate with control software. All Status Switching instructions, except the Supervisor Call instruction, are privileged operations.

The instructions described in this section are:

LPSW	Load Program Status Word
EPSR	Exchange Program Status Register
SINT	Simulate Interrupt
SVC	Supervisor Call

INSTRUCTION

Load Program Status Word (LPSW)

Assembler Notation		Op-Code	Format
LPSW	A(X2)	C2	RX

Operation

The 32-bit second operand becomes the current Program Status Word. The second operand is unchanged.

LPSW: PSW(0:15) $\leftarrow$ [A+(X2)]

PSW(16:31) $\leftarrow$ [A+(X2)+2]

Condition Code

Determined by the new PSW

Programming Note

The R1 field of this instruction is not used by the Processor.
The Assembler sets the R1 field to zero.

The quantity to be loaded into the current Program Status Word must be located on a full-word boundary.

This instruction is a privileged operation.

INSTRUCTION

Exchange Program Status Register (EPSR)

Assembler Notation		Op-Code	Format
EPSR	R1, R2	95	RR

Operation

Bits 0:15 of the current Program Status Word replace the contents of the register specified by R1. The contents of the register specified by R2 replace bits 0:15 of the current Program Status Word.

EPSR: PSW (0:15) ⟶ R1

PSW (0:15) ⟵ R2

Condition Code

Determined by the new status

Programming Note

If R1=R2, bits 0:15 of the current PSW are copied into the register specified by R1, but otherwise remain unchanged.

This instruction is a privileged operation.

INSTRUCTION

Simulate Interrupt (SINT)

Assembler Notation	Op-Code	Format
SINT 0, I(X2)	E2	RI

Operation

The least significant eight bits of the second operand are presented to the interrupt handler as a device number. The device number is used to index into the interrupt service pointer table, simulating an interrupt request from an external device. This results in either an immediate interrupt or an automatic I/O operation channel.

Condition Code

Unchanged, if execution of this instruction results in an automatic I/O channel operation with return to the software program.

Determined by the new status, if execution of this instruction results in an immediate interrupt.

Programming Note

The R1 field of this instruction must contain zero.

This instruction is a privileged operation.

INSTRUCTION

Supervisor Call (SVC)

Assembler Notation		Op-Code	Format
SVC	R1, A(X2)	E1	RX

Operation

The address of the second operand replaces the contents of memory locations X'0094' - X'0095'. The current Program Status Word replaces the contents of memory locations X'0096' - X'0099'. The halfword quantity in memory locations X'009A' - X'009B' becomes the new status. The R1 field of the instruction is doubled and added to X'009C'. The halfword value found at the resulting address becomes the new Location Counter.

SVC:

[X'0094'] ← A + (X2)

[X'0096'] ← PSW (0:31)

[X'009A'] → PSW (0:15)

[X'009C'+2* R1] → PSW (16:31)

Condition Code

Determined by the new status

Programming Note

The second operand must be located on a halfword boundary.

CHAPTER 9
INPUT/OUTPUT OPERATIONS

INTRODUCTION

Input/Output operations provide a versatile means for the exchange of information between the Processor, memory, and external devices. Communication between the Processor and external devices is accomplished over the Micro I/O Bus, or Multiplexor Bus. Data transfers to or from external devices may be performed in the byte mode or the halfword mode. Byte and halfword transfers require Processor intervention, either programmed or automatic, for each item transferred.

DEVICE CONTROLLERS

The basic functions of all device controllers are:

To provide synchronization with the Processor and to provide device address recognition.

To transmit operational commands from the Processor to the device.

To translate device status into meaningful information for the Processor.

To request Processor attention when required.

In addition, controllers may generate parity, convert serial data to parallel, buffer incoming or outgoing data, or perform other device dependent functions.

Device Addressing

The system design allows as many as 255 external devices. Each device must have its own unique device number, or address. Device numbers may range from X'01' through X'FF'. (Device number X'00' is not used.)

Processor/Controller Communication

Device controllers are attached directly to the I/O Bus. Communication between the Processor and controllers is a bidirectional, request-response type of operation.

If the Processor initiates the communication, it sends the device address out on the I/O Bus. When a controller recognizes the address, it returns a synchronization signal to the Processor, and remains ready to accept commands from the Processor. The Processor waits up to 15 microseconds for the synchronization signal. If no signal is received in this period of time, the Processor aborts the operation, and notifies the controlling program. Controller malfunction and software failure (incorrect device address) are the most common causes of this type of time-out.

In the other direction, a controller can initiate communication with the Processor. It does this by generating an attention signal. If the Processor is in the interruptable state (bit 1 of the current PSW set) it temporarily suspends the normal, "fetch instruction, execute, fetch next instruction" operation at the end of the execute phase, and transmits an acknowledge signal over the I/O Bus. The controller requesting attention responds with a synchronization signal, and transmits its device number to the Processor. (The acknowledge signal may be automatic or programmed, depending on the current state of the Processor.)

Device Priorities

Requests for attention are asynchronous. Therefore, more than one request may be pending at any time. The system resolves these conflicts according to device priority. The placement of the controllers on the I/O Bus determines their priority. When two or more controllers request attention at the same time, the one closest to the Processor receives the acknowledge signal first, and responds first. Those further down the line must wait until the Processor has acknowledged and acted upon requests from higher priority controllers. Requests for attention remain queued until all have been serviced.

INTERRUPT SERVICE POINTER TABLE

When automatic I/O is enabled (bits 1 and 4 of the current PSW set), device requests for service result in an immediate interrupt.

The interrupt service pointer table is an ordered list containing one entry for each possible device number in the system. The table starts at memory location X'00D0' and extends through X'02CF'. The software controlling I/O operations must set up the table.

When, having acknowledged a request for service, the Processor receives the device address, it adds two times the device address to X'00D0'. The result is the address, within the table, of the entry reserved for the device requesting attention. The entry specifies the address of an appropriate subroutine. The Processor takes an immediate interrupt, and transfers control to the appropriate software routine. If the entry in the table is odd (bit 15 equals one) the Processor activates the automatic I/O channel, without actually interrupting the currently running program.

With the immediate interrupt, at the time the Processor transfers control to the software routine, the old PSW (current at the time of the device request) has been saved at the location specified in the table; the current status has been loaded from the halfword immediately following the old PSW save location; the current Location Counter has been forced to a value equal to the address of the next halfword following the new status.

I/O INSTRUCTION FORMATS

The I/O instructions use the Register to Register (RR), and the Register and Indexed Storage (RX) instruction formats.

I/O INSTRUCTIONS

Following most I/O instructions, the V flag in the Condition Code indicates an instruction time-out. This means that the operation was not completed, either because the device did not respond, or because it responded incorrectly.

In the sense status and block I/O instructions, the V flag can also mean examine status. To distinguish between these two conditions, the program should test bits 0:3 of the status byte. If all of these bits are zero, instruction time-out has occurred.

The instructions described in this section are:

ACK (AI)	Acknowledge Interrupt
ACKR (AIR)	Acknowledge Interrupt Register
SS	Sense Status
SSR	Sense Status Register
OC	Output Command
OCR	Output Command Register
RD	Read Data
RDR	Read Data Register
RH	Read Halfword
RHR	Read Halfword Register
RB	Read Block
RBR	Read Block Register
WD	Write Data
WDR	Write Data Register
WH	Write Halfword
WHR	Write Halfword Register
WB	Write Block
WBR	Write Block Register
AL	Autoload

INSTRUCTION

Acknowledge Interrupt (ACK)
Acknowledge Interrupt Register (ACKR)

Assembler Notation		Op-Code	Format
ACK (AI)	R1,A2(X2)	DF	RX
ACKR (AIR)	R1,R2	9F	RR

Operation

The address of the interrupting device replaces the contents of the register specified by R1. The 8-bit device status replaces the contents of the second operand. The Condition Code is set equal to the right-most four bits of the device status byte. The device interrupt condition is then cleared.

ACK:
- [R1 (8:15)] ← Device number
- [R1 (0:7)] ← Zero
- [A+(X2)] ← Status byte
- [PSW (12:15)] ← Status byte (4:7)

ACKR:
- [R1 (8:15)] ← Device address
- [R1 (0:7)] ← Zero
- [R2 (8:15)] ← Status byte
- [R2 (0:7)] ← Zero
- [PSW (12:15)] ← Status byte (4:7)

Condition Code

C	V	G	L	
X	X	X	1	Device unavailable
X	X	1	X	End of medium
X	1	X	X	Examine status or time-out
1	X	X	X	Device busy

Programming Note

The Condition Code settings described above assume standard Perkin-Elmer device controllers.

These instructions are privileged operations.

The RX form (ACK) is subject to Memory Protect.

The mnemonics for these instructions under the CAL Assembler (03-066) and the CAL 16 Assembler (03-101) have been changed to avoid confusion with the Add Immediate (AI) instruction in the Perkin-Elmer 32-Bit Processor line. The O.S. Assembler (03-025) continues to accept AI and AIR for these instructions, but does not accept ACK or ACKR.

INSTRUCTION

Sense Status (SS)
Sense Status Register (SSR)

Assembler Notation		Op-Code	Format
SS	R1, A(X2)	DD	RX
SSR	R1, R2	9D	RR

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. The device is addressed, and the 8-bit device status is placed in the second operand location. The Condition Code is set equal to the least significant four bits of the device status byte. The first operand is unchanged.

SSR:
[R2 (8:15)] ← Status byte
[R2 (0:7)] ← Zero
[PSW (12:15)] ← Status byte (4:7)

SS:
[A +(X2)] ← Status byte
[PSW (12:15)] ← Status byte (4:7)

Condition Code

C	V	G	L	
0	0	0	0	Acceptable status
X	X	X	1	Device unavailable
X	X	1	X	End of medium
X	1	X	X	Examine status or time-out
1	X	X	X	Device busy

Programming Note

The Condition Code interpretations of status assume standard Perkin-Elmer device controllers.

In the RR format, the device status byte replaces bits 8:15 of the register specified by R2. Bits 0:7 are forced to zero.

These instructions are privileged operations.

The RX form (SS) is subject to Memory Protect

INSTRUCTION

Output Command (OC)
Output Command Register (OCR)

Assembler Notation		Op-Code	Format
OC	R1, A(X2)	DE	RX
OCR	R1, R2	9E	RR

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. The Processor addresses the device and transmits an 8-bit command byte from the second operand location to the device. Neither operand is changed.

OCR: Device ← R2 (8:15)

OC: Device ← [A + (X2)]

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
0	1	0	0	Instruction time-out

Programming Note

In the RR format, bits 8:15 of the register specified by R2 contain the device command.

These instructions are privileged operations.

INSTRUCTION

Read Data (RD)
Read Data Register (RDR)

Assembler Notation		Op-Code	Format
RD	R1, A(X2)	DB	RX
RDR	R1, R2	9B	RR

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. The Processor addresses the device. The device responds by transmitting an 8-bit data byte. This byte is placed in the second operand location.

RD: [A + (X2)] ← Data byte

RDR: [R2 (8:15)] ← Data byte
[R2 (0:7)] ← ZERO

Condition Code

C	V	G	L	
0	0	0	0	Operation Successful
0	1	0	0	Instruction time-out

Programming Note

In the RR format, the 8-bit data byte replaces bits 8:15 of the register specified by R2. Bits 0:7 of the register are forced to zero.

These instructions are privileged operations.

The RX form (RD) is subject to Memory Protect.

INSTRUCTION

Read Halfword (RH)
Read Halfword Register (RHR)

Assembler Notation		Op-Code	Format
RH	R1, A(X2)	D9	RX
RHR	R1, R2	99	RR

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. The Processor addresses the device. If the device is halfword oriented, the Processor transmits 16 bits of data from the device to the second operand location. If the device is byte oriented, the Processor transmits two 8-bit bytes in successive operations.

RH:
[A + (X2)] ← First data byte (8-bit oriented device controller)
[A + (X2)+1] ← Second data byte (8-bit oriented device controller)
or
[A + (X2)] ← Halfword of data (16-bit oriented device controller)

RHR:
[R2 (0:7)] ← First data byte (8-bit oriented device controller)
[R2 (8:15)] ← Second data byte (8-bit oriented device controller)
or
[R2 (0:15)] ← Halfword of data (16-bit oriented device controller)

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
0	1	0	0	Instruction time-out

Programming Notes

In the RR format, the data received from a halfword device replaces the contents of the register specified by R2. The first byte of data from a byte device replaces bits 0:7 of the register specified by R2 and the second byte replaces bits 8:15.

In the RX format, the second operand must be located on a halfword boundary.

These instructions are privileged operations.

The RX form (RH) is subject to Memory Protect.

Not allowed on Micro I/O Bus devices.

INSTRUCTION

Read Block (RB)

Assembler Notation		Op-Code	Format
RB	R1, A(X2)	D7	RX

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. Bits 0:15 of the halfword located at the second operand address contain the starting address of the data buffer. Bits 0:15 of the halfword located at the second operand address plus two contain the ending address of the data buffer.

The Processor transmits 8-bit data bytes from the device to consecutive locations in the specified buffer.

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
X	X	X	1	Device unavailable
X	X	1	X	End of medium
X	1	X	X	Examine status or time-out
1	X	X	X	Device busy

The Condition Code interpretations of status assume standard Perkin-Elmer device controllers.

Programming Note

The starting address must be less than, or equal to, the ending address. If the starting address is greater than the ending address, no transfer takes place, and the Processor forces the Condition Code to zero. If the addresses are equal, one data byte is transmitted.

The Processor is in a non-interruptable state during the transfer.

This instruction is a privileged operation.

This instruction is subject to Memory Protect.

This instruction should not be used with 16-bit oriented device controllers.

When PSW bits 8:11 equal 0000 or 1111, this instruction can be used to transfer up to 64K bytes starting and ending within the first 64KB of memory. When PSW bits 8:11 do not equal 0000 or 1111, this instruction cannot be used to transfer more than 32K bytes, nor can it transfer across a 32K boundary.

INSTRUCTION

Read Block Register (RBR)

Assembler Notation		Op-Code	Format
RBR	R1,R2	97	RR

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. The register specified by R2 contains the starting address of the data buffer. The register specified by R2+1 contains the ending address of the data buffer.

The Processor transmits 8-bit data bytes from the device to consecutive locations in the specified buffer.

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
X	X	X	1	Device unavailable
X	X	1	X	End of medium
X	1	X	X	Examine status or time-out
1	X	X	X	Device busy

The Condition Code interpretations of status assume standard Perkin-Elmer controllers.

Programming Note

The maximum value for R2 is 14.

The starting address must be less than, or equal to, the ending address. If the starting address is greater than the ending address, no transfer takes place, and the Processor forces the Condition Code to zero. If the addresses are equal, one byte is transmitted.

The Processor is in a non-interruptable state during the transfer.

This instruction is a privileged operation.

This instruction is subject to Memory Protect.

This instruction should not be used with 16-bit oriented device controllers.

When PSW bits 8:11 equal 0000 or 1111, this instruction can be used to transfer up to 64K bytes starting and ending within the first 64KB of memory. When PSW bits 8:11 do not equal 0000 or 1111, this instruction cannot be used to transfer more than 32K bytes, nor can it transfer across a 32K boundary.

INSTRUCTION

Write Data (WD)
Write Data Register (WDR)

Assembler Notation		Op-Code	Format
WD	R1, A(X2)	DA	RX
WDR	R1, R2	9A	RR

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. The Processor addresses the device, and transmits an 8-bit data byte from the second operand location to the device. Neither operand is changed.

WD: [A + (X2)] ⟶ Device

WDR: [R2 (8:15)] ⟶ Device

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
0	1	0	0	Instruction time-out

Programming Note

In the RR format, the data byte is taken from bits 8:15 of the register specified by R2.

These instructions are privileged operations.

INSTRUCTION

Write Halfword (WH)
Write Halfword Register (WHR)

Assembler Notation		Op-Code	Format
WH	R1, A(X2)	D8	RX
WHR	R1, R2	98	RR

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. The Processor addresses the device. If the device is halfword oriented, the Processor transmits 16 bits of data from the second operand location to the device. If the device is byte oriented, the Processor transmits two 8-bit data bytes in successive operations.

WH:	[A + (X2)] → Device	8-bit oriented device controller
	[A + (X2)+1] → Device	8-bit oriented device controller
	or	
	[A + (X2)] → Device	16-bit oriented device controller
WHR:	[R2 (0:7)] → Device	8-bit oriented device controller
	[R2 (8:15)] → Device	8-bit oriented device controller
	or	
	[R2 (0:15)] → Device	16-bit oriented device controller

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
0	1	0	0	Instruction time-out

Programming Notes

In the RR format, the data transmitted to a halfword device comes from bits 0:15 of the register specified by R2. The first byte transmitted to a byte device comes from Bits 0:7 of the register specified by R2 and the second byte comes from bits 8:15.

In the RX format, the second operand must be located on a halfword boundary.

These instructions are privlleged operations.

INSTRUCTION

Write Block (WB)

Assembler Notation		Op-Code	Format
WB	R1, A(X2)	D6	RX

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. Bits 0:15 of the halfword located at the second operand address contain the starting address of the data buffer. Bits 0:15 of the halfword located at the second operand address plus two contain the ending address of the data buffer.

The Processor transmits 8-bit data bytes from consecutive locations in the specified buffer to the device.

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
X	X	X	1	Device unavailable
X	X	1	X	End of medium
X	1	X	X	Examine status or time-out
1	X	X	X	Device busy

Programming Note

The Condition Code interpretations of status assume standard Perkin-Elmer controllers.

The starting address must be less than, or equal to, the ending address. If the starting address is greater than the ending address, no transfer takes place, and the Processor forces the Condition Code to zero. If the addresses are equal, one byte is transmitted.

The Processor is in a non-interruptable state during the transfer.

This instruction is a privileged operation.

This instruction should not be used with 16-bit oriented device controllers.

When PSW bits 8:11 equal 0000 or 1111, this instruction can be used to transfer up to 64K bytes starting and ending within the first 64KB of memory. When PSW bits 8:11 do not equal 0000 or 1111, this instruction cannot be used to transfer more than 32K bytes, nor can it transfer across a 32K boundary.

INSTRUCTION

Write Block Register (WBR)

Assembler Notation		Op-Code	Format
WBR	R1,R2	96	RR

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. The register specified by R2 contains the starting address of the data buffer. The register specified by R2+1 contains the ending address of the data buffer.

The Processor transmits 8-bit data bytes from consecutive locations in the specified buffer to the device.

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
X	X	X	1	Device unavailable
X	X	1	X	End of medium
X	1	X	X	Examine status or time-out
1	X	X	X	Device busy

Programming Note

The maximum value for R2 is 14.

The Condition Code interpretations of status assume standard Perkin-Elmer Controllers.

The starting address must be less than, or equal to, the ending address. If the starting address is greater than the ending address, no transfer takes place, and the Processor forces the Condition Code to zero. If the addresses are equal, one byte is transmitted.

The Processor is in a non-interruptable state during the transfer.

This instruction is a privileged operation.

This instruction should not be used with 16-bit oriented device controllers.

When PSW bits 8:11 equal 0000 or 1111, this instruction can be used to transfer up to 64K bytes starting and ending within the first 64KB of memory. When PSW bits 8:11 do not equal 0000 or 1111, this instruction cannot be used to transfer more than 23K bytes, nor can it transfer across a 32K boundary.

INSTRUCTION

Autoload (AL)

Assembler Notation		Op-Code	Format
AL	A(X2)	D5	RX

Operation

The Autoload instruction loads memory with a block of data from a byte oriented input device. The data is read a byte at a time, and stored in successive memory locations starting with location X'0080'. The last byte is loaded into the memory location specified by the address of the second operand. Any blank or zero bytes that are input prior to the first non-zero byte are considered to be leader, and are ignored. All other zero bytes are stored as data. The input device is specified by memory location X'0078'. The device command code is specified by memory location X'0079'.

1. If byte = 0, fetch next byte, otherwise step 2.
2. n ← zero
3. [X'80'+n] ← byte
4. n ← n+1
5. If A+(X2) < X'80' +n, the instruction is finished, otherwise return to step 3.

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
X	X	X	1	Device unavailable
X	X	1	X	End of medium
X	1	X	X	Examine status or time-out
1	X	X	X	Device busy

Programming Note

The R1 field of this instruction must be zero.

The Condition Code interpretations of status assume standard Perkin-Elmer device controllers.

This instruction is a privileged operation.

This instruction is subject to memory protect.

CONTROL OF I/O OPERATIONS

The design of the I/O structure allows data transfers in any of several ways. The choice of which I/O method to use depends on the particular application, and on the characteristics of the external devices. The primary methods of data transfer between the Processor and external devices are:

One byte or one halfword to or from any one of the general registers.

One byte or one halfword to or from memory.

A block of data to or from memory under direct Processor control.

A block of data to or from memory under control of a Selector Channel.

Multiplexed blocks of data to or from memory under control of the automatic I/O channel.

Perkin-Elmer standard device controllers expect a predetermined sequence of commands to effect data transfers. These commands address the device, put it in the correct mode, and cause data to be transferred. Because all I/O instructions are privileged operations, I/O control programs must run in the Supervisor mode, bit 7 of the current PSW reset. I/O control programs should also exercise care in enabling external interrupts.

STATUS MONITORING I/O

The simplest form of I/O programming is status monitoring I/O. In this mode of operation, only one device is handled at a time, and the Processor cannot overlap other operations with the data transfer. The sequence of operations in this type of programming is:

1. Address the device and set the proper mode (Output Command instruction).
2. Test the device status (Sense Status instruction).
3. Loop back to the Sense Status instruction until the status byte indicates that the device is ready (Conditional Branch instruction).
4. When the device is ready, transfer the data (Read or Write instruction).
5. If the transfer is not complete, branch back to the Sense Status instruction (step 2). If it is complete, terminate.

A variation on this type of programming makes use of the block I/O instructions. In this kind of programming, the program prepares the device, and waits for it to become ready. It then executes a block I/O instruction. The Processor takes over control and completes the transfer, one byte at a time, to or from memory. The Processor monitors device status during the transfer. Block I/O instructions may be used only with byte oriented devices whose ready status is zero.

INTERRUPT DRIVEN I/O

Interrupt driven I/O allows the Processor to take advantage of the disparity in speed between itself and the external devices being controlled. With status monitoring, the Processor spends much of its time waiting for the device. With interrupt driven programming, the Processor can use much of this time to perform other functions. This kind of programming establishes two levels of operation. On one level are the interrupt service programs. They usually run with external interrupts disabled. On the other level are the interruptable programs. They run with interrupts enabled.

Automatic Vectoring

The use of the automatic I/O features of the 16-Bit Processor allows hardware vectoring of external interrupts. In this type of programming, the software is relieved of the burden of identifying explicitly the interrupt source. This is done by the hardware through the interrupt service pointer table and the immediate interrupt. Automatic I/O is controlled by bits 1 and 4 of the current PSW.

Before starting operations of this type, the interrupt service pointer table must be set up. This table starts at memory location X'00D0'. It must contain a halfword address entry for every possible device. The value placed in the location reserved for a device is the address of the interrupt service routine for that device. The interrupt service routine must start with a 32-bit old PSW save area. This is followed by a halfword constant that defines the New PSW status. The first instruction of the routine must follow immediately after this constant.

Although there may be gaps in the device address assignments, the interrupt service pointer table should be completely filled. Entries for non-existent devices can point to an error recovery routine. (This precaution prevents system failure in the event of spurious interrupts caused by improper use of the simulate interrupt instruction.)

The next step is to prepare the device for data transfer. This is best done with the external interrupt disabled. Once the table pointer has been set up, and the device prepared, the Processor can move on to an interruptable program.

When the device signals that it requires service, the Processor saves its current state, and transfers control to the interrupt service routine. At this time, the old PSW has been saved in the first two halfword locations of the routine, the new status has been loaded, and the current Location Counter contains the address of the first instruction of the routine. The software routine can now:

1. Save any registers used in the routine.
2. Check the device status, and if satisfactory,
3. Make the data transfer, and
4. Restore the registers, then
5. Return to the interrupted program by reloading the old PSW from its save location.

The interrupt service routine for a device may enable immediate interrupts, provided it first disables interrupts from the particular device being serviced.* Because Perkin-Elmer hardware allows interrupts to be disabled at either the device level or the Processor level, nesting of interrupts is both possible and practical.

*Certain devices do not support this; refer to the specific device manual.

Software Vectoring

Software vectoring of interrupts is provided for compatibility with previous Perkin-Elmer Processors. The Processor reverts to this mode when bit 4 of the current PSW is reset and bit 1 of the current PSW is set.

The software must first set up the new external interrupt Program Status Word in memory locations X'0044' - X'0047'. This new PSW should disable external interrupts by resetting bit 1. The Location Counter of this new PSW contains the address of the interrupt service routine. Upon receipt of the interrupt signal, the Processor saves the current PSW in memory locations X'0040' - X'0043', and loads the new external interrupt PSW. This transfers control to the interrupt service routine which must:

1. Save any registers to be used.
2. Acknowledge the interrupt request to get the interrupting device address.
3. Transfer to an appropriate subroutine, based on the device address.

The subroutine then:

4. Checks the device status, and if satisfactory,
5. Makes the transfer.
6. Restores the registers.
7. Returns to the interrupted program by loading the PSW from location X'0040'.

This method for I/O transfers is not as efficient as is the use of the immediate interrupt. In addition, it is not practical with this method to nest interrupts.

SELECTOR CHANNEL I/O

The Selector Channel controls the transfer of data directly between high speed devices and memory. As many as 16 devices may be attached to the Selector Channel, only one of which may be operating at any one time. The advantage gained in using the Selector Channel is that other processing may proceed simultaneously with the transfer of data between external devices and memory. This is possible because the Selector Channel accesses memory on a cycle stealing basis, which permits the Processor and the Selector Channel to share memory. In some cases, execution times of the program in progress may be affected, while in others, the effect is negligible. This depends upon the rate at which the Selector Channel and Processor compete for memory cycles. For Selch Channel operations, memory is partitioned into four 64KB segments. A Selector Channel transfer can be directed to any 64KB segment. Attempting to cross a 64KB boundary results in wrap-around within the 64KB segment.

The Selector Channel is linked to the Processor over the I/O Bus. It has its own unique device number which it recognizes when addressed by the Processor. Like other device controllers, it can request Processor attention through the external or the immediate interrupt.

Selector Channel Devices

The Selector Channel has a private bus similar to the Processor's I/O Bus. Controllers for the devices associated with the Selector Channel are attached to this bus. When the Selector Channel is idle, its private bus is connected directly to the I/O Bus. If this condition exists, the Processor can address, command, and accept interrupt requests from the devices attached to the Selector Channel. When the Selector Channel is busy, this connection is broken. All communication between the Processor and devices on the Selector Channel is cut off. Any attempt by the Processor to address devices on the channel results in instruction time-out.

Selector Channel Operation

Two registers in the Selector Channel hold the current memory address and the final memory address. Before starting a Selector Channel operation, the control software, using Write instructions, places the address of the first byte of the data buffer in the current register and the address of the last byte of the data buffer in the final address register. The two bits that identify the 64KB Segment of Memory that the start and end address reside in are passed in the Output Command byte that initiates the transfer. During the data transfer, the channel increments the current address register by one for each byte transferred. When the current address equals the final address, the last byte has been transferred, and the channel terminates.

The Selector Channel accesses memory a halfword at a time. Because of this, the transfer must always involve an integral number of halfwords. The starting address of the data buffer must always be on an even byte (halfword) boundary. The ending address must always be on an odd byte boundary. The starting address must be less than the ending address.

Upon termination, the software can read back from the Selector Channel the address contained in the current address register. If this address is less than the final address specified for the transfer, and if the buffer limits were properly checked before the transfer, this condition indicates a device malfunction or an unusual condition within the device, for example, crossing a cylinder boundary on a disc.

Selector Channel Programming

The usual method of programming with the Selector Channel uses the immediate interrupt. The first step in the operation is to check the status of the Selector Channel. If it is not busy, the address of the termination interrupt service routine is placed in the location within the interrupt service pointer table reserved for the Selector Channel. Having done this, the program should proceed as follows:

1. Give the Selector Channel a command to stop. This command initializes the Selector Channel's registers and assures the idle condition with the Selector Channel's private (Selector) bus connected to the I/O Bus.

2. Prepare the device for the transfer with whatever information and commands may be required.

3. Give the Selector Channel the starting and final addresses.

4. Give the Selector Channel the command to start. Bits 6 and 7 of the command byte identify the 64KB segment that the transfer uses.

With the Start command, the Selector Channel breaks the connection between its private bus and the Processor's I/O Bus, and provides a direct path to memory from the last device addressed over its bus. When the device becomes ready, the channel starts the transfer, which proceeds to completion without further Processor intervention. Once the START command has been given, the Processor can be directed to the execution of concurrent programs.

On termination, the channel signals the Processor that it requires service. The Processor subsequently takes an immediate interrupt, transferring control to the Selector Channel interrupt service routine. At this point the software must check the Selector Channel and the device to insure that the transfer was successful.

AUTOMATIC I/O CHANNEL

The automatic I/O channel executes channel programs that control the activities of peripheral devices. The execution of channel programs takes place between the execution of user instructions, and results in a program delay rather than a program interrupt, with an exchange of Program Status Words. The I/O channel may generate an interrupt because of abnormal conditions or because of the occurrence of an event for which the software has requested an interrupt. Bits 1 and 4 of the current Program Status Word control the operation of the I/O channel. Both of these bits must be set to permit channel operations. Bits 8:11 of PSW have no affect on Channel operations. Channel operations also depend on the interrupt service pointer table, the Channel Control Block (CCB) with its associated Channel Command Word (CCW), and the system service queue. See Figures 9-1 and 9-2. Appendix 8 is a flow chart of I/O channel operations.

Interrupt Service Pointer Table

The interrupt service pointer table starts at location X'00D0'. It contains a halfword entry for each of the possible 256 external device addresses. If bit 15 of the entry in this table is ZERO, then the entry is the address within the first 64KB of an immediate interrupt software routine. If bit 15 of the entry is one, then the entry minus one is the address within the first 64KB of a Channel Command Word.

Channel Control Block

The Channel Control Block contains the Channel Command Word, and the storage locations and data required for the channel operation. The Channel Command Word is a bit encoded command that describes the automatic channel operation. Note that it is the address of the Channel Command Word plus one that is placed in the interrupt service pointer table. A complete Channel Control Block is shown in Figure 9-2.

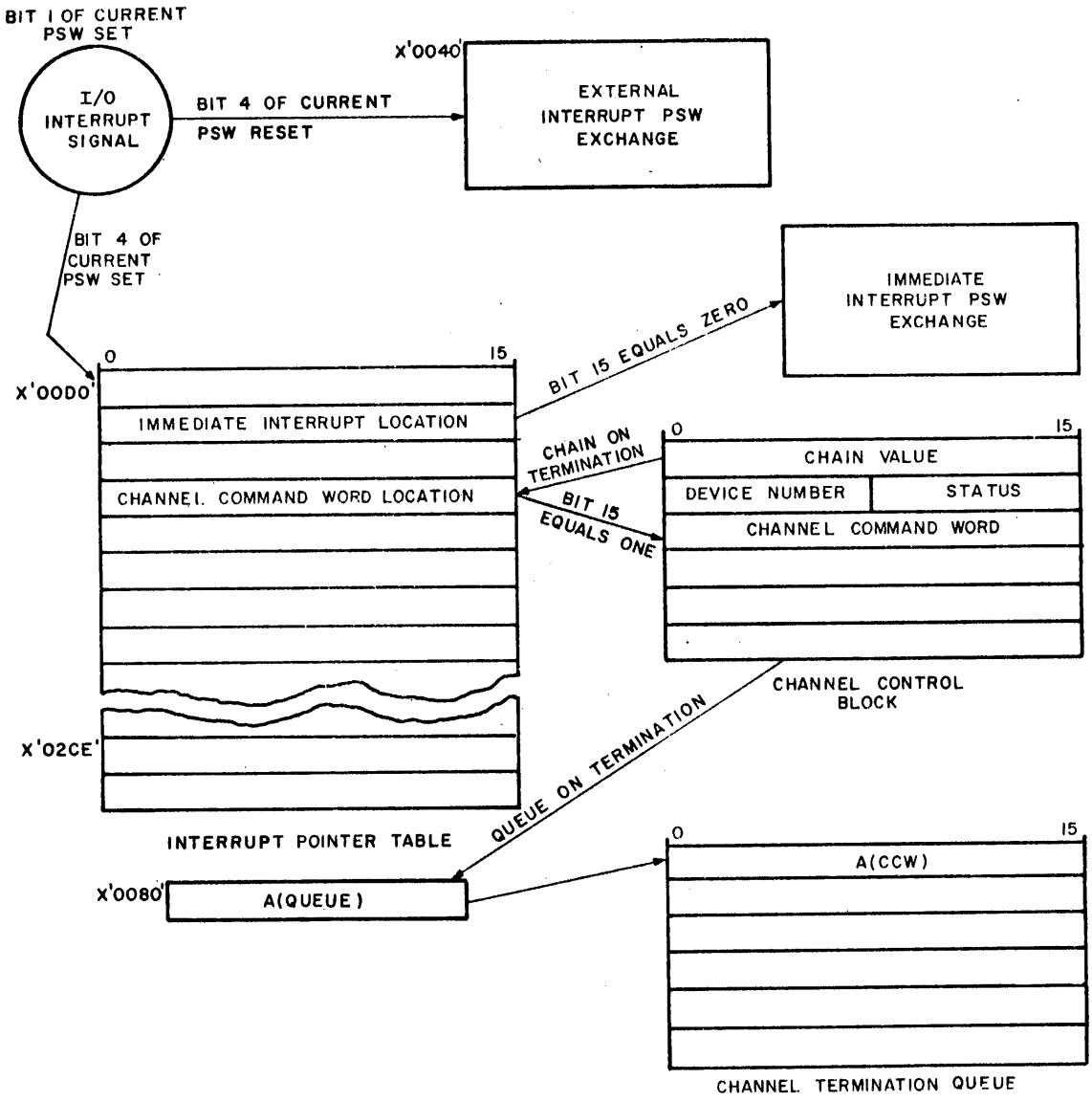


Figure 9-1. I/O Channel Operation Block Diagram

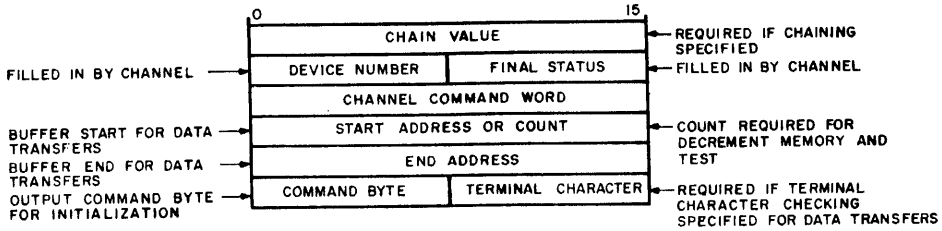


Figure 9-2. Channel Control Block

System Queue

The system queue is a circular list identical to those described for the list processing instructions. The queue may be set up at any convenient location in memory. The maximum size of the queue allows for 255 entries, but any smaller length may be used. (In actual practice, the queue should be big enough to hold one entry for each external device controlled by a channel or software program that makes use of the queue.) The system queue must reside in the first 64KB of main memory. The address of the queue must be placed in memory location X'0080' prior to starting any channel program. The automatic I/O channel uses the queue to record the termination of a channel program.

General Operation

When the Processor receives an interrupt signal from a peripheral device with PSW bits 1 and 4 set, it automatically acknowledges the signal and obtains the address of the device. It uses the device address times two to index into the interrupt service pointer table to the entry reserved for the device. If bit 15 of the entry is ZERO, the Processor takes an immediate interrupt. If bit 15 is one, the Processor takes activity in the I/O channel. In Models not equipped with the automatic I/O channel, the immediate interrupt is taken instead.

The I/O channel uses the entry minus one to locate the Channel Command Word. It decodes the command, and performs the required service, using the data entries in the Channel Command Block as necessary. If the channel operation for this device is not yet complete, the channel returns control to the Processor. The Processor now checks the pending interrupt signals. If any are present, it services them. Otherwise, it resumes program execution.

If the channel determines that the operation for this device is complete, it terminates the channel program by storing the device address and final status in the Channel Control Block, and for data transfers, changes the Channel Command Word to a "no operation". This causes subsequent interrupt signals from the device coming to this Channel Command Word to be ignored. At this point the channel can take any or all of the following actions:

Make an entry on the system queue.

Chain to another Channel Command Word.

Generate an immediate interrupt.

The action taken by the channel depends on the bit configuration of the Channel Command Word.

Channel Command Words

There are three phases involved in channel operations:

1. Initialization
2. I/O operation
3. Termination

All three phases are controlled by the bit configuration of the Channel Command Word. A single command word can be encoded to perform all three types of operation. The bit assignments for Channel Command Words are shown in Figure 9-3.

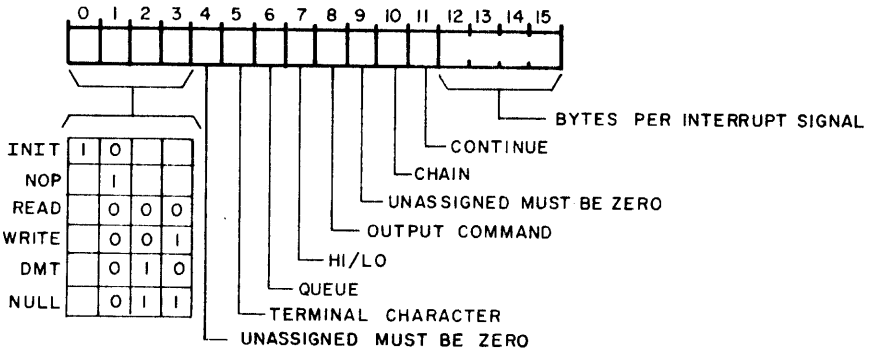


Figure 9-3. Bit Configuration For Channel Command Word

Initialization

Bits 0 (INT) and 8 (Output command) control the initialize phase of channel operations. If bit 0 is set when the channel decodes the command word, it resets bit 0, and checks bit 8. If bit 8 is set, the channel issues the Output command located in the Channel Control Block, and returns control to the Processor. Channel operations with the device resume when an interrupt signal from the device occurs. Since the channel resets bit 0, it can pass through the initialize phase only once. This phase is optional. The software may initialize the device with Output Command instructions prior to starting the channel operation. The bit configuration of the Channel Command Word for the initialize phase is shown in Figure 9-4.

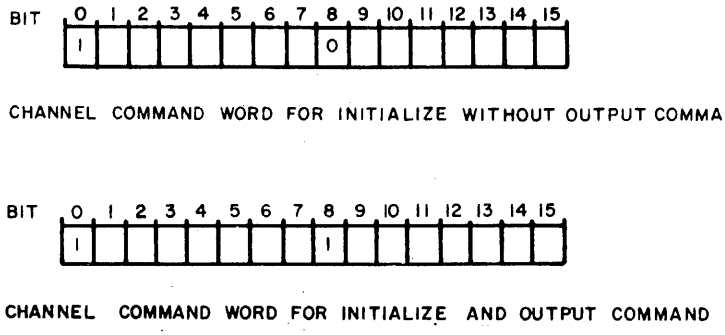


Figure 9-4. Channel command for Initialize and Output Commands

I/O Operations

There are five types of I/O operations that the I/O channel can perform:

Read

Write

Decrement memory and test

No operation

Null operation

The Channel Command Word configurations for these operations are illustrated in Figure 9-5

For all Read/Write operations, Bits 12 through 15 must contain the number of bytes to be transferred on each interrupt signal. (This is usually one or two since Perkin-Elmer standard controllers support byte or halfword transfers.) All zeros in these bit positions indicate that 16 bytes are to be transferred on each interrupt signal. The two halfwords following the Channel Command Word must contain the starting address of the I/O buffer and the ending address of the I/O buffer. After the number of bytes specified for each interrupt signal has been transferred, the starting address is incremented by the appropriate amount and compared to the ending address. If it is greater or equal, the channel enters the termination phase. If it is less, the channel returns control to the Processor for program execution. Bit 5 of the Channel Command Word controls terminal character transfers. When this bit is set, the transfer proceeds as described previously with the exception that the last byte transferred on each interrupt signal is compared with the terminal character byte located in the Channel Command Block. If these two bytes match, the channel enters the termination phase. In this way, a channel program can terminate because a terminal character has been found in the data stream before the buffer is exhausted.

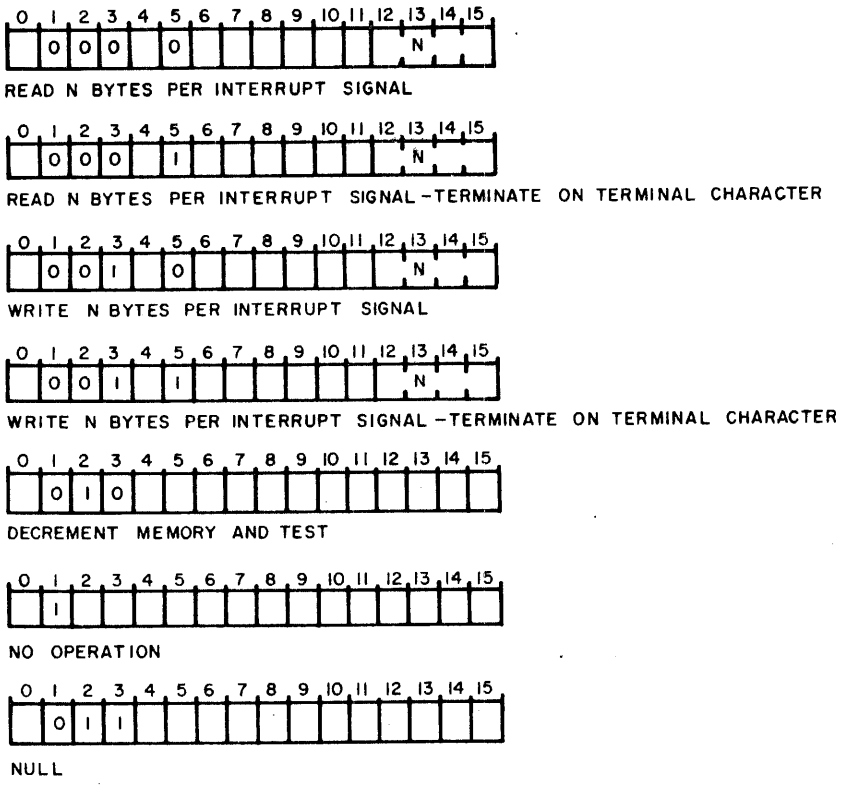


Figure 9-5. Channel Command Words for I/O Operation

Before starting a data transfer, the channel checks the device status. Any non-zero status condition stops the transfer, and causes the channel to enter the termination phase. Before entering the termination phase, the initialize bit and the no operation bit are set in the Channel Command Word; the queue bit is set to force an entry in the system queue; and the chain and continue bits are reset to prevent chaining.

The decrement memory and test operation causes the value contained in the halfword immediately following the Channel Command Word to be decremented by one for each interrupt signal. The new value is compared to zero. If it is greater than zero, the channel returns control to the Processor. If it is equal to zero, the channel enters the termination phase, without changing the Channel Command Word to a "no operation". Subsequent interrupt signals from the device cause the count field to increase negatively.

The no operation code in the Channel Command Word indicates that the channel is to ignore any interrupt signal from the associated device. The channel itself sets this code in the Channel Command Word on the completion of data transfers. The software can use this code to ignore unsolicited interrupt signals.

The null operation differs from the no operation in that, while no I/O function is performed, the channel enters the termination phase without modifying the Channel Command Word.

Termination

The automatic I/O channel enters the termination phase upon completion of a data transfer, when the count field of a decrement memory and test operation has reached zero, or when the null operation is decoded. All of the operations in the termination phase are optional. If none are specified the channel returns control to the Processor. The two termination functions are queue and chain. The Channel Command Word bit configuration for queuing and chaining is shown in Figure 9-6.

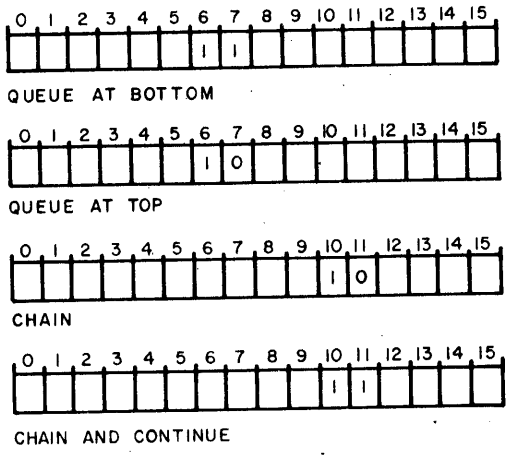


Figure 9-6. Channel Command Words for Termination

Bit 6 controls queuing. If this bit is set, the channel, on entering the termination phase, stores the address of the Channel Command Word in the system queue. The condition of bit 7 of the Channel Command Word controls positioning in the queue. If bit 7 is set, the entry is made at the bottom of the queue. If bit 7 is reset, the entry is made at the top of the queue.

Bit 10 of the Channel Command Word controls chaining. In this operation, the channel stores the contents of the first halfword of the Channel Control Block in the appropriate location in the interrupt service pointer table for the device. This chain value may be either the address of another Channel Command Word, or the address of an immediate interrupt PSW exchange location, If the chain bit (bit 10) and the continue bit (bit 11), are both set, the channel checks the new value placed in the table, and takes appropriate action before returning control to the Processor. In this way, depending on the new value stored in the table, the channel can either generate an immediate interrupt, or start another channel program.

CHAPTER 10
INPUT/OUTPUT SYSTEM

INTRODUCTION

The term interface is used with digital systems to define the junction between two different devices, elements, or pieces of equipment. Interface circuits may perform translation in voltage level, timing, or both.

Digital logic systems operate with a source of input data and an output medium. Inputs may consist of digital or analog signals (i.e., Keyboard, card reader, data set, etc.). Outputs may be a visual display (CRT), or a hard copy terminal (i.e., line printer or Teletypewriter) or control signals. Each signal processed by interfacing hardware must be adequately specified and defined for successful interfacing.

When planning an I/O system to which specific devices must interface, each line of the interface has a dedicated function such as:

Transfer data to or from the processor.
Convey control and timing signals to the peripheral devices.
Transfer status from the peripheral devices to the processor.

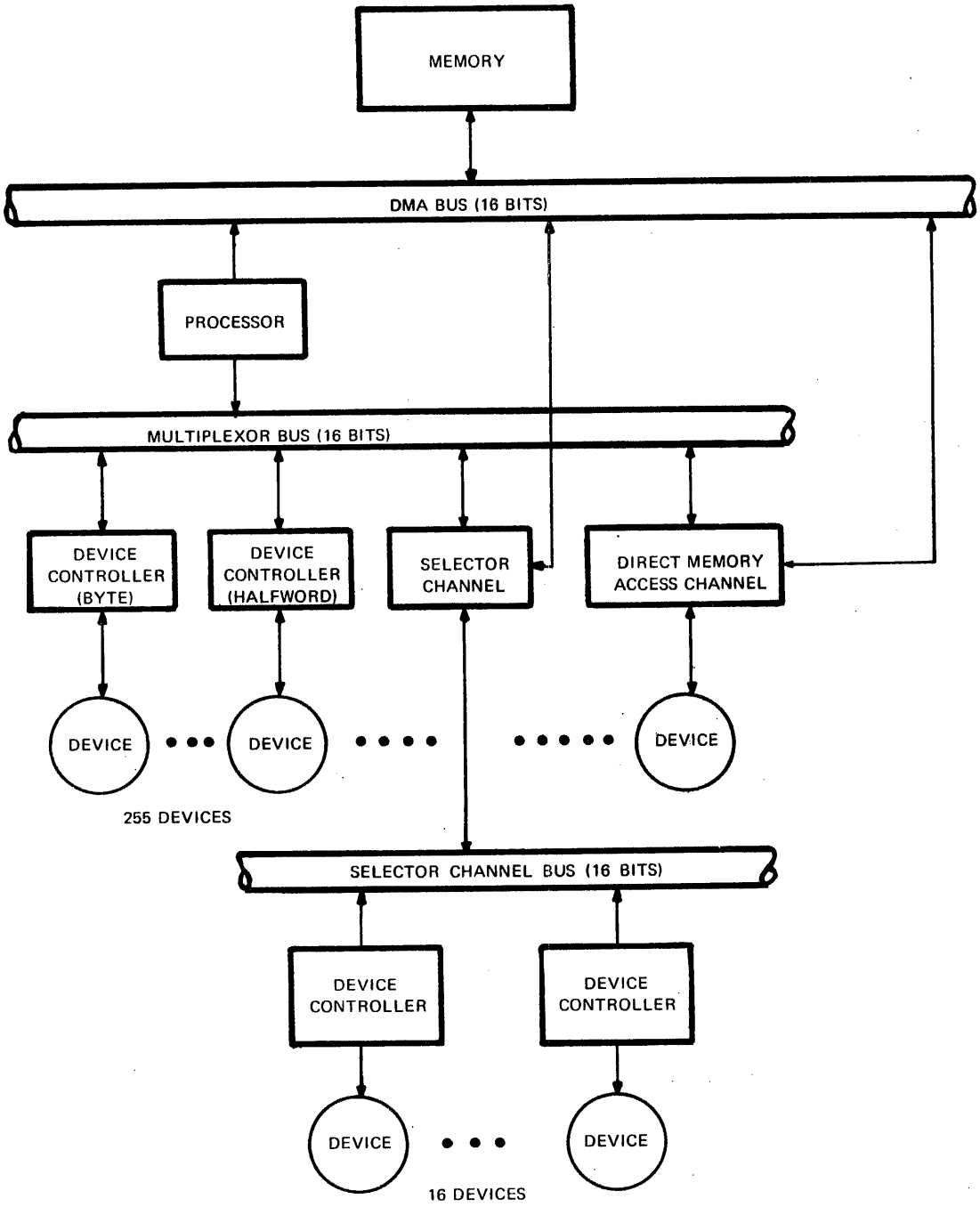
Input/output systems provide communication between the processor and its peripheral devices or other system elements. Methods of communication vary in speed, sophistication, and the amount of attention required by the processor.

There are two methods of interfacing peripheral devices on system elements:

To the Multiplexor (I/O) Bus
To the 16-Bit Extended Selector Channel (ESELCH) I/O Bus.

Figure 10-1 emphasizes the different types of system interface capabilities.

This chapter defines both the electrical and mechanical specifications of Perkin-Elmer's Input/Output System. A functional description of each I/O subsystem follows with a description of the layout and interconnection for a typical system interface. Input/output instruction sequences with considerations and specifications for designing device controllers are discussed.



NOTE

The Multiplexor Bus and the Selector Channel Bus are electrically identical.

Figure 10-1. System Interface, Block Diagram

MULTIPLEXOR BUS

The Multiplexor Bus is a byte or halfword oriented I/O system which can communicate with up to 255 peripheral devices. Perkin-Elmer's complete line of peripheral equipment, can be interfaced to the Multiplexor Bus. The Multiplexor Bus comprises 27 lines – 16 bi-directional data lines, 7 control lines, 3 test lines, and 1 initialize line shown in Table 10-1.

TABLE 10-1. MULTIPLEXOR BUS LINES

FUNCTION	DESIGNATION	DIRECTION
DATA LINES	D00:15	PROCESSOR ⟷ DEVICE
CONTROL LINES	SR DR CMD DA ADRS ACK CL07	PROCESSOR ⟶ DEVICE
TEST LINES	ATN SYN HW	PROCESSOR ⟵ DEVICE
INITIALIZE	SCLR	PROCESSOR ⟶ DEVICE

The following general definitions apply to the Multiplexor Bus lines.

Data Lines (D00:15)

The data lines are used to transfer an 8-bit byte or a 16-bit halfword of data between the processor and the device. An 8-bit device address is transferred from the processor to the device over data lines 08:15 when accompanied by the Address (ADRS) control line. An 8-bit command byte is transferred over D08:15 accompanied by the Command (CMD) control line. One byte or one halfword is transferred from the processor to the device accompanied by the Data Available (DA) control line. The device, in response to an Acknowledge (ACK) control line, sends an 8-bit address to the processor over D08:15, or 8-bits of status information over D08:15 in response to the Status Request (SR) control line. In response to the Data Request (DR) control line, the device sends either an 8-bit byte or a 16-bit halfword of data to the processor.

NOTE

The device always sends a Synchronize (SYN) signal to the processor after it has accepted an operation from the processor. The SYN signal is then removed immediately after the processor removes the control line.

Control Lines

SR	Status Request. The device controller returns device status on D08:15.
DR	Data Request. The device controller returns data to D08:15 or D00:15. If a halfword of data is presented, the Controller also activates the Halfword (HW) test line and returns data on D00:15.
ACK	Acknowledge. The interrupting device controller returns its address on D08:15 if ATN is active.
DA	Data Available. The processor presents data on D00:15 for transfer to the device. The device controller accepts the low byte or the entire halfword and responds with a SYN.
CMD	Command. The processor presents control information to the device on D08:15.
ADRS	Address. The processor presents an 8-bit address on D08:15.
CL070	Early Power Fail Warning. This control line is activated by the processor when a Power Fail condition is detected by the processor. This line is held active until the SCLR0 signal occurs.

Test Lines

ATN	Attention. Any device desiring to interrupt the processor activates the ATN line and holds this line active until an ACK is received from the processor The device controller must not deactivate ATN until the processor deactivates ACK.
HW	Halfword. The HW line is activated by a halfword oriented device controller whenever it is communicating normally with the processor.
SYN	Synchronize. This signal is generated by the device to inform the processor that it has properly responded to a control line.

Initialize Line

SCLR	System Clear. This is a metallic contact to ground that occurs during Power Fail, Power Up, or Initialize.

NOTE

All control lines, except ACK, are connected in parallel to all devices. The ACK line is activated by the processor in response to an external interrupt and is connected in series with all devices. If no interrupt is pending in the first controller when the ACK signal arrives, the signal is passed in daisy-chain fashion to the next controller, and so on until it is captured by the interrupting controller. See definition of ACK.

Communication over the Multiplexor Bus is performed on a request/response basis where each sequence is controlled by the micro-program in the processor's Read-Only-Memory (ROM). A typical sequence to perform an I/O instruction with a device controller is:

1. The processor addresses the device controller by placing an 8-bit address on the data lines and activates the ADRS control line. The device controller whose address corresponds to the 8-bit address on the data lines responds by setting its Address flip-flop and returning SYN to the processor. (All other device controllers reset their Address flip-flops.) Once a device controller is addressed, it remains so until another device is addressed or until the system is initialized. The addressed device controller responds to subsequent activity on the Multiplexor Bus until another controller is addressed.

2. If the I/O instruction involves transferring data from the processor to the device controller, the Processor places the data on the data lines and activates the DA control line. The addressed device controller responds with a SYN after it has received the data, and the processor removes DA.

3. If the I/O instruction involves transferring data to the processor from the device controller, the processor activates the DR control line, and waits for the device controller to respond by placing the data on the data lines then activating SYN. When the processor receives SYN, it accepts the data and removes the DR.

4. In all cases, the device controller removes the SYN whenever the processor removes the control line.

The sequence described here is somewhat simplified for the sake of clarity. The exact sequence for each I/O instruction is listed later.

Whenever a device controller detects an extraordinary condition, it may interrupt the processor by activating the ATN test line. This may be done by any device controller at any time, provided that device interrupts are enabled, regardless of whether it is addressed or not. If interrupts are enabled in the Current Program Status Word, the processor responds to ATN by interrupting the currently running program and directing the processor to a new program (or a new micro-program) which identifies and services the interrupt as required.

I/O SYSTEMS MODULE

Introduction

The Perkin-Elmer Digital System incorporates the Extended Selector Channel (ESELCH) intended to reduce I/O programming requirements, increase throughput, and to add to the flexibility of the Perkin-Elmer Multiplexor Bus. A brief description of this device is given in the following paragraphs.

Extended Selector Channel (ESELCH)

The Perkin-Elmer 16-bit ESELCH provides a high speed Direct Memory Access (DMA) port for block data transfer, bypassing the Processor. The ESELCH generates a private I/O Bus called the SELCH Bus. When the ESELCH is idle, the SELCH Bus is electronically connected to the Multiplexor Bus. However, when the ESELCH is active, the private I/O Bus is disconnected from the Multiplexor Bus.

The ESELCH operates in a Status-Polling mode with the selected I/O device controller on the private I/O Bus. The ESELCH uses the device controllers Busy status bit to control the rate of data transfer, and terminates the data transfer if any of the Status Bits (13, 14, or 15) are set. For additional information on typical handshaking time, refer to the paragraph on I/O Bus Sequence Timing.

The ESELCH transfers data to / from only one device controller during a block transfer. The ESELCH also does not perform interrupt servicing. In interrupts on the private I/O Bus are allowed to queue, but are not gated to the Processor until the completion of the block transfer.

For maximum data throughput rates, the SYNC return delay, in response to a control line, should be minimized. For additional information, refer to the paragraph on Multiplexor Bus Timing.

MULTIPLEXOR I/O DEVICE CONTROLLER LOGIC DESIGN

This section describes the procedures to follow in designing device controllers which connect to the Multiplexor Bus. While it is impossible to describe all possible controllers, this section explains representative circuits in sufficient detail to facilitate design of most controllers.

Multiplexor Bus

The Multiplexor Channel is a byte or halfword oriented I/O system which communicates with up to 191 peripheral devices. The Multiplexor Bus consists of 27 lines: 16 bi-directional Data Lines, 7 Control Lines, 3 Test Lines, and an Initialize Line as described previously.

All busses are false type, i.e., low level is active, high level is inactive. The device controller circuits used to communicate with the Multiplexor Bus are shown in **Figure 10-2.**

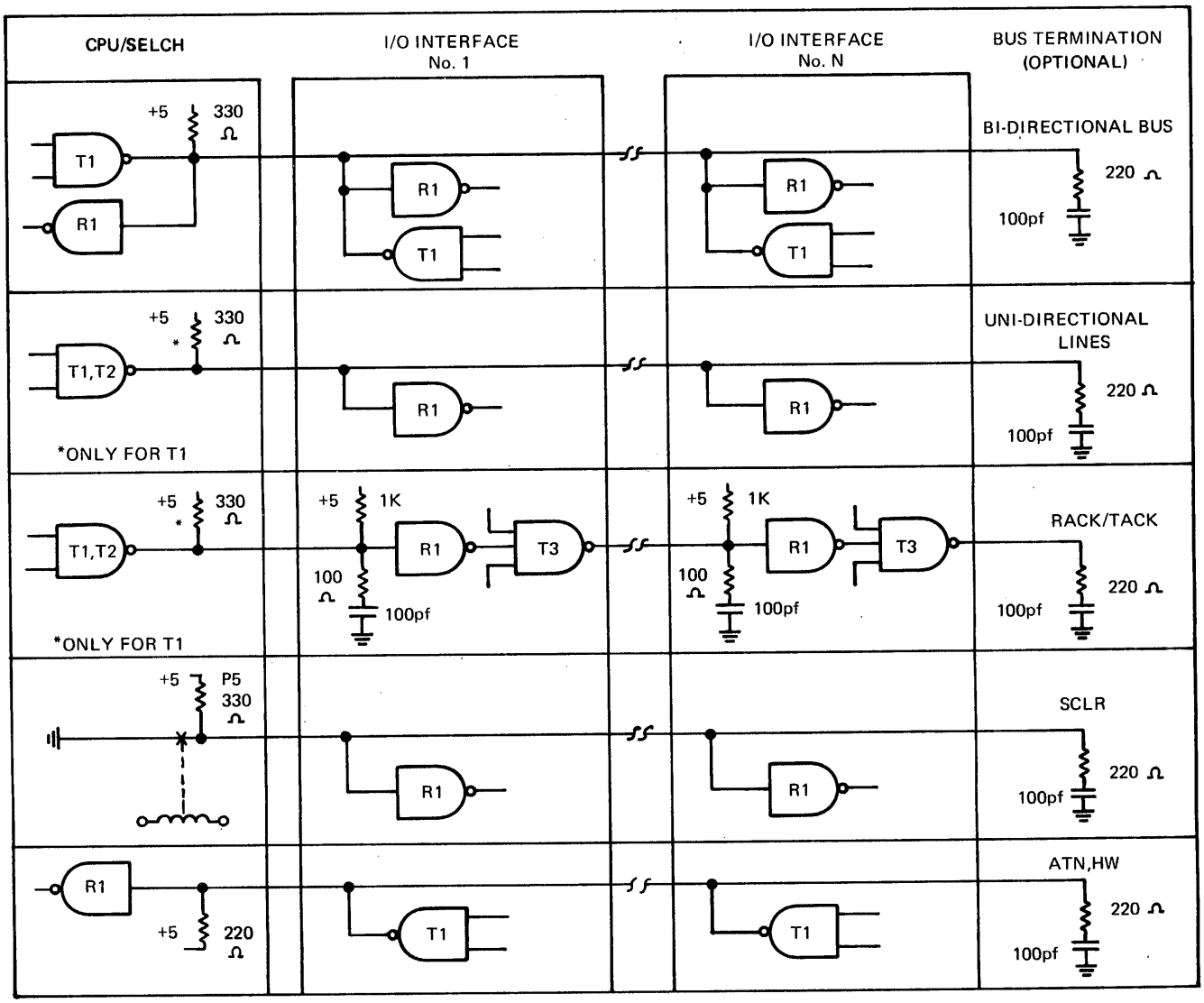
In a typical case, a device controller receives an 8-bit address, an 8-bit Command Byte, and either an 8-bit data byte or a 16-bit data halfword from the processor over the 16 bi-directional Data Lines (D00:15). When only a byte of data is transferred, that byte is passed over the lower eight Data Lines (D08:15). The load resistors for all lines in the Multiplexor Bus are located in the Processor.

Each device controller is permitted one TTL load, 2 milliamperes maximum, on any of the 16 bi-directional Data Lines, the 7 Control Lines, or the single Initialize Line. Each device controller is permitted one high power open collector TTL OR-tied onto each of the 16 bi-directional Data Lines and each of the 3 Test Lines. (The open collector bus driver must be capable of sinking 48 milliamperes at 0.5 VDC maximum VCE. (See Figure 10-2.)

Multiplexor Bus Loading Rules

The Multiplexor Bus, generated at the processor, is capable of driving a total of 16 I/O controllers.

The Multiplexor Bus Buffer or I/O Bus Switch extend the bus drive by regenerating the bus. These devices all represent one load to the bus they are driven by. Each of these devices is capable of driving up to 16 loads.



TRANSMITTER CHARACTERISTICS

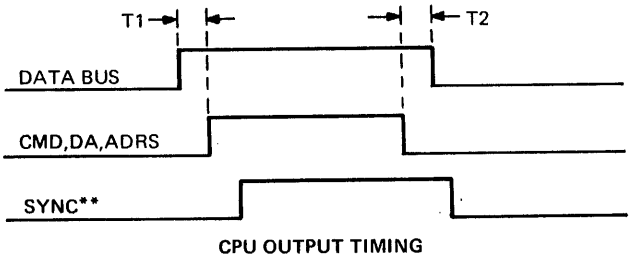
PARAMETER	T1	T2	T3
VOL, LOW LEVEL OUTPUT	0.4V@48 ma.	0.4V@48 ma.	0.4V@20 ma.
VOH, HIGH LEVEL OUTPUT	5.5V max.	2.4V min.	2.4V min.
IOH, HIGH LEVEL LEAKAGE, VOH=5.5V	250 ua.	N.A.	N.A.
tPLH, DELAY, LOW TO HIGH	22ns max.	22ns max.	10ns max.
tPHL, DELAY, HIGH TO LOW	18ns max.	18ns max.	10ns max.

RECEIVER CHARACTERISTICS

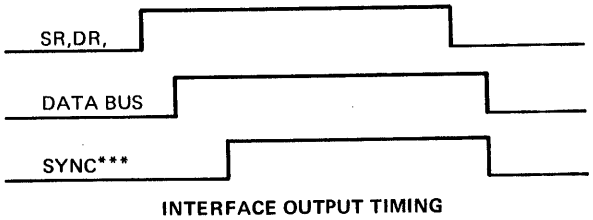
PARAMETER	R1
V_{IH}, INPUT THRESHOLD,HIGH	2.0V MIN
V_{IL}, INPUT THRESHOLD,LOW	0.8V MAX
I_{IH}, INPUT LEAKAGE,HIGH@5.5V	1mA MAX
I_{IL}, INPUT LOW LEVEL, V_{IN}=0.4V	2mA MAX
t_{PLH}, DELAY,LOW TO HIGH	15NS MAX
t_{PHL} DELAY,HIGH TO LOW	12NS MAX

MAXIMUM BUS LOAD (DATA & CONTROL) : 29 mA
MAXIMUM BUS LOAD (TACK, SCLR) : 2 mA

T1 (TYPICAL) 7438
T2 (TYPICAL) 7437
T3 (TYPICAL) 74H10
R1 (TYPICAL) 74H04



CPU OUTPUT TIMING



INTERFACE OUTPUT TIMING

T1 = T2 ≥ 75 ns

** RETURN SYNC AFTER DATA HAS BEEN ACCEPTED

*** RETURN SYNC AFTER DATA HAS BEEN PRESENTED

Figure 10-2. I/O Interface Transmit and Receive Characteristics

Multiplexor Bus Length Restrictions

The processors Multiplexor Bus must be complete within the processor chassis and two adjacent 178 mm (7") expansion chassis. The Multiplexor Bus normally may not be extended to any expansion chassis by use of a cable longer than 102 mm (4"). For this configuration, a bus buffer must be used to extend the bus.

Private I/O Busses, which are generated by a Bus Buffer or I/O Bus Switch, must be complete within the chassis the bus is generated in, and a maximum of two 178 mm (7") expansion chassis. Any private bus may be extended by no more than one 914 mm (36") cable plus an additional cable with a maximum length of 102 mm (4").

Multiplexor Bus Terminators

I/O Terminators (Perkin-Elmer Part Number 35-433 or 35-434) must be installed at the end of the Multiplexor Bus. If the Multiplexor bus is present on both connector 0 and connector 1 of a chassis, terminators must be installed on both sides. If a given Bus extends no more than four adjancent slots in a single chassis, the I/O terminator is optional; however, a terminator should be installed if reflection problems are noted.

Device Controller Addressing

Refer to Figure 10-3 during the following description. When a device controller is addressed, the 10-bit address code is placed on the Data Lines (D060:150). The least significant 8-bits are switch selectable with Perkin-Elmer Part Number 33-032 switches. Bits D06 and D07 generate CHNAD1. If these bits are both ZERO, CHNAD1 is high. This is ANDed with the switch selected 8-bits. The resultant logic function is strobed with ADRS to set or reset the AD flip flop.

The Synchronize (SYN) signal is returned to the processor, during the presence of ADRS1. The ONE output from the Address flip-flop, AD1, is used to gate all other I/O control lines to the device controller. When a different device is addressed, the ADRS1 strobe line resets the AD flip-flop thus inhibiting further communications between the Processor and controller. Thus, only one device controller may be addressed at any time. During the address cycle, only the device which was addressed returns a SYN.

NOTE

The device controller must be designed such that when some other device is addressed, the previously addressed controller clears its Address flip-flop in no more than 350 nanoseconds after the receipt of ADRS. Otherwise the system could have two devices addressed simultaneously.

The device controller logic must delay SYN until it has reacted to the Multiplexor Bus control line, however, unnecessarily long delays reduce the system input/output operation.

NOTE

If the device controller is a 16-bit halfword-oriented controller, the Halfword Enable line (HW0) is activated while its Address flip-flop is set, if the interface is strapped for Halfword mode. The HW0 is used by the processor to determine if the device is capable of sending or receiving 16-bit halfword data in parallel.

Interrupt Control

Figure 10-3 shows a complete general purpose interrupt and interrupt acknowledge logic system. When an interrupt is generated, the ATN flip-flop is set. The output from the ATN flip-flop generates an Attention signal (ATN0) to the processor. The program responds with an Acknowledge Interrupt signal, which is received by the controller as Receive Acknowledge (RACK). Since the ATN flip-flop was set prior to receiving RACK, ATSYN0 goes low, and the Transmit Acknowledge (TACK0) output is held high. Thus, the Acknowledge Interrupt signal is captured by the interrupting controller, and is prevented from propagating to the next device as Receive Acknowledge (RACK0). The ATSYN0 signal active causes the device address to be returned on the Multiplexor Bus Data Lines D080-150 when input number three is selected on the 4:1 multiplexor, and then causes a SYN to be returned to the Processor. On receiving the SYN0, the processor deactivates RACK0, causing the ATN flip-flop to reset, deactivating ATN0. The device controller must not deactivate ATN0 until the processor deactivates RACK0.

NOTE

If the ATN flip-flop is reset, the RACK1 signal passes through the device to TACK0, and on to the next device.

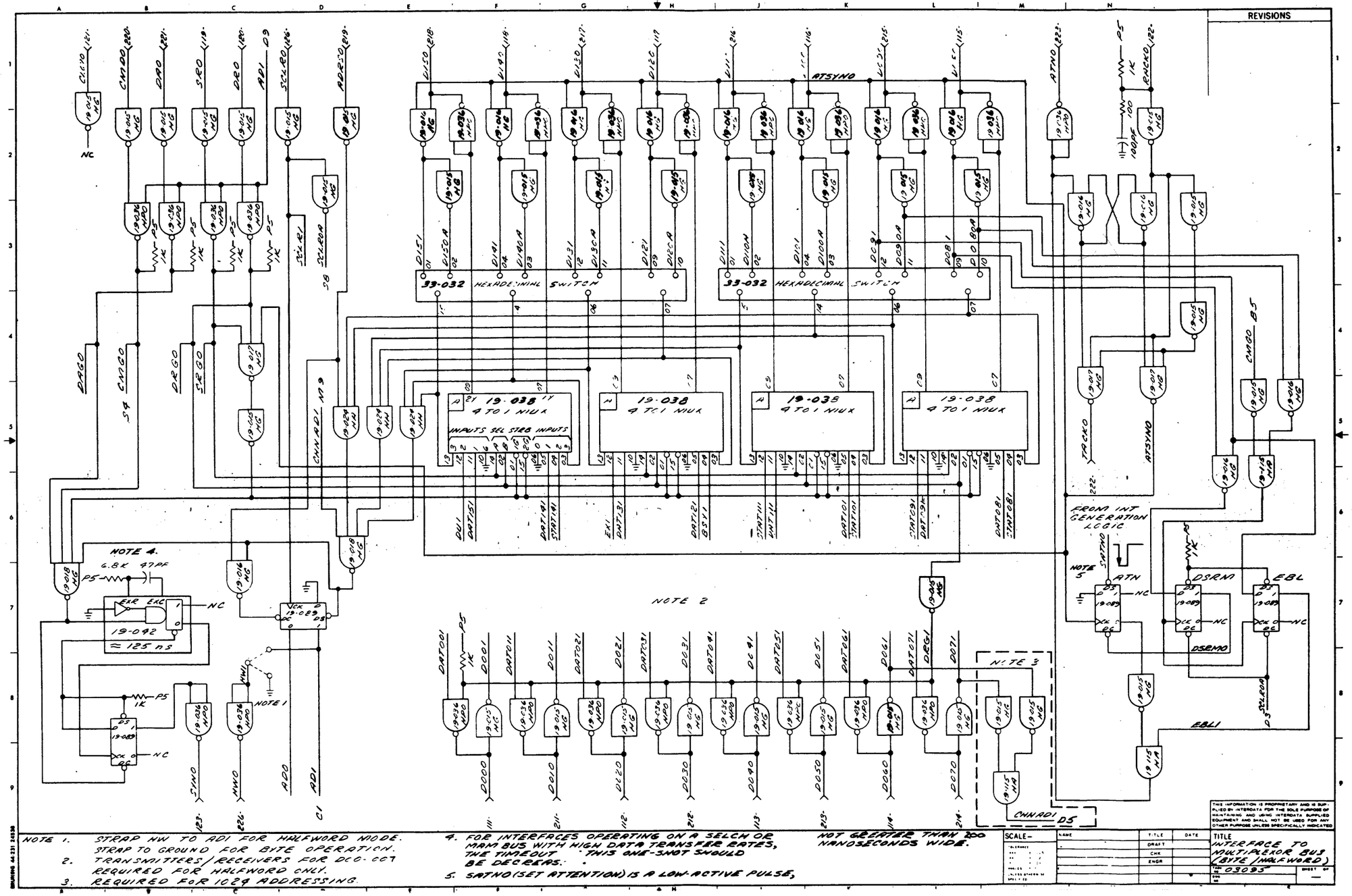


Figure 10-3. General Interface to Multiplexor Bus (Byte or Halfword Oriented)

The EBL and DSRM flip-flops provide control over the interrupt ATN flip-flop and the ATN control line to the processor. Two bits of the Command Byte (bits 8 and 9) are decoded as one of three possible functions: Enable, Disable, or Disarm.

A command with bit 8 reset (ZERO) and bit 9 set (ONE) causes interrupts to be Enabled. The output from the DSRM flip flip-flop (DSRM0) is high, allowing the ATN flip-flop to be set by a SATN0 pulse, and the output from the EBL flip-flop (EBL1) is high, enabling the output from the ATN flip-flop to activate ATN to the Processor.

A command with bit 8 set (ONE) and bit 9 reset (ZERO) causes interrupts to be Disabled. The output from the DSRM flip-flop (DSRM0) is high, allowing the ATN flip-flop to be set by a SATN0 pulse, however, the output from the EBL flip-flop (EBL1) is low, preventing the ATN flip-flop from activating ATN. Thus, interrupts are allowed to queue, but are not passed to the Processor.

A command with both bits 8 and 9 set (ONEs) causes interrupts to be Disarmed. The output from the DSRM flip-flop (DSRM0) is low, forcing the clear input to the ATN flip-flop low, thus disallowing an interrupt to queue. Note that the output from the EBL flip-flop is a don't-care condition when DSRM0 is low.

A command with both bits 8 and 9 reset (ZERO) causes no change in the interrupt controls. This condition prevents the command pulse from loading the DSRM and EBL flip-flops.

As described previously, RACK from the processor is the Interrupt Acknowledge (ACK) signal. This line breaks up into a series of short lines to form the daisy-chain priority system. The RACK signal must pass through every device controller that is equipped with Interrupt Control circuits. This implies that the device controller's interrupt priority in the Perkin-Elmer system is determined by the physical location within the system. That is, the controller nearest to the Processor (first in line in the ACK daisy-chain) has the highest priority.

At any I/O slot, the Received ACK (RACK0) appears at Pin 122-1 and the Transmitted ACK (TACK0) at Pin 222-1. The daisy-chain bus is formed by a series of isolated lines which connect Terminal 222-1 of a given position to Terminal 122-1 of the next position (lower priority). On unequipped positions, a back panel jumper shorts 122-1 and 222-1 on the same connector to complete the bus. Back panels are wired with jumpers on all I/O positions. Whenever a card chassis position is equipped with a device controller that has interrupt circuits, the jumper from 122-1 to 222-1 must be removed from the back panel at that position.

For controllers that occupy several positions, the jumper is removed only at the position where the controller board has interrupt circuits.

Multiplexor Bus Wiring

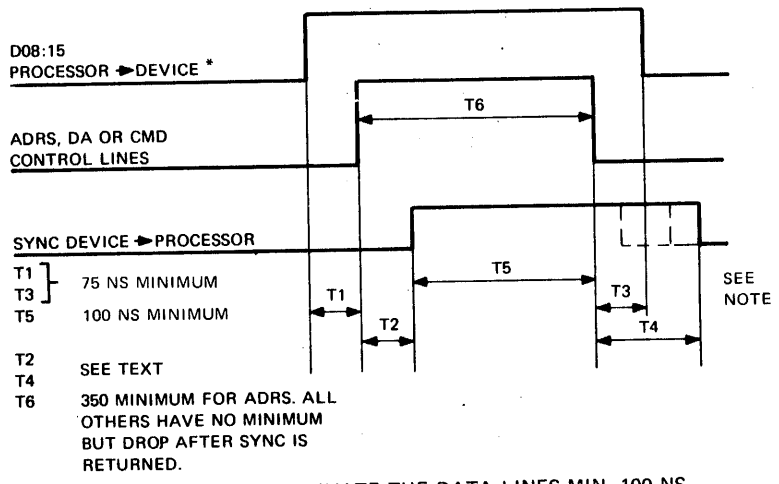
Wiring for the Multiplexor Bus and the Selector Channel Bus is identical in the processor and Expansion chassis. Each card position contains two connectors with the Multiplexor Bus wired to each at pin positions indicated in Figure 10-16

Multiplexor Bus Timing

Both the Input and Output operations on the Multiplexor Bus make use of request/response signaling. This allows the system to run at its maximum speed whenever possible, but permits a graceful slowdown if the characteristics of a particular device controller require signals of longer duration. Device controller designs should keep Multiplexor Bus usage as fast as possible, consistent with practical circuit margins. Doing this assures the fastest computer input/output operation when a system is configured with many peripheral devices.

Timing for typical Output operations is shown on Figure 10-14. On the Output operation, the processor places a signal on the data lines followed by an appropriate control line signal. This stagger (T1) varies, but it is guaranteed to be at least 75 nanoseconds. When the device controller has received the Output Byte, the SYN signal is returned to the processor, which terminates the control line signal. Realizing that T5 is 100 nanoseconds minimum, the SYN delay T2 should be only long enough to guarantee proper reception of the Output Byte. The control line/data line removal time (T3) is important where single-rail to double-rail operation is used, e.g., the ADRS flip-flop of Figure 10-3. A minimum of 75 nanoseconds is guaranteed for T3. For SYN generation as per Figure 10-4 the control line signal is DC coupled through the gates to form the SYN signal. The SYN removal time (T4) should be minimized. This delay should not be extended unnecessarily since the processor does not begin another Input/Output operation until SYN is removed.

It should be emphasized that the times shown on Figure 10-4 are defined for signals on the Multiplexor Bus. Within a given device controller, one signal may flow through more gates than another signal and these delays must be considered.



* THE PROCESSOR MUST DEACTIVATE THE DATA LINES MIN. 100 NS BEFORE ACTIVATING ANY CONTROL LINE FOR THE NEXT I/O OPERATION.

Figure 10-4. Multiplexor Bus Output Timing

NOTE

The time between the completion of one I/O operation and the start of the next I/O operation is undefined. In certain cases, there is no delay between consecutive I/O operations. The device controller must be ready to respond immediately.

Timing for typical Input operations is shown on Figure 10-5. For the Input operation, the processor activates a control line. The currently addressed device controller should gate signals to the data lines as soon as possible to keep T1 at a minimum. The SYN delay (T2) must guarantee that the Input Byte is on the data lines, considering the slowest data gates and the fastest SYN gates. The processor removes the control line signal when SYN is received with a minimum delay (T4) of 100 nanoseconds. With SYN and the byte gate DC coupled to the control line, the removal delay (T3) is the sum of the corresponding gate delays. The processor considers the operation complete when SYN deactivates.

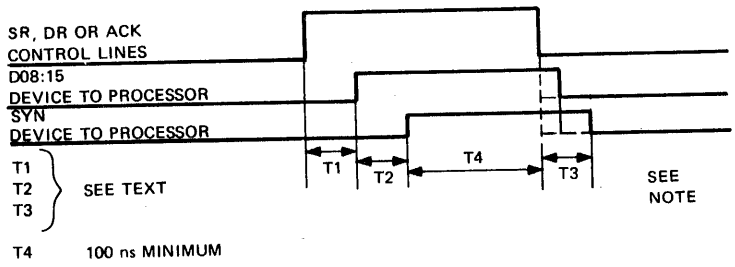


Figure 10-5. Multiplexor Bus Input Timing

NOTE

The time between the completion of one I/O operation and the start of the next I/O operation is undefined. In certain cases, there is no delay between consecutive I/O operations. The device controller must *never* hold the data lines any longer than necessary.

When the control signal is ACK, the delay (T1) includes the cumulative gate delays (see Figure 10-3) for all the device controllers between the responding device controller and the processor. This is less than the processor time-out even with the maximum limit of 255 device controllers.

NOTE

With a SYN delay of 50 nanoseconds, device controllers must be designed to accept a minimum width of 170 nanoseconds on all control line pulses except ADRS which is guaranteed to be 350 nanoseconds minimum. The SYN delay in the device controller may be increased to effectively lengthen the control line pulses if it is absolutely necessary. It is essential to realize that, after the processor initiates an I/O operation, the processor does nothing until the SYN signal is returned by the device controller; one or more processor clock cycles may be skipped if necessary and the data throughput decreased proportionally. While this may not affect a particular device controller, the overall system performance is degraded. Furthermore, if a device controller fails to respond with a SYN within the time out period of approximately 15 to 35 microseconds, the processor aborts the I/O operation and removes the control line signal.

General Multiplexor Bus Interface

Figure 10-3 illustrates a general interface to the Multiplexor Bus which may be used when designing custom device controllers, either 8-bit byte or 16-bit halfword oriented. (If an 8-bit byte oriented interface is being designed, gates connecting to D001:071 and to DAT001:071 can be eliminated.) The figure illustrates an interface with the 74H logic family; however, other logic families may be used provided they conform to the characteristics in Figure 10-2 The address straps can be hardwired by the user for any device number from X'002' to X'3FF' with the exception of X'007', which is reserved for the manually selected clock. The user can use the Gated Status Request (SRG0) or the Gated Data Request (DRG0) control lines to gate status or data from appropriate points in his logic. Data from the processor is available to the user's circuits, double rail, at the points labeled D001:151 and D000A:150A. The user can use the Gated Data Available (DAG0) and the Gated Command (CMG0) control lines to gate the data from the processor to appropriate points in his logic. The delay of the SYN0 signal should be arranged such that it is the minimum delay necessary for the custom controllers to function properly, per the Multiplexor Bus Timing Section.

Additional Requirements for I/O Interface:

1. +5 volts ± 5% is assumed for all interface designs. Other voltages from the system power supply should not be used in the design of the interface. If voltages other than +5 volts are required in the design of the interface, an on-board DC to DC converter may be used. (See DC-DC Converter Specifications.)

2. When appropriate, use 16-bit I/O for devices.

3. In general, all unused inputs of the logic packages, flip-flop (F/F), counter, etc., should be terminated to +5 volts through a pull-up resistor (1K ohm). A maximum of 25, 74H gate loads can typically be connected to one pull-up resistor.

4. Avoid the use of capacitors for delaying the edge of a logic signal. Use a one-shot 19-042, 19-031, etc., or delay lines for delays.

5. The interface should be designed with the assumption that all data transferred to and from the processor is valid when the BSY status bit (BSY1) changes from a logic 1 to a logic 0.

6. Avoid the use of RC networks for differentiation of logic signal edges. Use a one-shot 19-042, 19-031, etc., for differentiation. The timing components used on these one-shots must be immediately adjacent to the IC.

7. High-frequency decoupling capacitors should be located adjacent to ICs as required. The number of decouplers required is a function of the logic family being used – 74S, 74LS, etc. As a minimum, there should be one decoupler for each two ICs. Provide extra decoupling and/or isolated P5 and GND connections for high-current driver ICs.

8. In general, all I/O device controllers which have a data transfer rate greater than 10 K bytes/second are supported by a Selector Channel or Extended Selector Channel (SELCH or ESELCH).

9. All design rules and requirements of an I/O device controller to operate with Perkin-Elmer Multiplexor Bus structure are applicable for device operation with a Selector Channel.

10. It is good design practice, in cases where address, command, or data is loaded from the Multiplexor Bus into an edge-triggered flip-flop, to accomplish the loading operation on the leading edge of the appropriate control signal (i.e., coincident with the high-to-low transition of ADRS0, CMD0, or DA0).

11. Device controllers which have more than one device address, should have contiguous addresses (i.e., X'A0', 'A1', 'A2', 'A3').

12. Device controller addresses must not be restricted to a single fixed address or fixed range of addresses and must decode ten address bits.

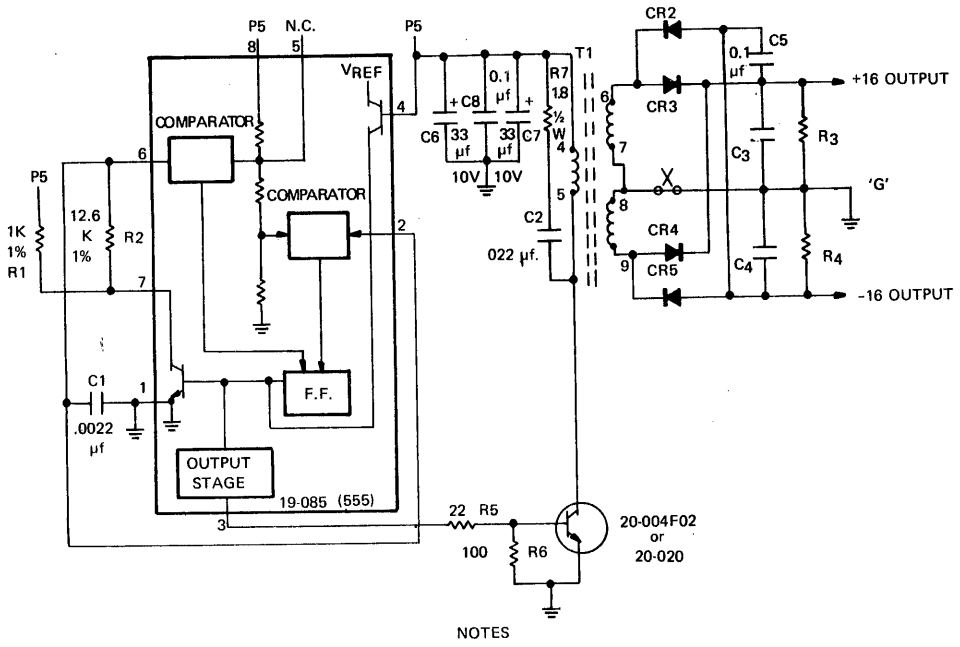
13. The 381 mm x 381 mm (15" x 15") controllers are limited to an absolute maximum of 13 amperes of P5 power, and the 178 mm x 381 mm (7" x 15") controllers are limited to an absolute maximum of 7 amperes of P5 power. These limitations are imposed by the standard Perkin-Elmer back panel connector.

14. In cases where command information is loaded into a level-triggered or R-S flip-flop by the presence of CMG0, the I/O interface must allow for wide variation of the width of the CMG0 pulse.

DC-DC Converter Specifications

Functional Use

The circuit shown in Figure 8-23 may be used to provide a means of developing low current voltage levels to operate Teletypewriter (TTY) I/O interfaces, operational amplifiers, etc., from on-board +5 volts logic power.



NOTES

1. ALL DIODES 23-001
2. ALL RESISTORS ¼W. + 5% TOLERANCE UNLESS OTHERWISE SPECIFIED
3. $C_3 = C_4 = 15\ \mu f\ 20V$
4. $R_3 = R_4 = 3.9K$
5. T1 = 30-020

Figure 10-6. DC/DC Converter

General Description

The circuit is comprised of a 19-085 IC timer connected as a 24K Hz oscillator having a square wave output, which drives a switching transistor feeding a step-up transformer. The output from the transformer is rectified and filtered to provide +16 and -16 volts at 50 milliamperes each, or optional levels as listed in the following specifications.

Specifications

Input +5 VDC ±5% @ 200 to 500 milliamperes depending on output load.

Output: Standard circuit supplies + and - 18 volts at no load (with 5 volt input) decreasing to + and -16 volts at full load (100 milliamperes total sum of either or both polarity).

Output Power Options:

1. Option 1 supplies +18V only at no load, decreasing to +16 volts at full load (100 milliamperes).

2. Option 2 supplies -18V only at no load, decreasing to -16 volts at full load (100 milliamperes).

3. Option 3 supplies +36V only at no load, decreasing to +32 volts at full load (50 milliamperes).

4. Option 4 supplies -36V only at no load, decreasing to -32 volts at full load (50 milliamperes).

Output Ripple: Standard circuit and Options 1 and 2 = 150 millivolts peak-to-peak (p/p) maximum -- Options 3 and 4 = 300 millivolts p/p maximum.

Line Regulation: Output varies directly as input.

Operating Frequency: Approximately 24K Hz.

Start-up Time: Output voltage is up to nominal level within 200 milliseconds after power is applied.

Output Configuration Options. Refer to Figure 10-6

1. For single polarity +16 volts output (full load functional variations), delete CR2, CR5, C4, and R4.

2. For single polarity -16 volts output (full load), delete CR3, CR4, C3, and R3.

3. For single polarity +32 volts output (full load), open ground connection to transformer at point X, delete ground point G, and ground -16 volts output.

4. For single polarity -32 volts output (full load), open ground connection to transformer at point X, delete ground point G, and ground +16 volts output.

NOTE

If voltage levels other than those supplied by the converter are needed, or close regulation is required, an IC regulator such as a 19-094 can be attached to the output of the DC to DC Converter.

The recommended format for the Status Byte of an I/O device controller is shown in Figure 10-7. There may be some exceptions to the status format, depending upon the function of the I/O device controller.

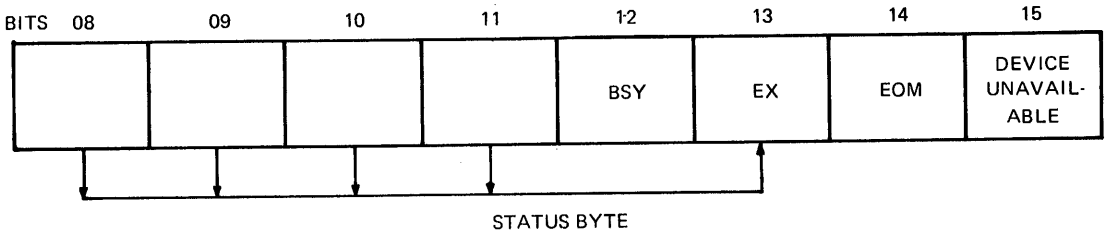
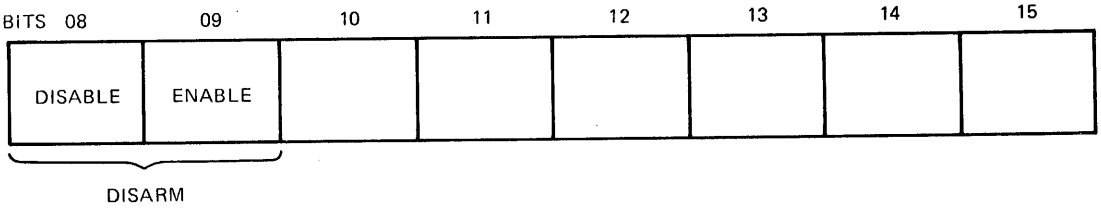


Figure 10-7. Status Byte

At least one of Bits 08:11 shall be set when the Examine bit (EX) is set. In many cases, all of the upper 4-bits (Bits 08:11) of the Status Byte may be used to set the Examine bit depending upon the function of the device controller.

The standard Output Command Byte is shown in Figure 10-8. Bits 10:15 are used for control functions of the device controller. Bits 08 and 09 must be used to control the interrupt circuit of the device controller. The meaning of Bits 08 and 09 are as shown on Figure 10-8.



BITS		MEANING
08	09	
1	0	DISABLE INTERRUPT FUNCTION (QUEUE INTERRUPTS)
0	1	ENABLE INTERRUPT FUNCTION (PASS INTERRUPTS TO PROCESSOR)
1	1	DISARM INTERRUPT FUNCTION (DO NOT QUEUE INTERRUPTS)
0	0	NO CHANGE IN THE INTERRUPT CONTROLS

Figure 10-8. Command Byte

Data and Status Input

Data

Figure 10-3 shows how a byte or halfword of data may be read into the processor. When the byte-oriented device controller is addressed, AD1 is high, enabling the Data Request (DR) control line from the Processor. (The HW1 is strapped to ground for byte operation.) The DR enables the data byte onto the eight bottom Data Lines D080:150. If a halfword-oriented device controller is addressed, 16 bits are gated onto Data Lines D000:150 since the halfword input is strapped to a high AD1 output from the Address flip-flop, enabling the Halfword mode. A system requirement is that the addressed controller must respond to all control lines (i.e., Data Request) with a SYN.

Status

Figure 10-3 shows how a byte of status may be read into the processor. When the byte or halfword oriented device controller is addressed, AD1 is high, enabling the Status Request (SR) control line from the processor. Open collector gates are used for OR tying the 4:1 multiplexor onto the eight Data Lines (D080:150).

The device controller logic should place a high on BSY1 until the data is ready. The processor may now be synchronized to the device data rate by testing the device status until the Busy bit is low. When the Busy bit is low, the program may transfer data. Device synchronization can also be achieved by generating an interrupt when the data is ready.

The End of Medium (EOM) bit is normally placed active at the termination of the device medium, such as End of Tape. The Device Unavailable (DU) bit, when active, typically signifies that device power is not turned on.

The Examine Status (EX) bit is used to signify other appropriate device conditions. In this case, the user assigns D08 through D11 to appropriate conditions, such as Parity Error, etc.

It is appropriate to note here that the Busy Status is unconditionally defined such that data cannot be transferred unless Busy is inactive. The remaining status bits are defined as required by the device controller. Not all device controllers require all eight status bits.

Data and Command Output

Data

Figure 10-3 shows how a data byte may be output from the processor. The buffered true and false Data Lines (D081:151 and D080:150) connect to the set and reset inputs of the Data Register.

When the device is addressed, AD1 is high, enabling the control line DAG0 to return the SYN signal to the processor.

Command

The command lines are shown on Figure 10-3 as being used to enable/disable interrupts, etc.

Again, note that definition of the command bits is a function of the device controller only. Not all device controllers require eight separate bits.

Byte-Oriented Device Controller Design

A byte-oriented device controller may be designed to accommodate Halfword Data Transfer instructions (RH/RHR and WH/WHR). This allows slightly more efficient I/O programming (one user-level instruction instead of two) for each two bytes of data transfer. To accommodate these instructions, the device controller must be designed to transfer data in accordance with the I/O sequence shown on Figure 10-9

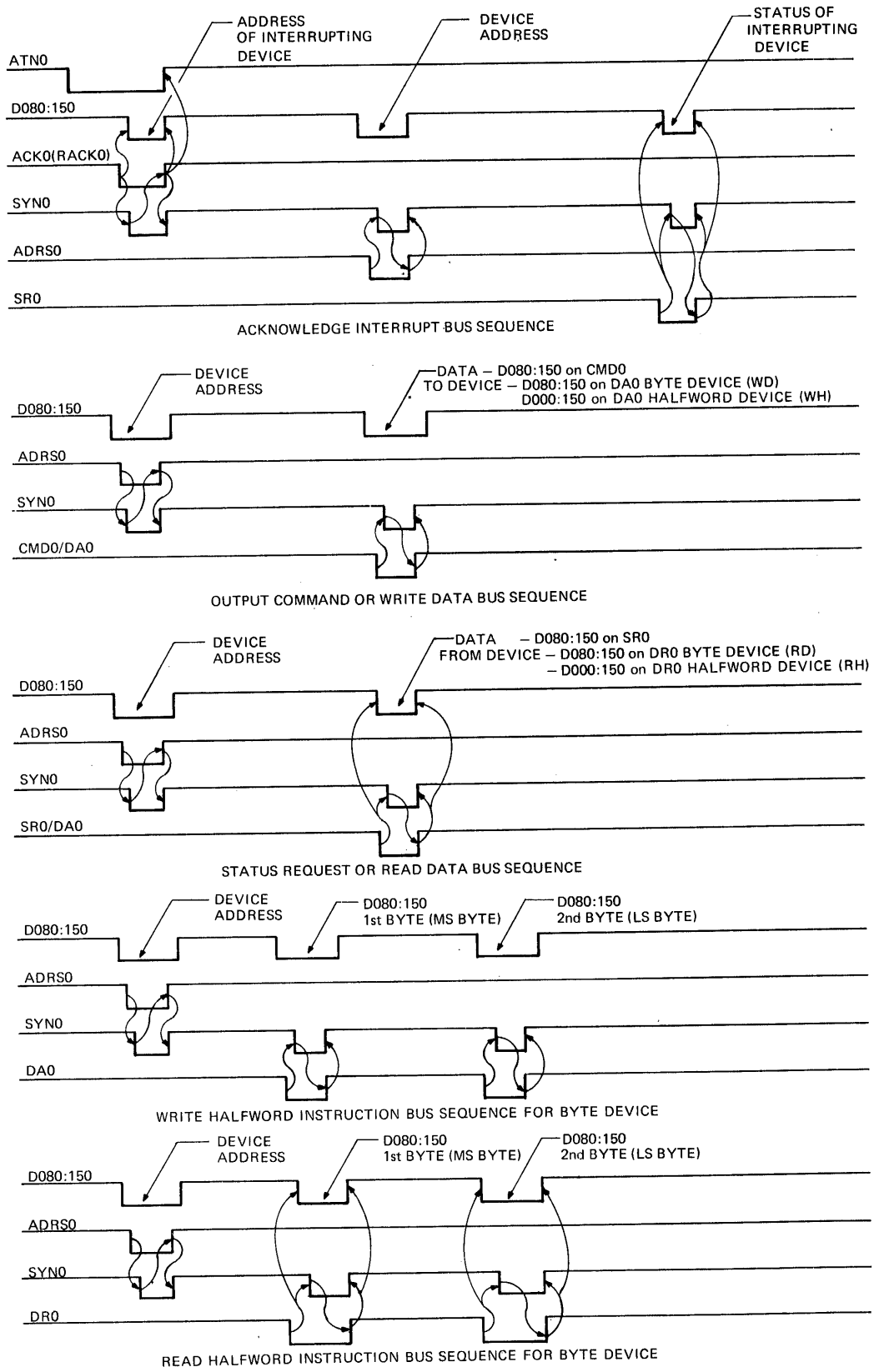


Figure 10-9. Bus Sequence For Byte Or Halfword Device

I/O Bus Sequence and Timing

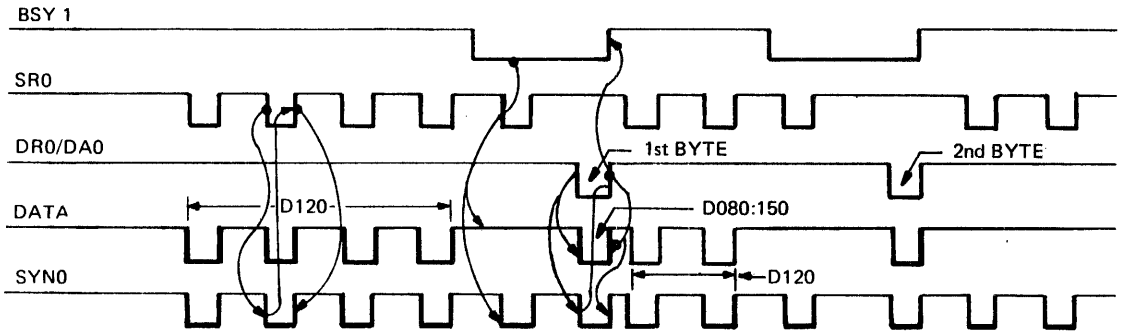
The standard I/O Bus sequence and timing for the I/O Instruction Set is shown on Figures 10-9 through 10-13 for reference.

NOTE

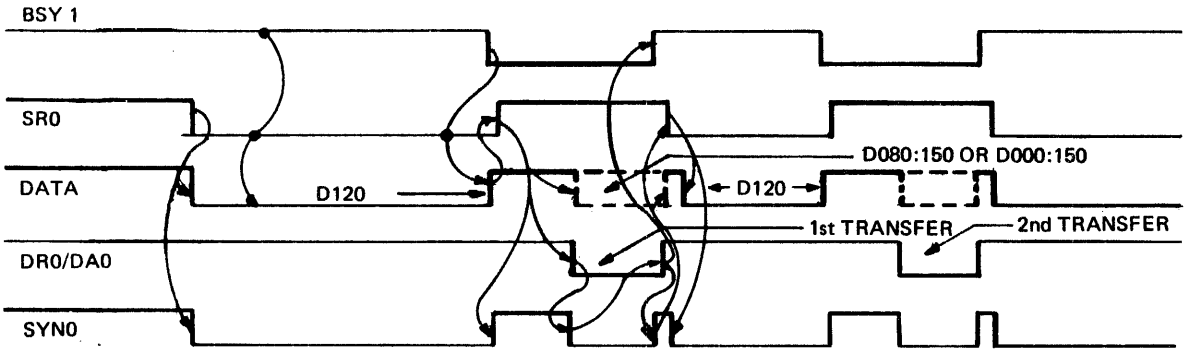
The Busy Status bit must always be set only on the trailing edge of DA0 or DR0.

Data Transfer

The data transfer sequence between the processor and the device controller in the Block Transfer mode (Read Block/Write Block) and Selector Channel is shown on Figure 10-10 for reference.



READ/WRITE BLOCK DATA TRANSFER BUS SEQUENCE



SELECTOR CHANNEL
READ/WRITE DATA TRANSFER BUS SEQUENCE

Figure 10-10. Read/Write Data Transfer Bus Sequence

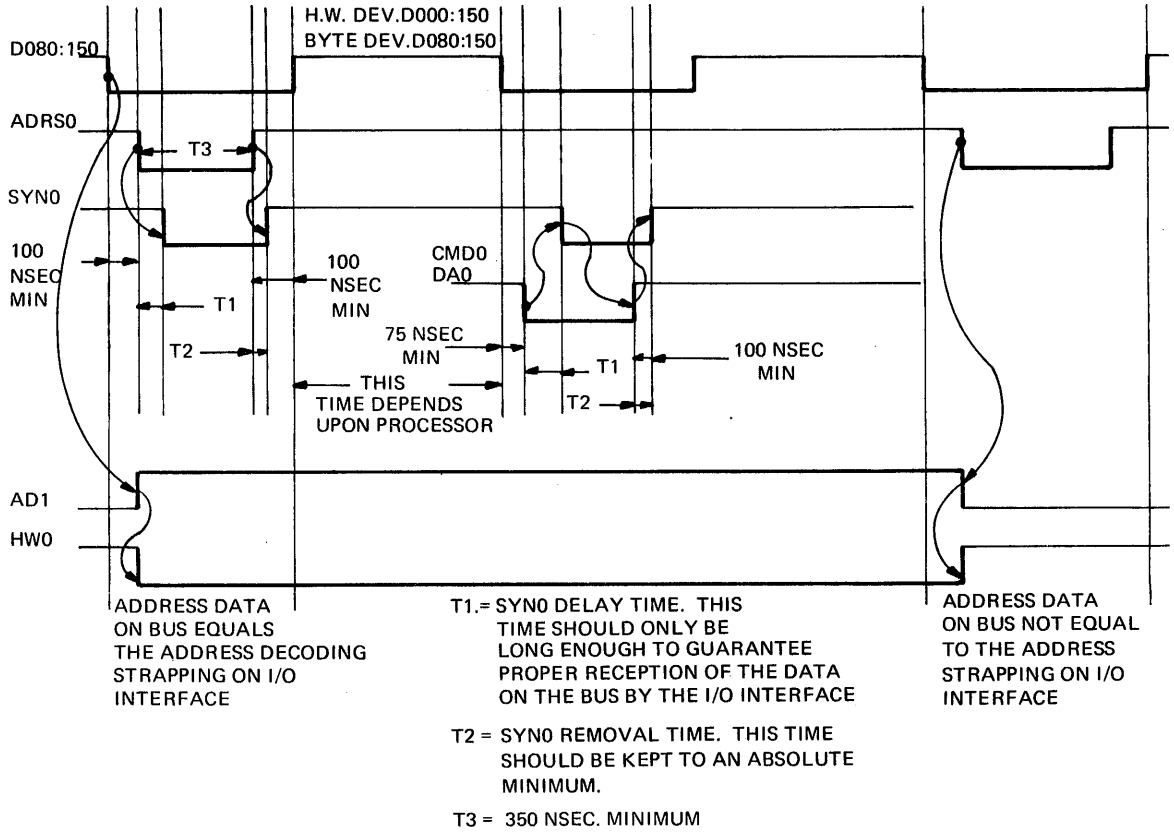


Figure 10-11. Address and Data Transfer Timing Between Processor and I/O Device Interface (Command and Data Available)

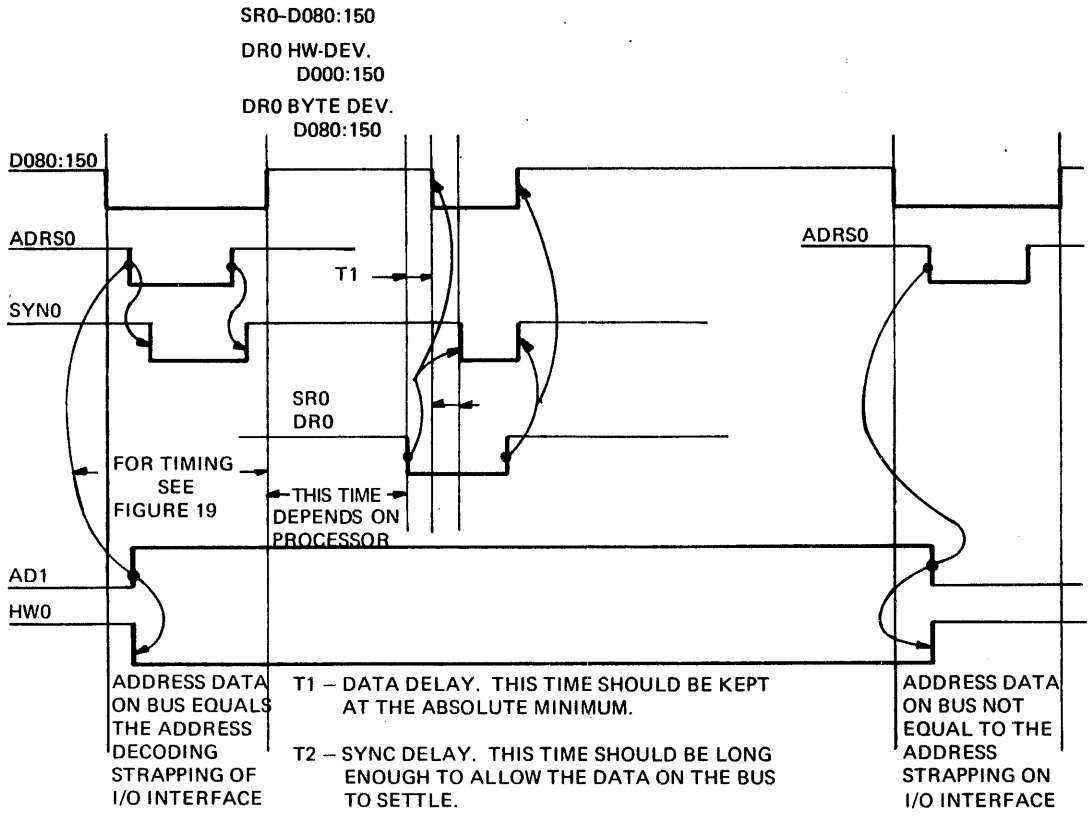


Figure 10-12. Address and Data Transfer Timing Between I/O Interface and Processor (Data Request and Sense Status)

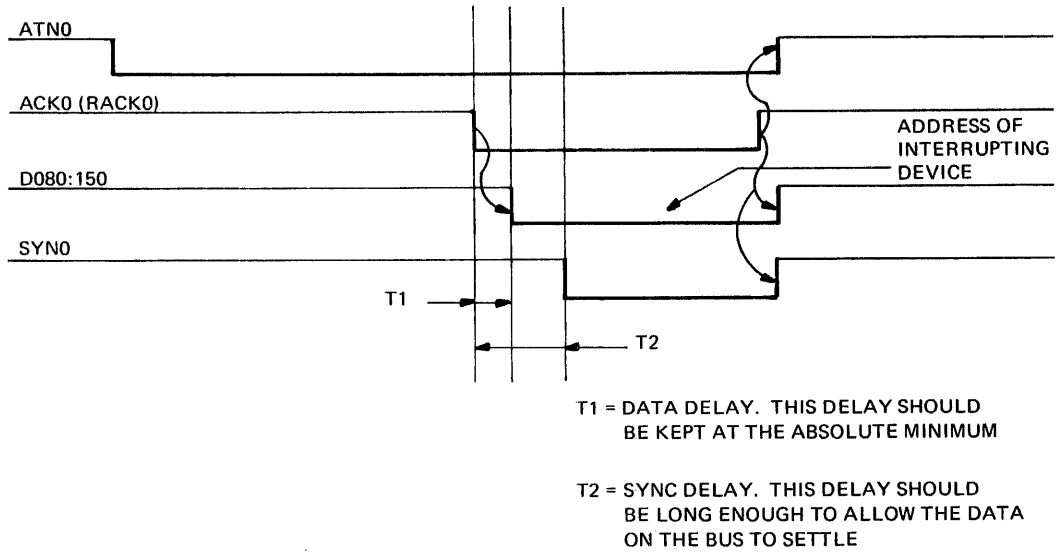


Figure 10-13. Interrupt Timing

MULTIPLEXOR I/O INTERFACE PHYSICAL PACKAGING, CABLING, AND CONNECTIONS

The I/O interface between any peripheral device and the processor Multiplexor I/O Bus uses 381 mm x 381 mm (15" x 15") printed circuit boards of 178 mm x 381 mm (7" x 15") printed circuit half-boards, as required. Hereafter, these boards are referred to as 381 mm (15") boards or 178 mm (7") half-boards respectively. The size of the printed circuit board used in an I/O interface depends upon the amount of logic required in its design.

7 Inch Half-boards

Two 178 mm (7") half-boards can be inserted into a 381 mm (15") chassis via the 16-398 Half-Board Adapter Kit (see Figure 10-14). The 16-398 Half-Board Adapter Kit may contain two active 178 mm (7") half-boards or one active and one blank 178 mm (7") half-board, depending on the system requirement. No wiring takes place between the boards and the adapters. The adapters are designed such that the 84-pin connector on the board plugs directly into the back panel connector in the chassis.

The 178 mm (7") half-board contains one 84-pin back panel connector to pick up the I/O Bus signals. For peripheral device connection, one front connector is provided. The number of pins used (up to 50) is dictated by the application. All the connections are mechanically mounted per drawing SK653. The 84-pin back panel connector and the front connector may be Connectors 1 (CONN 1) and 3 (CONN 3), or Connectors 0 (CONN 0) and 2 (CONN 2) respectively, depending upon which side of the 178 mm (7") half-boards is to be connected to the I/O Bus. Refer to Figure 10-14. Generally, if the I/O interface contains less than 60 ICs the I/O interface fits on one 178 mm (7") half-board.

15 Inch Boards

Each 381 mm (15") board may contain two 84-pin back panel connectors, labeled Connector 1 (CONN 1) and Connector 0 (CONN 0), to pick up the I/O Bus, and two 50-pin front connectors, labeled Connector 3 (CONN 3) and Connector 2 (CONN 2), for peripheral device connection. All the connectors are mechanically mounted per drawing SK653. Refer to Figure 10-15.

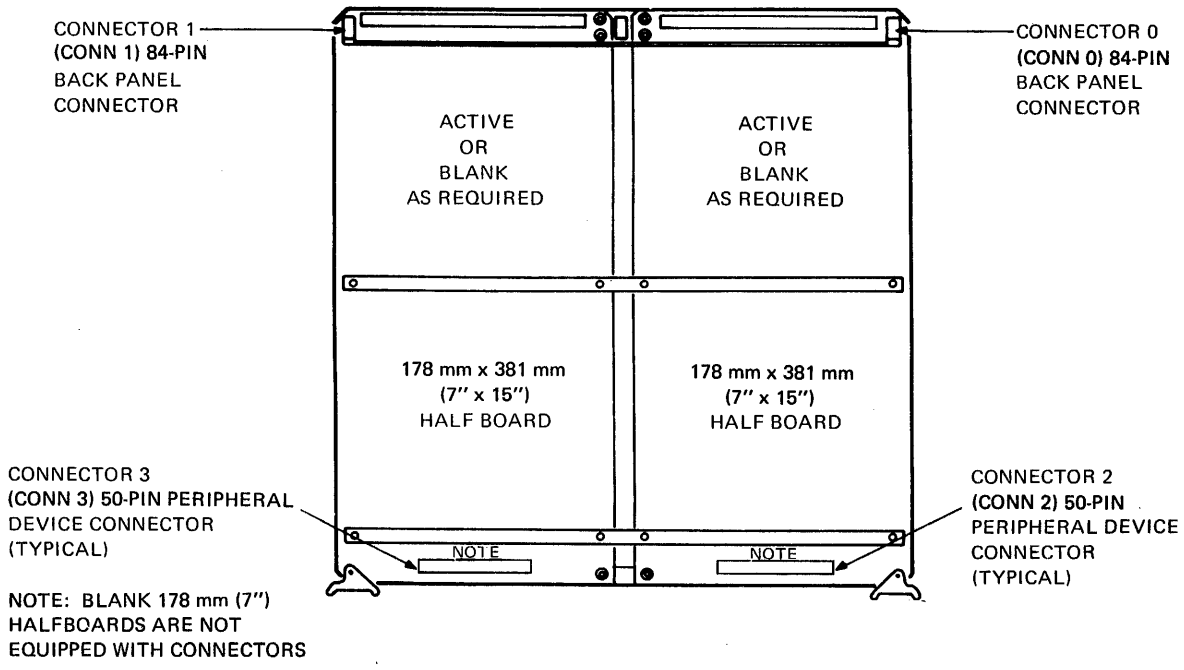


Figure 10-14. 16-398 Half Board Adapter Kit

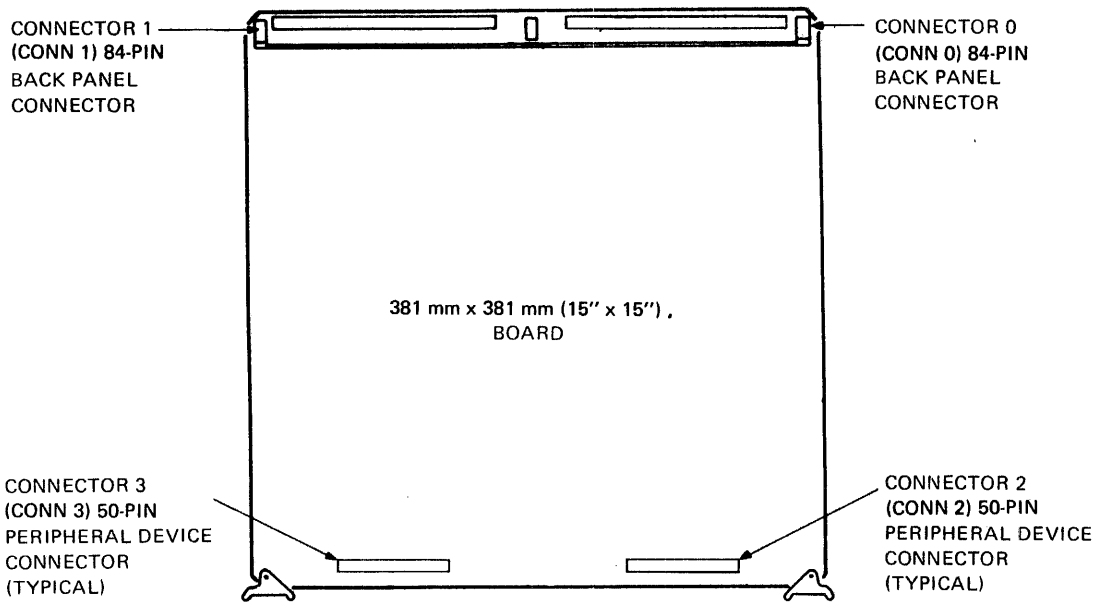


Figure 10-15. 381 mm x 381 mm (15" x 15") Printed Circuit Board

If the proposed I/O interface design requires more than a 178 mm (7") half-board, one or more 381 mm (15") boards may be used. No back panel stitch pattern exists to interconnect multi-board designs. Thus, cables must be used for board-to-board interconnects.

When designing a 381 mm (15") board, use either Connector 0 (CONN 0) or Connector 1 (CONN 1) to pick up the I/O signals from the back panel.

NOTE

Do not pick up some I/O signals from one connector and some from another for convenience of layout. Always use one connector.

Functions Common to the 178 mm (7") and 381 mm (15") I/O Interface Boards

1. No pins on the back panel may be used for I/O purposes other than those listed in Figure 10-16. All other pins are reserved for Memory connections or other purposes.
2. The I/O signals are duplicated in the Connector 0 and Connector 1 84-pin connectors in the same pin positions. That is, 119-0 (Conn 0) has the same logic function as 119-1 (CONN 1).
3. Only the standard board cable connector is used for low and medium speed signals. The maximum number of connections is 50 per connector.
4. The 3M-type Ribbon cable may be used for high speed signals. The maximum number of connections is 50 per connector.
5. All cables used in the interior of the cabinet shall be covered with a U.L. approved material.

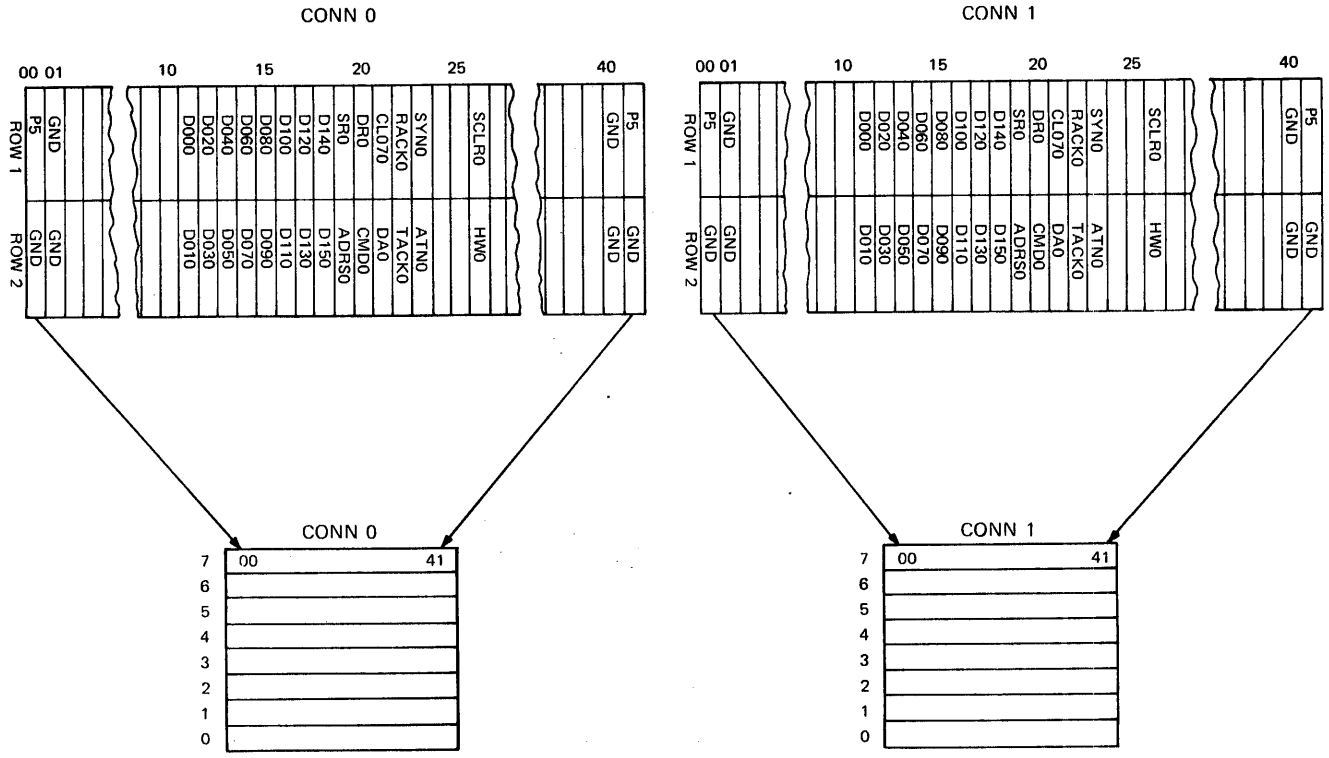


Figure 10-16. I/O Back Panel Connections

CHAPTER 11
M71·102 HEXADECIMAL DISPLAY PANEL AND M71·101 BINARY DISPLAY PANEL PROGRAMMING SPECIFICATION

INTRODUCTION

The M71-102 Hexadecimal Display Panel and M71-101 Binary Display Panel provide a means to manually control the Processor, interrogate and display various Processor registers and machine status, set and display Processor memory locations, and may be programmed as an I/O device by the user. The Hexadecimal Display Panel and Binary Display Panel are identical in operation. For convenience of the operator the Hexadecimal Display is equipped with a Hexadecimal readout in addition to the standard Binary readout.

CONFIGURATION

The Hexadecimal Display Panel provides the system operator with visual indications of the state of the Processor, as well as manual control over the system.

The Hexadecimal Display Panel, shown in Figure 11-1, is a RETMA standard 133 mm x 483 mm (5 1/4 " x 19") panel which is plug removable from the Processor. It displays the current state of the Processor and provides all necessary manual control over the system. The following paragraphs describe the control and display elements of the Hexadecimal Display Panel.

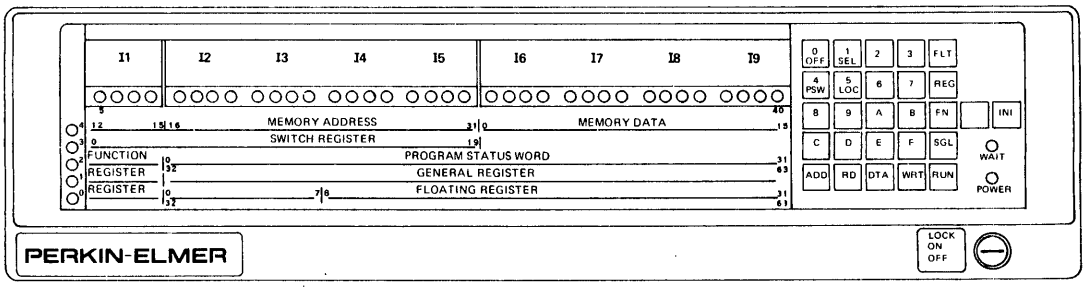


Figure 11-1. Hexadecimal Display Panel

Internal to the Hexadecimal Display Panel are five 8-bit byte Display Registers, D1 through D5, that hold data output from the Processor, and a 20-bit Switch Register that holds data input from The Hexadecimal Keyboard. Refer to Figure 11-2.

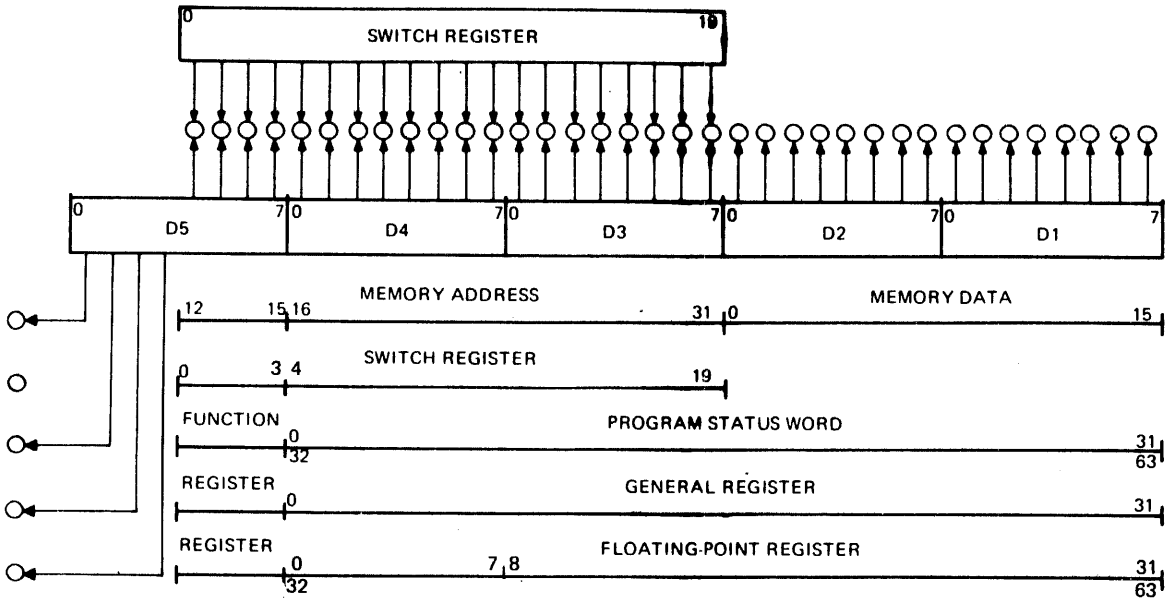


Figure 11-2. Display Registers and Indicators

Associated with each of Display Registers D1 through D4 are eight indicator lamps that provide a binary read-out and two optional hexadecimal read-out indicators. Associated with the least significant four bits of Display Register D5 are four indicator lamps for binary display and one optional hexadecimal read-out indicator.

The most significant four bits of Display Register D5 (bits 0:3) control four of the five indicator lamps along the left edge of the Hexadecimal Display Panel. The fifth indicator lamp is controlled by logic internal to the Hexadecimal Display Panel. To the right of each of these five lamps is a diagram that defines what is being displayed. In general, only one of the diagram lamps is on at a time. If none of the diagram lamps are on, a user program has written data to the Display Register D5.

As seen in Figure 9-2, the most significant 20-bits of the display show the contents of Display Registers D3 and D4 and the least significant four bits of Display Register D5 (bits 4:7); or the contents of the 20 bit Switch Register. When the Switch Register is being displayed, the lamp next to the Switch Register diagram is turned ON. Any other diagram lamp that may have been ON, remains ON. When the Switch Register is no longer displayed, its diagram lamp goes out and the most significant 20-bits of the display again show the contents of Display Registers D3 and D4 and the least significant four bits of Display Register D5 (bits 4:7).

The methods of displaying the Switch Register and the other diagrammed items are discussed later.

Key Operated Security Lock

This is a three-position, OFF-ON-LOCK, key-operated locking switch, which controls the primary power to the system. This switch can also disable the Hexadecimal Display Panel, thereby preventing any accidental manual input to the system. The power indicator lamp (PWR) associated with the key lock is located in the lower right corner of the Hexadecimal Display Panel. The PWR lamp is ON when the key lock is in the ON or LOCK position. The relationship between the key lock switch positions, primary power, the Control keys, and the Hexadecimal keys is:

OFF	The primary power is OFF.
ON	The primary power is ON and the Control keys and Hexadecimal keys are enabled.
LOCK	The primary power is ON and the Control keys and Hexadecimal keys are disabled.

Control Keys

The momentary contact Control keys are only active when the key-operated locking switch is in the ON position.

INITIALIZE (INT)	The Initialize (INT) key causes the system to be initialized. After the initialize operation, all device controllers on the system Multiplexor Bus are cleared and certain other functions in the Processor are reset.
DATA (DTA)	The Data (DTA) key clears the Switch Register and connects it to the most significant 20 display indicators. The Switch Register diagram lamp is turned ON. Hexadecimal data may now be entered into the Switch Register from the Hexadecimal Keyboard. As each Hexadecimal key is depressed, the data shifts into the Switch Register from the right. If more than five hexadecimal digits are entered, data shifted out of the Switch Register is lost. Depressing any non-hexadecimal key disconnects the Switch Register from the high order 20 display lamps and extinguishes the Switch Register diagram lamp.
ADDRESS (ADD)	The Address (ADD) key causes the Processor to halt and copy the contents of the Switch Register into the Location Counter field of the Program Status Word. The new value of the Location Counter is then output to Display Registers D1, D2, D3, and D4. The function diagram lamp is turned ON and a Hexadecimal 5 is output to the top four display lamps (bits 4:7 of D5).
MEMORY READ (RD)	The Memory Read (RD) key causes the Processor to halt and read the halfword contents of the memory location presently pointed to by the Location Counter. The memory operation is subject to translation as defined by PSW bits 8:11. The halfword data read is output to Display Registers D1 and D2. The Location Counter is incremented by two and output to Display Registers D3 and D4. The lamp next to the Memory Address/ Memory Data diagram is turned ON.

MEMORY WRITE (WRT)

The Memory Write (WRT) key causes the Processor to halt and read in the least significant 16 bits of the 20-bit Switch Register. The halfword of data is written into the memory location presently pointed to by the Location Counter. The memory operation is subject to translation or defined by PSW bits 8:11. The data written is then output to Display Registers D1 and D2. The Location Counter is incremented by two and output to Display Registers D3 and D4. The lamp next to the Memory Address/Memory Data diagram is turned ON.

EXAMINE REGISTER (REG)

The Examine Register (REG) key sets up the Hexadecimal Display Panel to interpret the next Hexadecimal key depressed as a General Register number. When the hexadecimal register number key is depressed, the Processor halts and the content of the selected General Register is output to Display Registers D1, D2, D3 and D4. The General Register diagram lamp is turned ON and the number of the displayed register is output to the top four display lamps.

EXAMINE SINGLE PRECISION FLOATING-POINT REGISTER (FLT)

The Examine Floating-Point Register (FLT) key sets up the Hexadecimal Display Panel to interpret the next hexadecimal key depressed as the number of a Floating-Point Register. When the hexadecimal register number key is depressed, the Processor halts and the content of the selected Floating-Point Register is output to Display Registers D1, D2, D3, and D4. The Floating-Point Register diagram lamp is turned ON and the number of the displayed register is output to the top four display lamps. If an odd numbered register had been selected and the processor is not equipped with double precision option, the register number is forced to the next lower even value before being used. On Processors not equipped with floating-point, the result of this operation is undefined.

FUNCTION (FN)

The Function (FN) key sets up the Hexadecimal Display Panel to interpret the next hexadecimal key depressed as the number of one of sixteen functions. When the hexadecimal key is depressed, the Processor halts to interpret the selected function. If the function is undefined, the Processor remains halted with no change to the display indicators. The defined functions are detailed later.

SINGLE STEP (SGL)

The Single Step (SGL) key causes the Processor to execute one user level instruction at current location counter, increment the LOC and then halt. The register that was selected (PSW, LOC, General Register, etc.) is displayed.

RUN (RUN)

The Run (RUN) key causes the Processor to begin program execution at the address pointed to by the Location Counter (LOC).

OPERATING PROCEDURES

Power Up

To power up the system, turn the key-operated security lock clockwise from the OFF position to the ON position. This action provides electrical power to the system and leaves all device controllers on the Multiplexor Bus in an initialized state.

Power Down

To shut down power to the system:

1. Halt the Processor.

2. Turn the key-operated security lock to the OFF position.

This removes primary power from the system and forces a Primary Power Fail (PPF) interrupt to the Processor. When power is re-applied, the Processor displays the contents of the Location Counter (LOC) and then assumes the Halt mode. If the Processor had been running when power was turned OFF, the Run mode is assumed when power is re-applied.

Program Status Word Display and Modification

To examine the Status field (most significant half) of the current PSW:

1. Depress the Function (FN) key.

2. Depress Hexadecimal key 4. The Processor halts and the status field (most significant half) of PSW is displayed.

To examine the Location Counter field (least significant half) of the current PSW:

1. Depress the Function (FN) key.

2. Depress Hexadecimal key 5. The Processor halts and the Location Counter field (least significant half) of PSW is displayed.

To modify the most significant 16 bits (bits 0-15) of the Program Status Register:

1. Depress the Data (DATA) key.

2. Enter the data (to be written into bits 0-15 of the PSW) from the Hexadecimal keyboard.

3. Depress the Function (FN) key.

4. Depress Hexadecimal key 1. The Processor halts and copies the 16 bits of the Switch register into bits 0-15 of the PSW. The modified PSW is then displayed.

Address a Memory Location

To select an address:

1. Depress the Data (DTA) key. The Switch Register is cleared and displayed.

2. Enter the desired address from the Hexadecimal Keyboard.

3. Depress the Address (ADD) key. The Processor halts and copies the contents of the Switch Register into the Location Counter field of the PSW. The new value of the Location Counter is then displayed.

Memory Read

To display the contents of memory locations:

1. As the memory read function is subject to translation as defined by PSW bits 8:11, the user should check these bits and modify them if necessary as in Program Status Word Display and Modification.

2. Select the memory read start address as in Address a Memory Location.

3. Depress the Read (RD) key. The address read from, plus two, and the data read from memory are displayed.

4. Repeat from Step 3 to read successive memory locations. The Location Counter is automatically incremented by two each time RD is depressed.

Memory Write

To write data from the Switch Register into memory:

1. As the memory read function is subject to translation as defined by PSW bits 8:11, the user should check these bits and modify them if necessary as in Program Status Word Display and Modification.

2. Select the memory write start address as in Address a Memory Location.

3. Depress the Data (DTA) key. The Switch Register is cleared and displayed.

4. Enter the data to be written from the Hexadecimal Keyboard.

5. Depress the Write (WRT) key. The address written into, plus two, and the data written are displayed.

6. Repeat from Step 3 to write different data into successive locations or from Step 5 to write the same data into successive locations. The Location Counter is automatically incremented by two each time WRT is depressed.

General Register Display

To examine the contents of a General Register:

1. Depress the Register (REG) key.

2. Depress the hexadecimal register number. The Processor halts and the contents of the selected General Register is displayed.

Single Precision Floating-Point Register Display

To examine the contents of a Floating-Point Register:

1. Depress the Floating-Point Register (FLT) key.

2. Depress the hexadecimal register number. If the Processor is not equipped with floating-point the result of this operation is undefined. If the Processor is equipped with floating-point, the selected register number is forced even and the Floating-Point Register is displayed. The Processor is left in the Halt mode.

Double Precision Floating-Point Register Display

After initialize or after a Function 2, all manual references to floating register are single precision. After a Function 3, all references to floating registers are double precision, if the Double Floating Point Unit (DFU) is equipped.

Using Even/Odd Concept

The even numbered register of an even/odd pair refers to the most significant 32 bits and the odd numbered register refers to the least significant 32 bits.

References to an odd numbered floating point register when in the single precision mode (FN 2) produce different results depending on whether or not the DFU is equipped. If DFU is absent, then the number is forced to the next lower even number and that single precision register is displayed. If DFU is present, then the LS 32 bits of the corresponding double precision register are displayed.

Program Execution

To begin execution of a program:

1. Select the program start address as in Address a Memory Location.
2. Select the register to be displayed.
3. Depress the Run (RUN) key.

To execute a program in the Single-Step mode:

1. Select the program start address as in Address a Memory Location.
2. Select the register to be displayed.
3. Depress the Single-Step (SGL) key. One instruction is executed, the last selected register (PSW, LOC, General Register, etc.) is displayed and the Processor halts.
4. Repeat Step 3 to execute successive instructions. Return to Step 2 to display different registers.

Program Termination

To manually halt the execution of a program, display any register or depress the Single-Step (SGL) key. In the latter case, the last selected register is displayed.

Console Interrupt

To generate an interrupt from the Hexadecimal Display Panel:

1. Depress the Function (FN) key.
2. Depress Hexadecimal key 0. If enabled by the current PSW, an interrupt from device number 1 is simulated. If not enabled, the Processor enters the Run mode. Hexadecimal Display Panel interrupts are not queued.

The Hexadecimal Display Panel interrupt feature allows an operator to inform the running program that some operator service or function is needed. No acknowledgement of the interrupt is required of the running program.

Switch Register

To examine the Switch Register at any time during execution of a program, depress any hexadecimal key. The Switch Register is displayed for as long as the key is depressed. No information enters the Switch Register. When the hexadecimal key is released, the top 20 display lamps return to their previous state.

The Switch Register can be modified without interrupting the Processor as follows:

1. Depress the Data (DTA) key. The Switch Register is cleared and displayed.
2. Enter the desired hexadecimal data.

Power Fail

When the Processor detects a power failure, the micro-program senses the Hexadecimal Display Panel status. The present status of the display is stored in main memory at a dedicated area by the micro-program. The current Program Status Word, Location Counter and the programmable registers are then saved in dedicated main memory locations and the micro-program deactivates the System Clear (SCLR) relay.

On power up, after the system clear relay has re-activated, the Program Status Word, Location Counter, and programmable registers are restored from their main memory save locations. The status of the display prior to the power failure is retrieved and interrogated by the micro-program.

If the Hexadecimal Display was in the Run mode and if the Machine Malfunction Interrupt Enable bit of the PSW is set, a Machine Malfunction Interrupt is taken. If Machine Malfunction Interrupts are not enabled, the Processor enters the Run mode beginning at the instruction pointed to by the Location Counter.

If the Hexadecimal Display Panel was not in the Run mode the value of the Location Counter is output to the display registers, the WAIT lamp on the console is turned ON and the Halt mode is entered.

Power failure and operation of the Initialize key are indistinguishable to the Micro-Program. Consequently, operation of the Initialize key should be considered carefully when the Machine Malfunction Interrupt is enabled.

Care should also be taken when using the Hexadecimal Display Panel as an input device (testing Switch Register bits) due to the volatility of the Switch Register in a power fail situation.

After a power up, the contents of the Switch Register are undefined. The display status byte is forced to X'40' on power up or initialize.

Wait State

The running program can place the Processor into the Wait state by setting the Wait bit of the current PSW. The WAIT indicator on the lower right of the panel is turned ON to inform the operator of the Wait state. The Processor can leave the Wait state and resume execution in two ways:

1. An Interrupt can occur causing the Processor to jump to an interrupt service routine. When the routine restores the original PSW, the Wait state is re-established.

2. The operator can depress the RUN key which causes the Wait bit in the PSW and the WAIT lamp to be reset and execution to resume at the address specified by LOC.

PROGRAMMING INSTRUCTIONS

Input/Output Programming

The Hexadecimal Display Panel is available to any running program as an I/O device with device address 01. The status and command bytes for the Hexadecimal Display Panel are summarized in Table 11-1. The status byte indicates the mode of the Hexadecimal Display Panel and is of little interest to a running program as it always indicates Run mode or Hexadecimal Display Panel Interrupt (Function 0). The command byte selects Normal or Incremental mode, which pertains to data Transfers. The selection logic which determines the Switch Register byte or register display byte to transfer is reset every time the Hexadecimal Display Panel is addressed when in the Normal mode. When an Output Command Incremental mode is issued to the Hexadecimal Display Panel, the byte selection logic is initially reset. Subsequent Read or Write instructions transfer bytes as shown in Figure 11-3.

Block I/O with the Hexadecimal Display Panel is only feasible when the least significant four status bits are reset.

NOTE

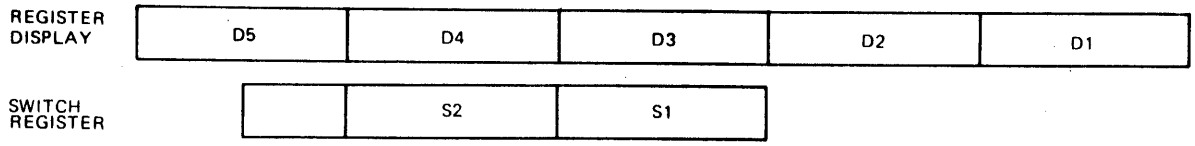
After an initialize sequence or after any manual Hexadecimal Display Panel operation that results in anything being displayed, the Display Device Controller is automatically placed in the Normal mode.

When programming the Hexadecimal Display Panel in the Incremental mode, the Output Command Incremental mode must be issued before each set of data transfers to guarantee that the byte selection logic is reset.

The most significant four bits of the Switch Register are only available to the micro-program. These four bits are transferred as bits 5, 6, 7, and 0 of the status when the Hexadecimal Display Panel status is Address (i.e., Display Status = X011XXXX).

DATA FORMAT

A byte or a halfword can be transferred to or from the Display using a WD, WH, WDR, WHR, or RD, RH, RDR, RHR instruction. Refer to Figure 11-3.



INSTRUCTIONS EXECUTED	DATA TRANSFERRED	
	NORMAL MODE	INCREMENTAL MODE
RD (R)	S1	S1
RD (R)	S1	S2
RD (R)	S1	S1
RD (R)	S1	S2
RH (R)	S1,S2	S1,S2
RB (R) *	S1,S2,S1.S2	S1,S2,S1,S2
WD (R)	D1	D1
WD (R)	D1	D2
WD (R)	D1	D3
WD (R)	D1	D4
WD (R)	D1	D5
WH (R)	D1,D2	D1,D2
WH (R)	D1,D2	D3,D4
WH (R)	D1,D2	D5,NOTE 1
WB (R) * *	D1,D2,D3,D4,D5	D1,D2,D3,D4,D5

* BLOCK LENGTH = 4 BYTES ** BLOCK LENGTH = 5 BYTES NOTE 1. SUBSEQUENT BYTES OUTPUT ARE LOST.

Figure 11-3. Hexadecimal Display Panel Data Transfers

PROGRAMMING SEQUENCES

The Hexadecimal Display has a device address of X'01'.

This device can be used to output up to five bytes of data to the Console Panel Indicators. The following program sequence outputs four bytes of data starting from the memory location BUF:

```
LIS     R1,1        (R1) = Display Address
LHI     R3,X'40'
OCR     R1,R3       Display in Incremental Mode
WD      R1,BUF
WD      R1,BUF+1
WD      R1,BUF+2
WD      R1,BUF+3
```

At this time the Console Panel Indicators are ON as shown below:

D5	D4	D3	D2	D1
	(BUF+3)	(BUF+2)	(BUF+1)	(BUF)

In order to light indicators D1 and D2, the Console can be in the normal mode and one halfword can be output. The following programming sequenoe can be used:

```
LIS     R1,1
LHI     R3,X'80'
OCR     R1,R3       Console in Normal Mode
WH      R1,BUF
```

The Console Panel Indicators will be ON as shown below:

D5	D4	D3	D2	D1
			(BUF+1)	(BUF)

Note that when a halfword of data is output to the Console Panel, the most significant byte loads in indicator D1 and the least significant byte loads in D2.

The Console Panel Switch Register can be read by using the read instructions as shown below:

```
LIS     R1,1        (R1) = Console Address
LHI     R3,X'80'    (R3) = 80 = Normal Mode
OCR     R1,R3
RHR     R1,R4       Read 1 Halfword
EXBR    R4,R4       Exchange Bytes
```

At this time Register 4 has the 16 data switches.

Programming Note:

If more than five bytes are output to the Display Panel, the data is lost after five bytes. The Console must then be initialized by giving an Output Command to it before outputting any data, if the data is to be displayed.

TABLE 11-1. DISPLAY STATUS AND COMMAND

STATUS

	0	1	2	3	4	5	6	7
Run	X	0	0	0	X	X	X	X
Memory write	X	0	0	1	X	X	X	X
Memory read	X	0	1	0	X	X	X	X
Address	X	0	1	1	X	X	X	X
Fixed Register	X	1	0	0	X	X	X	X
Floating Register	X	1	0	1	X	X	X	X
Function	X	1	0	0	X	X	X	X

General Register									Floating Register
0	0	1	0	X	1	0	0	0	0
1	1	1	0	X	1	0	0	0	
2	0	1	0	X	1	0	0	1	2
3	1	1	0	X	1	0	0	1	
4	0	1	0	X	1	0	1	0	4
5	1	1	0	X	1	0	1	0	
6	0	1	0	X	1	0	1	1	6
7	1	1	0	X	1	0	1	1	
8	0	1	0	X	1	1	0	0	8
9	1	1	0	X	1	1	0	0	
A	0	1	0	X	1	1	0	1	A
B	1	1	0	X	1	1	0	1	
C	0	1	0	X	1	1	1	0	C
D	1	1	0	X	1	1	1	0	
E	0	1	0	X	1	1	1	1	Floating Register E
General Register F	1	1	0	X	1	1	1	1	

Function									
0	0	1	0	0	0	0	0	0	Console Interrupt
1	1	1	0	0	0	0	0	0	PSW Select
2	0	1	0	0	0	0	0	1	Set Single precision display mode
3	1	1	0	0	0	0	0	1	Set Double precision display mode
4	0	1	0	0	0	0	1	0	Display PSW
5	1	1	0	0	0	0	1	0	Display LOC
6	0	1	0	0	0	0	1	1	
7	1	1	0	0	0	0	1	1	
8	0	1	0	0	0	1	0	0	
9	1	1	0	0	0	1	0	0	
A	0	1	0	0	0	1	0	1	
B	1	1	0	0	0	1	0	1	
C	0	1	0	0	0	1	1	0	
D	1	1	0	0	0	1	1	0	
E	0	1	0	0	0	1	1	1	
Function F	1	1	0	0	0	1	1	1	

COMMAND

Normal	1	0	0	0	0	0	0	0
Incremental	0	1	0	0	0	0	0	0

APPENDIX 1
MODEL 8/16E OP-CODE MAP

LSD \ MSD	0	1	2	3	4	5	6	8	9	C	D	E
0			BTBS		STH[1]		STE[1]$_2$	STD[1]$_2$	SRLS	BXH[1]	STM[1]	
1	BALR		BTFS		BAL[1]		AHM[1]	STME[1]$_2$	SLLS	BXLE[1]	LM[1]	SVC[1]
2	BTCR		BFBS		BTC[1]			LME[1]$_2$	STBR	LPSW[1] *	STB	SINT *
3	BFCR	SETMR *	BFFS	LPSR *	BFC[1]	SETM *		LPS *	LBR	THI	LB	
4	NHR		LIS		NH[1]		ATL[1]		EXBR	NHI	CLB	
5	CLHR		LCS		CLH[1]		ABL[1]		EPSR *	CLHI	AL *	
6	OHR		AIS		OH[1]		RTL[1]		WBR *	OHI	WB[1] *	
7	XHR		SIS		XH[1]		RBL[1]		RBR *	XHI	RB[1] *	
8	LHR		LER$_2$	LDR$_2$	LH[1]		LE[1]$_2$	BRK *	WHR *	LHI	WH[1] *	
9	CHR		CER$_2$	CDR$_2$	CH[1]		CE[1]$_2$	CD[1]$_2$	RHR *	CHI	RH[1] *	
A	AHR		AER$_2$	ADR$_2$	AH[1]		AE[1]$_2$	AD[1]$_2$	WDR *	AHI	WD *	RRL
B	SHR		SER$_2$	SDR$_2$	SH[1]		SE[1]$_2$	SD[1]$_2$	RDR *	SHI	RD *	RLL
C	MHR		MER$_2$	MDR$_2$	MH[1]		ME[1]$_2$	MD[1]$_2$	MHUR	SRHL	MHU[1]	SRL
D	DHR		DER$_2$	DDR$_2$	DH[1]		DE[1]$_2$	DD[1]$_2$	SSR *	SLHL	SS *	SLL
E	ACHR		FXR$_2$	FXDR$_2$	ACH[1]			STMD[1]$_2$	OCR *	SRHA	OC *	SRA
F	SCHR		FLR$_2$	FLDR$_2$	SCH[1]			LMD[1]$_2$	ACKR (AIR) *	SLHA	ACK (AI) *	SLA

*PRIVILEGED INSTRUCTIONS

NOTES

1. SECOND OPERAND MUST BE ALIGNED ON A HALFWORD BOUNDARY.
2. THESE INSTRUCTIONS ARE OPTIONAL.

APPENDIX 2
INSTRUCTION SUMMARY – ALPHABETICAL WITH ATTRIBUTES

Attributes

A: Arithmetic Fault Interrupt can occur
C: Condition Code in the PSW is set to reflect the result
D: Second operand should be on doubleword boundary for consistent results
F: Second operand should be on fullword boundary for consistent result (also see H)
H: Second operand must be on halfword boundary for consistent result
I: Illegal Instruction Interrupt can be initiated
IA: Immediate Interrupt can be initiated
P: Protect Mode violation can occur

INSTRUCTION	OP-CODE	MNEMONIC	ATTRIBUTES	PAGE NO.
Acknowledge Interrupt	DF	ACK(AI)	C,P	9-4
Acknowledge Interrupt Register	9F	ACKR(AIR)	C,P	9-4
Add Double Precision Floating Point	7A	AD	C,D,A,I	6-30
Add Floating Point	6A	AE	C,F,A,I	6-14
Add Floating Point Register	2A	AER	C,A,I	6-14
Add Halfword	4A	AH	C,H	5-3
Add Halfword Immediate	CA	AHI	C	5-3
Add Halfword to Memory	61	AHM	C,H	5-4
Add Halfword Register	0A	AHR	C	5-3
Add Immediate Short	26	AIS	C	5-3
Add Register Double Precision Floating Point	3A	ADR	C,A,I	3-25
Add to Bottom of List	65	ABL	C,F	3-25
Add to Top of List	64	ATL	C,F	3-25
Add with Carry Halfword	4E	ACH		5-6
AND Halfword	44	NH	C,H	3-15
AND Halfword Immediate	C4	NHI	C	3-15
AND Halfword Register	04	NHR	C	3-15
Autoload	D5	AL	C,P	9-15
Branch and Link	41	BAL	H	4-4
Branch and Link Register	01	BALR		4-4
Branch on False Condition	43	BFC	H	4-3
Branch on False Condition Backward Short	22	BFBS		4-3
Branch on False Condition Forward Short	23	BFFS		4-3
Branch on False Condition Register	03	BFCR		4-3
Branch on Index High	C0	BXH	H	4-6
Branch on Index Low or Equal	C1	BXLE	H	4-5
Branch on True Condition	42	BTC	H	4-2
Branch on True Condition Backward Short	20	BTBS		4-2
Branch on True Condition Forward Short	21	BTFS		4-2
Branch on True Condition Register	02	BTCR		4-2
Compare Double Precision Floating Point	79	CD	C,D,I	6-32
Compare Floating Point	69	CE	C,F,I	6-18
Compare Floating Point Register	29	CER	C,I	6-18
Compare Halfword	49	CH	C,H	5-8
Compare Halfword Immediate	C9	CHI	C	5-8
Compare Halfword Register	09	CHR	C	5-8
Compare Logical Byte	D4	CLB	C	3-14
Compare Logical Halfword	45	CLH	C,H	3-13
Compare Logical Halfword Immediate	C5	CLHI	C	3-13
Compare Logical Halfword Register	05	CLHR	C	3-13
Compare Register Double Precision Floating Point	39	CDR	C,I	6-32
Divide Double Precision Floating Point	7D	DD	C,D,A,I	6-34
Divide Floating Point	6D	DE	C,F,A,I	6-21
Divide Floating Point Register	2D	DER	C,A,I	6-21
Divide Halfword	4D	DH	H,A	5-11
Divide Halfword Register	0D	DHR	A	5-11
Divide Register Double Precision Floating Point	3D	DDR	C,A,I	6-34

INSTRUCTION	OP-CODE	MNEMONIC	ATTRIBUTES	PAGE NO.
Exchange Byte Register	94	EXBR		3-9
Exchange Program Status Register	95	EPSR	C,P,IA	8-9
Exclusive OR Halfword	47	XH	C,H	3-17
Exclusive OR Halfword Immediate	C7	XHI	C	3-17
Exclusive OR Halfword Register	07	XHR	C	3-17
Fix Register	2E	FXR	C,I	6-23
Fix Register Double Precision Floating Point	3E	FXDR	C,I	6-35
Float Register	2F	FLR	C,I	6-24
Float Register Double Precision Floating Point	3F	FLDR	C,I	6-36
Load Byte	D3	LB		3-8
Load Byte Register	93	LBR		3-8
Load Complement Short	25	LCS	C	3-5
Load Double Precision Floating Point	78	LD	C,D,A,I	6-25
Load Floating Point	68	LE	C,F,A,I	6-10
Load Floating Point Multiple	72	LME	F,I	6-11
Load Floating Point Register	28	LER	C,A,I	6-10
Load Halfword	48	LH	C	3-6
Load Halfword Immediate	C8	LHI	C	3-8
Load Halfword Register	08	LHR	C	3-5
Load Immediate Short	24	LIS	C	3-5
Load Multiple	D1	LM	F	3-7
Load Multiple Double Precision Floating Point	7F	LMD	D,I	6-27
Load Program Status	73	LPS		7-5
Load Program Status Register	33	LPSR		7-5
Load Program Status Word	C2	LPSW	C,P,IA	8-8
Load Register Double Precision Floating Point	38	LDR	C,A,I	6-25
Multiply Double Precision Floating Point	7C	MD	C,D,A,I	6-33
Multiply Floating Point	6C	ME	C,F,A,I	6-19
Multiply Floating Point Register	2C	MER	C,A,I	6-19
Multiply Halfword	4C	MH	H	5-9
Multiply Halfword Register	0C	MHR		5-9
Multiply Halfword Unsigned	DC	MHUR		5-10
Multiply Halfword Unsigned Register	9C	MHU		5-10
Multiply Register Double Precision Floating Point	3C	MDR	C,A,I	6-33
OR Halfword	46	OH	C,H	3-16
OR Halfword Immediate	C6	OHI	C	3-16
OR Halfword Register	06	OHR	C	3-16
Output Command	DE	OC	C,P,IA	9-6
Output Command Register	9E	OCR	C,P,IA	9-6
Read Block	D7	RB	C,F,P	9-9
Read Block Register	97	RBR	C,P	9-10
Read Data	DB	RD	C,P	9-7
Read Data Register	9B	RDR	C,P	9-7
Read Halfword	D9	RH	C,H,P	9-7
Read Halfword Register	99	RHR	C,P	9-8
Remove from Bottom of List	67	RBL	C,F	3-26
Remove from Top of List	66	RTL	C,F	3-26
Rotate Left Logical	EB	RLL	C	3-23
Rotate Right Logical	EA	RRL	C	3-24
Sense Status	DD	SS	C,P	9-5
Sense Status Register	9D	SSR	C,P	9-5
Set Map	53	SETM		7-6
Set Map Register	13	SETMR		7-6

INSTRUCTION	OP-CODE	MNEMONIC	ATTRIBUTES	PAGE NO.
Shift Left Arithmetic	EF	SLA	C	5-13
Shift Left Halfword Arithmetic	CF	SLHA	C	5-14
Shift Left Halfword Logical	CD	SLHL	C	3-21
Shift Left Logical Short	91	SLLS	C	3-21
Shift Left Logical	ED	SLL	C	3-10
Shift Right Arithmetic	EE	SRA	C	5-15
Shift Right Halfword Arithmetic	CE	SRHA	C	5-15
Shift Right Halfword Logical	CC	SRHL	C	3-22
Shift Right Logical Short	90	SRLS	C	3-22
Shift Right Logical	EC	SRL	C	3-20
Simulate Interrupt	E2	SINT	C,P,IA	8-10
Store Byte	D2	STB	RP	3-12
Store Byte Register	92	STBR		3-12
Store Double Precision Floating Point	70	STD	D,I	6-28
Store Floating Point	60	STE	F,I	6-12
Store Floating Point Multiple	71	STME	F,I	6-13
Store Halfword	40	STH	H	3-10
Store Multiple	D0	STM	F	3-11
Store Multiple Double Precision Floating Point	7E	STMD	D,I	6-29
Subtract Double Precision Floating Point	7B	SD	C,D,A,I	6-31
Subtract Floating Point	6B	SE	C,F,A,I	6-16
Subtract Floating Point Register	2B	SER	C,A,I	6-16
Subtract Halfword	4B	SH	C,H	5-5
Subtract Halfword Immediate	CB	SHI	C	5-5
Subtract Halfword Register	0B	SHR	C	5-5
Subtract Immediate Short	27	SIS	C	5-5
Subtract Register Double Precision Floating Point	3B	SDR	C,A,I	6-31
Subtract with Carry Halfword	4F	SCH		5-7
Supervisor Call	E1	SVC	C,F	8-11
Test Halfword Immediate	C3	THI	C	3-18
Write Block	D6	WB	C,F,P	9-13
Write Block Register	96	WBR	C,P	9-14
Write Data	DA	WD	C,P	9-11
Write Data Register	9A	WDR	C,P	9-11
Write Halfword	D8	WH	C,H,P	9-12
Write Halfword Register	98	WHR	C,P	9-12

APPENDIX 3
INSTRUCTION SUMMARY NUMERICAL

OP CODE	MNEMONIC	INSTRUCTION	PAGE NO.
01*	BALR	Branch and Link Register	4-4
02*	BTCR	Branch on True Condition Register	4-2
03*	BFCR	Branch on False Condition Register	4-3
04	NHR	AND Halfword Register	3-15
05	CLHR	Compare Logical Halfword Register	3-13
06	OHR	OR Halfword Register	3-16
07	XHR	Exclusive OR Halfword Register	3-17
08	LHR	Load Halfword Register	3-5
09	CHR	Compare Halfword Register	5-8
0A	AHR	Add Halfword Register	5-3
0B	SHR	Subtract Halfword Register	5-5
0C*	MHR	Multiply Halfword Register	5-9
0D*	DHR	Divide Halfword Register	5-11
0E	ACHR	Add with Carry Halfword Register	5-6
0F	SCHR	Subtract with Carry Halfword Register	5-7
13	SETMR	Set Map Register	7-6
20*	BTBS	Branch on True Condition Backward Short	4-2
21*	BTFS	Branch on True Condition Forward Short	4-2
22*	BFBS	Branch on False Condition Backward Short	4-3
23*	BFFS	Branch on False Condition Forward Short	4-3
24	LIS	Load Immediate Short	3-5
25	LCS	Load Complement Short	3-5
26	AIS	Add Immediate Short	5-3
27	SIS	Subtract Immediate Short	5-5
28	LER	Floating Point Load Register	6-10
29	CER	Floating Point Compare Register	6-18
2A	AER	Floating Point Add Register	6-14
2B	SER	Floating Point Subtract Register	6-16
2C	MER	Floating Point Multiply Register	6-19
2D	DER	Floating Point Divide Register	6-21
2E	FXR	Fix Register	6-23
2F	FLR	Float Register	6-24
33	LPSR	Load Program Status Register	7-5
38	LDR	Load Register Double Precision Floating Point	6-25
39	CDR	Compare Register Double Precision Floating Point	6-32
3A	ADR	Add Register Double Precision Floating Point	3-25
3B	SDR	Subtract Register Double Precision Floating Point	6-31
3C	MDR	Multiply Register Double Precision Floating Point	6-33
3D	DDR	Divide Register Double Precision Floating Point	6-34
3E	FXDR	Fix Register Double Precision Floating Point	6-35
3F	FLDR	Float Register Double Precision Floating Point	6-36
40*	STH	Store Halfword	3-10
41*	BAL	Branch and Link	4-4
42*	BTC	Branch on True Condition	4-5
43*	BFC	Branch on False Condition	4-3
44	NH	AND Halfword	4-15
45	CLH	Compare Logical Halfword	3-13

OP CODE	MNEMONIC	INSTRUCTION	PAGE NO.
46	OH	OR Halfword	3-16
47	XH	Exclusive OR Halfword	3-17
48	LH	Load Halfword	3-6
49	CH	Compare Halfword	5-8
4A	AH	Add Halfword	5-3
4B	SH	Subtract Halfword	5-5
4C*	MH	Multiply Halfword	5-9
4D*	DH	Divide Halfword	5-11
4E	ACH	Add with Carry Halfword	5-6
4F	SCH	Subtract with Carry Halfword	5-7
53	SETM	Set Map	7-6
60*	STE	Store Floating Point	6-12
61	AHM	Add Halfword to Memory	5-4
64	ATL	Add to Top of List	3-25
65	ABL	Add to Bottom of List	3-25
66	RTL	Remove from Top of List	3-26
67	RBL	Remove from Bottom of List	3-26
68	LE	Load Floating Point	6-10
69	CE	Compare Floating Point	6-18
6A	AE	Add Floating Point	6-14
6B	SE	Subtract Floating Point	6-16
6C	ME	Multiply Floating Point	6-19
6D	DE	Divide Floating Point	6-21
70	STD	Store Double Precision Floating Point	6-28
71	STME	Store Floating Point Multiple	6-13
72	LME	Load Floating Point Multiple	6-11
73	LPS	Load Program Status	7-5
78	LD	Load Double Precision Floating Point	6-25
79	CD	Compare Double Precision Floating Point	6-32
7A	AD	Add Double Precision Floating Point	6-30
7B	SD	Subtract Double Precision Floating Point	6-31
7C	MD	Multiply Double Precision Floating Point	6-33
7D	DD	Divide Double Precision Floating Point	6-34
7E	STMD	Store Multiple Double Precision Floating Point	6-29
7F	LMD	Load Multiple Double Precision Floating Point	6-27
90	SRLS	Shift Right Logical Short	3-22
91	SLLS	Shift Left Logical Short	3-21
92*	STBR	Store Byte Register	3-12
93*	LBR	Load Byte Register	3-8
94*	EXBR	Exchange Byte Register	3-9
95	EPSR	Exchange Program Status Register	8-9
96	WBR	Write Block Register	9-14
97	RBR	Read Block Register	9-10
98	WHR	Write Halfword Register	9-12
99	RHR	Read Halfword Register	9-8
9A	WDR	Write Data Register	9-11
9B	RDR	Read Data Register	9-7
9C*	MHUR	Multiply Halfword Unsigned Register	5-10
9D	SSR	Sense Status Register	9-5
9E	OCR	Output Command Register	9-6
9F	ACKR (AIR)	Acknowledge Interrupt Register	9-4

OP CODE	MNEMONIC	INSTRUCTION	PAGE NO.
C0*	BXH	Branch on Index High	4-6
C1*	BXLE	Branch on Index Low or Equal	4-1
C2	LPSW	Load Program Status Word	8-8
C3	THI	Test Halfword Immediate	3-18
C4	NHI	AND Halfword Immediate	3-15
C5	CLHI	Compare Logical Halfword Immediate	3-13
C6	OHI	OR Halfword Immediate	3-16
C7	XHI	Exclusive OR Halfword Immediate	3-17
C8	LHI	Load Halfword Immediate	3-8
C9	CHI	Compare Halfword Immediate	5-8
CA	AHI	Add Halfword Immediate	5-3
CB	SHI	Subtract Halfword Immediate	5-5
CC	SRHL	Shift Right Halfword Logical	3-22
CD	SLHL	Shift Left Halfword Logical	3-21
CE	SRHA	Shift Right Halfword Arithmetic	5-15
CF	SLHA	Shift Left Halfword Arithmetic	5-14
D0*	STM	Store Multiple	3-11
D1*	LM	Load Multiple	3-7
D2*	STB	Store Byte	3-12
D3*	LB	Load Byte	3-8
D4	CLB	Compare Logical Byte	3-14
D5	AL	Autoload	9-15
D6	WB	Write Block	9-13
D7	RB	Read Block	9-9
D8	WH	Write Halfword	9-12
D9	RH	Read Halfword	9-8
DA	WD	Write Data	9-11
DB	RD	Read Data	9-7
DC*	MHU	Multiply Halfword Unsigned	5-10
DD	SS	Sense Status	9-5
DE	OC	Output Command	9-6
DF	ACK (AI)	Acknowledge Interrupt	9-4
E1	SVC	Supervisor Call	8-11
E2	SINT	Simulate Interrupt	8-10
EA	RRL	Rotate Right Logical	3-24
EB	RLL	Rotate Left Logical	3-23
EC	SRL	Shift Right Logical	3-20
ED	SLL	Shift Left Logical	3-10
EE	SRA	Shift Right Arithmetic	5-15
EF	SLA	Shift Left Arithmetic	5-13

* Condition Code NOT CHANGED.

APPENDIX 4
EXTENDED BRANCH MNEMONICS

INSTRUCTION	OP CODE (M1)	MNEMONIC	OPERAND
Branch on Carry	428	BC	A(X2)
Branch on Carry Register	028	BCR	R2
Branch on No Carry	438	BNC	A(X2)
Branch on No Carry Register	038	BNCR	R2
Branch on Equal	433	BE	A(X2)
Branch on Equal Register	033	BER	R2
Branch on Not Equal	423	BNE	A(X2)
Branch on Not Equal Register	023	BNER	R2
Branch on Low	428	BL	A(X2)
Branch on Low Register	028	BLR	R2
Branch on Not Low	438	BNL	A(X2)
Branch on Not Low Register	038	BNLR	R2
Branch on Minus	421	BM	A(X2)
Branch on Minus Register	021	BMR	R2
Branch on Not Minus	431	BNM	A(X2)
Branch on Not Minus Register	031	BNMR	R2
Branch on Plus	422	BP	A(X2)
Branch on Plus Register	022	BPR	R2
Branch on Not Plus	432	BNP	A(X2)
Branch on Not Plus Register	032	BNPR	R2
Branch on Overflow	424	BO	A(X2)
Branch on Overflow Register	024	BOR	R2
Branch on No Overflow	434	BNO	A(X2)
Branch on No Overflow Register	034	BNOR	R2
Branch Unconditional	430	B	A(X2)
Branch Unconditional Register	030	BR	R2
Branch on Zero	433	BZ	A(X2)
Branch on Zero Register	033	BZR	R2
Branch on Not Zero	423	BNZ	A(X2)
Branch on Not Zero Register	023	BNZR	R2
No Operation	420	NOP	
No Operation Register	020	NOPR	
Branch on Carry Short	208	BCS	A (Backward Reference)
	218	BCS	A (Forward Reference)
Branch on No Carry Short	228	BNCS	A (Backward Reference)
	238	BNCS	A (Forward Reference)
Branch on Equal Short	223	BES	A (Backward Reference)
	233	BES	A (Forward Reference)
Branch on Not Equal Short	203	BNES	A (Backward Reference)
	213	BNES	A (Forward Reference)
Branch on Low Short	208	BLS	A (Backward Reference)
	218	BLS	A (Forward Reference)
Branch on Not Low Short	228	BNLS	A (Backward Reference)
	238	BNLS	A (Forward Reference)

INSTRUCTION	OP CODE (M1)	MNEMONIC	OPERANDS
Branch on Minus Short	201	BMS	A (Backward Reference)
	211	BMS	A (Forward Reference)
Branch on Not Minus Short	221	BNMS	A (Backward Reference)
	231	BNMS	A (Forward Reference)
Branch on Plus Short	202	BPS	A (Backward Reference)
	212	BPS	A (Forward Reference)
Branch on Not Plus Short	222	BNPS	A (Backward Reference)
	232	BNPS	A (Forward Reference)
Branch on Overflow Short	204	BOS	A (Backward Reference)
	214	BOS	A (Forward Reference)
Branch on No Overflow Short	224	BNOS	A (Backward Reference)
	234	BNOS	A (Forward Reference)
Branch Unconditional Short	220	BS	A (Backward Reference)
	230	BS	A (Forward Reference)
Branch on Zero Short	223	BZS	A (Backward Reference)
	233	BZS	A (Forward Reference)
Branch on Not Zero Short	203	BNZS	A (Backward Reference)
	213	BNZS	A (Forward Reference)

APPENDIX 5
ARITHMETIC REFERENCES

TABLE OF POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5

APPENDIX 5 (Continued)

TABLE OF POWERS OF SIXTEEN

16^n							n
						1	0
						16	1
						256	2
					4	096	3
					65	536	4
				1	048	576	5
				16	777	216	6
				268	435	456	7
			4	294	967	296	8
			68	719	476	736	9
		1	099	511	627	776	10
		17	592	186	044	416	11
		281	474	976	710	656	12
	4	503	599	627	370	496	13
	72	057	594	037	927	936	14
1	152	921	504	606	846	976	15

Decimal Values

HEXADECIMAL ADDITION AND SUBTRACTION TABLE

Examples: 5+A = F; 18-D = B; A+B = 15

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	1
2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	2
3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	3
4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	4
5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	5
6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	6
7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	7
8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	8
9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	9
A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	A
B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	B
C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	C
D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	D
E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	E
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	F
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

HEXADECIMAL MULTIPLICATION AND DIVISION TABLE

Examples: 5x6 = 1E; 75÷D = 9; 58÷8 = B; 9xC = 6C

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	1
2	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E	2
3	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D	3
4	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C	4
5	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B	5
6	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A	6
7	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69	7
8	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78	8
9	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87	9
A	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96	A
B	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5	B
C	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4	C
D	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3	D
E	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2	E
F	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1	F
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

TABLE OF MATHEMATICAL CONSTANTS

CONSTANT	DECIMAL VALUE	HEXADECIMAL VALUE	FLOATING POINT VALUE
			DOUBLE PRECISION
			SINGLE PRECISION
π	3.14159 26535 89793 23846	3.243F 6A88 85A3 08D3	4132 43F6 A888 5A31
$\pi-1$	0.31830 98861 83790 67154	0.517C C1B7 2722 0A95	4051 7CC1 B727 220B
$\sqrt{\pi}$	1.77245 38509 05516 02730	1.C5BF 891B 4EF6 AA7A	411C 5BF8 91B4 EF6B
Ln π	1.14472 98858 49400 17414	1.250D 048E 7A1B D0BD	4112 B67A E858 4CAA
$\sqrt{3}$	1.73205 08075 68877 29353	1.BB67 AE85 84CA A73B	411B 67AE 8584 CAA7
e	2.71828 18284 59045 23536	2.B7E1 5162 8AED 2A6B	412B 7E15 1628 AED3
e^{-1}	0.36787 94411 71442 32160	0.5E2D 58D8 B3BC DF1B	405E 2D58 D8B3 BCDF
$\sqrt{e}$	1.64872 12707 00128 14683	1.A612 98E1 E069 BC97	411A 6129 8E1E 069C
$\log_{10}e$	0.43429 44819 03251 82765	0.6F2D EC54 9B94 38CB	406F 2DEC 5A9B 9439
$\log_{2}e$	1.44269 50408 88963 40736	1.7154 7652 B82F E177	4117 1547 652B 82FE
γ	0.57721 56649 01532 86061	0.93C4 67E3 7DB0 C7A5	4093 C467 E37D B0C8
Ln γ	-0.54953 93129 81644 82234	-0.8CAE 9BC1 1F5A 5FF4	C08C AE9B C11F 5A60
$\sqrt{2}$	1.41421 35623 73095 04880	1.6A09 E667 F3BC C909	4116 A09E 667F 3BCD
Ln2	0.69314 71805 59945 30942	0.B172 17F7 D1CF 79AC	40B1 7217 F7D1 CF7A
$\log_{10}2$	0.30102 99956 63981 19521	0.4D10 4D42 7DE7 FBCC	404D 104D 427D E7FC
$\sqrt{10}$	3.16227 76601 68379 33199	3.298B 075B 4B6A 5240	4132 98B0 75B4 B6A5
Ln10	2.30258 50929 94045 68402	2.4D76 3776 AAA2 B05C	4124 D763 776A AA2B

APPENDIX 5 (Continued)
INTEGER CONVERSION TABLE

Hexadecimal and Decimal Integer Conversion Table

HALFWORD								HALFWORD							
BYTE				BYTE				BYTE				BYTE			
BITS: 0123		4567		0123		4567		0123		4567		0123		4567	
Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	268,435,456	1	16,777,216	1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	536,870,912	2	33,554,432	2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	805,306,368	3	50,331,648	3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	1,073,741,824	4	67,108,864	4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	1,342,177,280	5	83,886,080	5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	1,610,612,736	6	100,663,296	6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	1,879,048,192	7	117,440,512	7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	2,147,483,648	8	134,217,728	8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	2,415,919,104	9	150,994,944	9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	2,684,354,560	A	167,772,160	A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	2,952,790,016	B	184,549,376	B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	3,221,225,472	C	201,326,592	C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	3,489,660,928	D	218,103,808	D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	3,758,096,384	E	234,881,024	E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	4,026,531,840	F	251,658,240	F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
8		7		6		5		4		3		2		1	

TO CONVERT HEXADECIMAL TO DECIMAL

1. Locate the column of decimal numbers corresponding to the left-most digit or letter of the hexadecimal; select from this column and record the number that corresponds to the position of the hexadecimal digit or letter.
2. Repeat step 1 for the next (second from the left) position.
3. Repeat step 1 for the units (third from the left) position.
4. Add the numbers selected from the table to form the decimal number.

EXAMPLE

Conversion of Hexadecimal Value	D34
1. D	3328
2. 3	48
3. 4	4
4. Decimal	3380

To convert integer numbers greater than the capacity of table, use the techniques below:

HEXADECIMAL TO DECIMAL

Successive cumulative multiplication from left to right, adding units position.

Example: $D34_{16} = 3380_{10}$

D = 13
x16
208
3 = + 3
211
x16
3376
4 = +4
3380

TO CONVERT DECIMAL TO HEXADECIMAL

1. (a) Select from the table the highest decimal number that is equal to or less than the number to be converted.
   (b) Record the hexadecimal of the column containing the selected number.
   (c) Subtract the selected decimal from the number to be converted.
2. Using the remainder from step 1(c) repeat all of step 1 to develop the second position of the hexadecimal (and a remainder).
3. Using the remainder from step 2 repeat all of step 1 to develop the units position of the hexadecimal.
4. Combine terms to form the hexadecimal number.

EXAMPLE

Conversion of Decimal Value	3380
1. D	-3328
	52
2. 3	-48
	4
3. 4	-4
4. Hexadecimal	D34

DECIMAL TO HEXADECIMAL

Divide and collect the remainder in reverse order.

Example: $3380_{10} = X_{16}$

16 | 3380 → remainder 4
16 | 211 → 3
16 | 13 → D

$3380_{10} = D34_{16}$

APPENDIX 5 (Continued)
FRACTION CONVERSION TABLE

Hexadecimal and Decimal Fraction Conversion Table

HALFWORD													
BYTE					BYTE								
BITS 0123		4567			0123				4567				
Hex	Decimal	Hex	Decimal		Hex	Decimal			Hex	Decimal Equivalent			
.0	.0000	.00	.0000	0000	.000	.0000	0000	0000	.0000	.0000	0000	0000	0000
.1	.0625	.01	.0039	0625	.001	.0002	4414	0625	.0001	.0000	1525	8789	0625
.2	.1250	.02	.0078	1250	.002	.0004	8828	1250	.0002	.0000	3051	7578	1250
.3	.1875	.03	.0117	1875	.003	.0007	3242	1875	.0003	.0000	4577	6367	1875
.4	.2500	.04	.0156	2500	.004	.0009	7656	2500	.0004	.0000	6103	5156	2500
.5	.3125	.05	.0195	3125	.005	.0012	2070	3125	.0005	.0000	7629	3945	3125
.6	.3750	.06	.0234	3750	.006	.0014	6484	3750	.0006	.0000	9155	2734	3750
.7	.4375	.07	.0273	4375	.007	.0017	0898	4375	.0007	.0001	0681	1523	4375
.8	.5000	.08	.0312	5000	.008	.0019	5312	5000	.0008	.0001	2207	0312	5000
.9	.5625	.09	.0351	5625	.009	.0021	9726	5625	.0009	.0001	3732	9101	5625
.A	.6250	.0A	.0390	6250	.00A	.0024	4140	6250	.000A	.0001	5258	7890	6250
.B	.6875	.0B	.0429	6875	.00B	.0026	8554	6875	.000B	.0001	6784	6679	6875
.C	.7500	.0C	.0468	7500	.00C	.0029	2968	7500	.000C	.0001	8310	5468	7500
.D	.8125	.0D	.0507	8125	.00D	.0031	7382	8125	.000D	.0001	9836	4257	8125
.E	.8750	.0E	.0546	8750	.00E	.0034	1796	8750	.000E	.0002	1362	3046	8750
.F	.9375	.0F	.0585	9375	.00F	.0036	6210	9375	.000F	.0002	2888	1835	9375
1		2			3				4				

TO CONVERT .ABC HEXADECIMAL TO DECIMAL

Find .A in position 1 .6250

Find .0B in position 2 .0429 6875

Find .00C in position 3 .0029 2968 7500

.ABC Hex is equal to .6708 9843 7500

TO CONVERT .13 DECIMAL TO HEXADECIMAL

1. Find .1250 next lowest to .1300
   subtract -.1250 = .2 Hex
2. Find .0039 0625 next lowest to .0050 0000
   -.0039 0625 = .01
3. Find .0009 7656 2500 .0010 9375 0000
   -.0009 7656 2500 = .004
4. Find .0001 0681 1523 4375 .0001 1718 7500 0000
   -.0001 0681 1523 4375 = .0007
   .0000 1037 5976 5625 = .2147 Hex
5. 13 Decimal is approximately equal to ——————————↑

To convert fractions beyond the capacity of table, use techniques below:

HEXADECIMAL FRACTION TO DECIMAL

Convert the hexadecimal fraction to its decimal equivalent using the same technique as for integer numbers. Divide the results by 16^n (n is the number of fraction positions).

Example: $.8A7 = .54077_{10}$

$8A7_{16} = 2215_{10}$

$16^3 = 4096$

```
          .540771
4096|2215.000000
```

DECIMAL FRACTION TO HEXADECIMAL

Collect integer parts of product in the order of calculation.

Example: $.5408_{10} = .8A7_{16}$

```
   .5408
    x16
8 ← [8].6528
    x16
A ← [10].4448
    x16
7 ← [7].1168
```

APPENDIX 6
INSTRUCTION TIMING
MODEL 8/16E WITH 750NS CORE

	RR/SF	RI	RX	COMMENTS	NOTES
ABL	-	-	2.75/7.00/7.50	OVF/NORM/WRAP	1
ACH	1.00	-	2.25		1
ACK	3.25	-	5.50		1
AD					
AE					
AH	0.75	1.25	2.00		1
AHM	-	-	3.00		1
AIS	1.00				
AL	-	-	4.75+2.75L+3.75N	L=LEADER, N=BYTES	1
ATL	-	-	2.75/6.75/7.00	OVF/NORM/WRAP	1
BAL	1.25	-	1.75		
BFBS	1.00/2.25	-	-	NO/YES	
BFC	1.00/1.25	-	1.50/1.75	NO/YES	
BFFS	1.00/2.00	-	-	NO/YES	
BTBS	1.00/2.25	-	-	NO/YES	
BTC	1.00/1.25	-	1.50/1.75	NO/YES	
BTFS	1.00/2.00	-	-	NO/YES	
BXH	-	-	2.75/3.00	NO/YES	1
BXLE	-	-	2.50/2.75	NO/YES	1
CD					
CE					
CH	1.25/1.75	1.75/2.25	2.50/3.00	SIGNS ALIKE/DIFFER	1
CLB	-	-	2.75		1
CLH	0.75	1.25	2.00		1
DD					
DE					
DH					
EPSR	3.00	-	-		
EXBR	1.00	-	-		
FLDR					
FLR					
FXDR					
FXR					
LB	1.25	-	2.75		1
LCS	0.75	-	-		
LD					
LE					
LH	0.75	1.25	2.00		1
LIS	1.00	-	-		
LM	-	-	2.25+10N	N=REGISTERS	1
LMD					
LME					
LPSS	3.00	-	4.25		1
LPSW	-	-	5.00		1
MD				MIN/AVE/MAX	
ME				MIN/AVE/MAX	
MH	5.50/6.00/6.50	-	6.75/7.25/7.75	MIN/AVE/MAX	1
MHU				MIN/AVE/MAX	
NH	0.75	1.25	2.00		1
OC	3.25	-	4.50		1
OH	0.75	1.25	2.00		1

	RR/SF	RI	RX	COMMENTS	NOTES
RB	2.75+.75R+3.75N	-	4.00+3.75N	N=BYTES	4, 1
RBL	-	-	2.25/6.50/6.75	UNF/NORM/WRAP	1
RD	2.50	-	4.25		1
RH	2.75/3.50	-	5.25/6.00	HALFWORD/BYTE	1
RLL	-	5.5/4.50+.25N	-	N=0/N=SHIFTS	2
RRL	-	5.5/4.50+.25N	-	N=0/N=SHIFTS	2
RTL	-	-	2.25/6.75/7.25	UNF/NORM/WRAP	1
SCH	1.00	-	2.25		1
SD					
SE					
SETM	4.50/6.00/6.25	-	5.75/7.25/7.50		5, 1
SH	0.75	1.25	2.00		
SINT	-	7.00	-	IMMEDIATE INTERRUPT	
SIS	1.00	-	-		
SLA	-	6.25/5.50+.25N	-	N=0/N=SHIFTS	2, 3
SLHA	-	5.75/4.00+.25N	-	N=0/N=SHIFTS	2, 3
SLHL	-	4.0/3.25+.25N	-	N=0/N=SHIFTS	2,
SLL	-	5.5/4.25+.25N	-	N=0/N=SHIFTS	2
SLLS	1.25/1.75+.25N	-	-	N=0/N=SHIFTS	
SRA	-	6.25/4.25+1.50N	-	N=0/N=SHIFTS	2, 3
SRHA	-	5.75/5.00+0.25N	-		2
SRHL	-	4.0/3.50+.25N	-	N=0/N=SHIFTS	2
SRL	-	5.5/4.50+.25N	-	N=0/N=SHIFTS	2
SRLS	1.25/1.75+2.00+25N	-	-	N=0/N=SHIFTS	
SS	2.50	-	4.75		1
STB	2.00	-	3.50		1
STD					
STE					
STH	-	-	2.50		
STM	-	-	1.75+1.0N	N=REGISTERS	1
STMD					
STME					
SVC	-	-	6.25		
THI	-	1.25	-		
WB	2.75+75R+3.75N	-	4.00+3.75N	N=BYTES	4, 1
WD	2.50	-	3.50		1
WH	2.75/3.50	-	3.75/4.50	HALFWORD BYTE	1
XH	0.75	1.25	2.00		1

NOTE 1 add 0.25 if indexed RX
NOTE 2 add 0.25 if indexed
NOTE 3 add 0.50 if argument negative
NOTE 4 R=Value of R2 field
NOTE 5 (BANK=0:6 or F) or (8:E and R1 minus)/BANK=7/BANK=8:E and R1 positive on LPSW, EPSR, LPSS, LPSSR, Add 2.25 if Queue Service interrupt enabled. Add 4.75 if interrupt taken

APPENDIX 7

I/O REFERENCES

TELETYPE/ASCII HEX CONVERSION TABLE

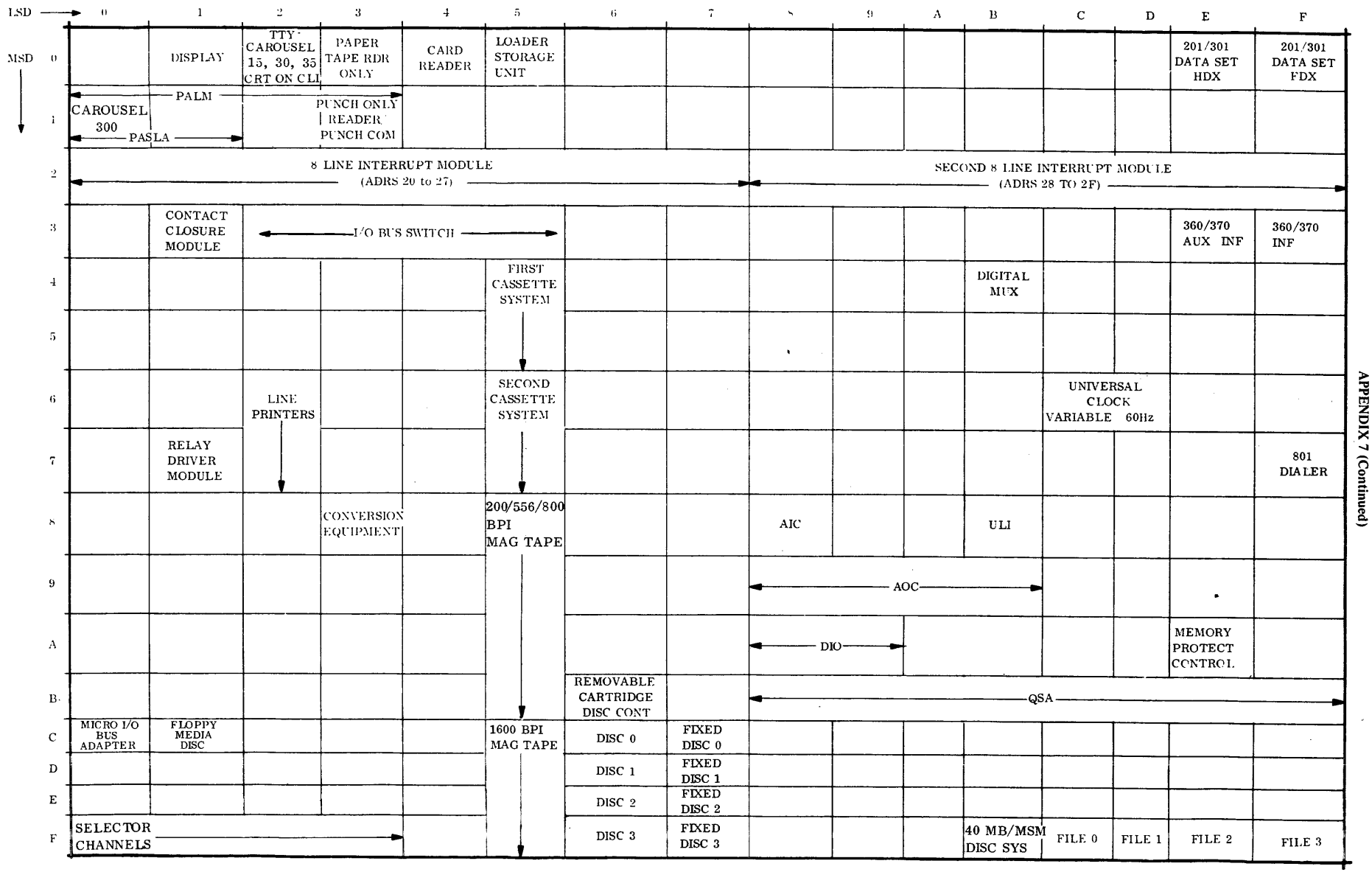
HEX (MSD) →					0	1	2	3	4	5	6	7
(LSD) ↓	Teletype Tape Channels →			8	DEPENDS UPON PARITY*							
				7	0	0	0	0	1	1	1	1
				6	0	0	1	1	0	0	1	1
				5	0	1	0	1	0	1	0	1
	4	3	2	1								
0	0	0	0	0	NULL	DC_0	SPACE	0	@	P		
1	0	0	0	1	SOM	X-ON	!	1	A	Q		
2	0	0	1	0	EOA	TAPE ON	"	2	B	R		
3	0	0	1	1	EOM	X-OFF	#	3	C	S		
4	0	1	0	0	EOT	TAPE OFF	$	4	D	T		
5	0	1	0	1	WRU	ERR	%	5	E	U		
6	0	1	1	0	RU	SYNC	&	6	F	V		
7	0	1	1	1	BELL	LEM	'	7	G	W		
8	1	0	0	0	FE_0	S_0	(	8	H	X		
9	1	0	0	1	HT/SK	S_1	)	9	I	Y		
A	1	0	1	0	LF	S_2	*	:	J	Z		
B	1	0	1	1	VT	S_3	+	;	K	[		
C	1	1	0	0	FF	S_4	,	<	L	\		ACK
D	1	1	0	1	CR	S_5	-	=	M	]		ALT. MODE
E	1	1	1	0	SO	S_6	.	>	N	↑		ESC
F	1	1	1	1	SI	S_7	/	?	O	←		DEL

*Parity bit adjusted for even parity (even number of 1's) on input from Teletype keyboard. Parity bit is ignored on output to Teletype printer.

ASCII CARD CODE CONVERSION TABLE
(029 EBCDIC)

GRAPHIC	7-BIT ASCII CODE	CARD CODE	GRAPHIC	7-BIT ASCII CODE	CARD CODE
SPACE	20	BLANK	@	40	8-4
!	21	12-8-7	A	41	12-1
"	22	8-7	B	42	12-2
#	23	8-3	C	43	12-3
$	24	11-8-3	D	44	12-4
%	25	0-8-4	E	45	12-5
&	26	12	F	46	12-6
'	27	8-5	G	47	12-7
(	28	12-8-5	H	48	12-8
)	29	11-8-5	I	49	12-9
*	2A	11-8-4	J	4A	11-1
+	2B	12-8-6	K	4B	11-2
,	2C	0-8-3	L	4C	11-3
-	2D	11	M	4D	11-4
.	2E	12-8-3	N	4E	11-5
/	2F	0-1	O	4F	11-6
0	30	0	P	50	11-7
1	31	1	Q	51	11-8
2	32	2	R	52	11-9
3	33	3	S	53	0-2
4	34	4	T	54	0-3
5	35	5	U	55	0-4
6	36	6	V	56	0-5
7	37	7	W	57	0-6
8	38	8	X	58	0-7
9	39	9	Y	59	0-8
:	3A	8-2	Z	5A	0-9
;	3B	11-8-6	[	5B	12-8-2
<	3C	12-8-4	\	5C	11-8-1
=	3D	8-6	]	5D	11-8-2
>	3E	0-8-6	↑	5E	11-8-7
?	3F	0-8-7	←	5F	0-8-5

STANDARD PREFERRED ADDRESS TABLE



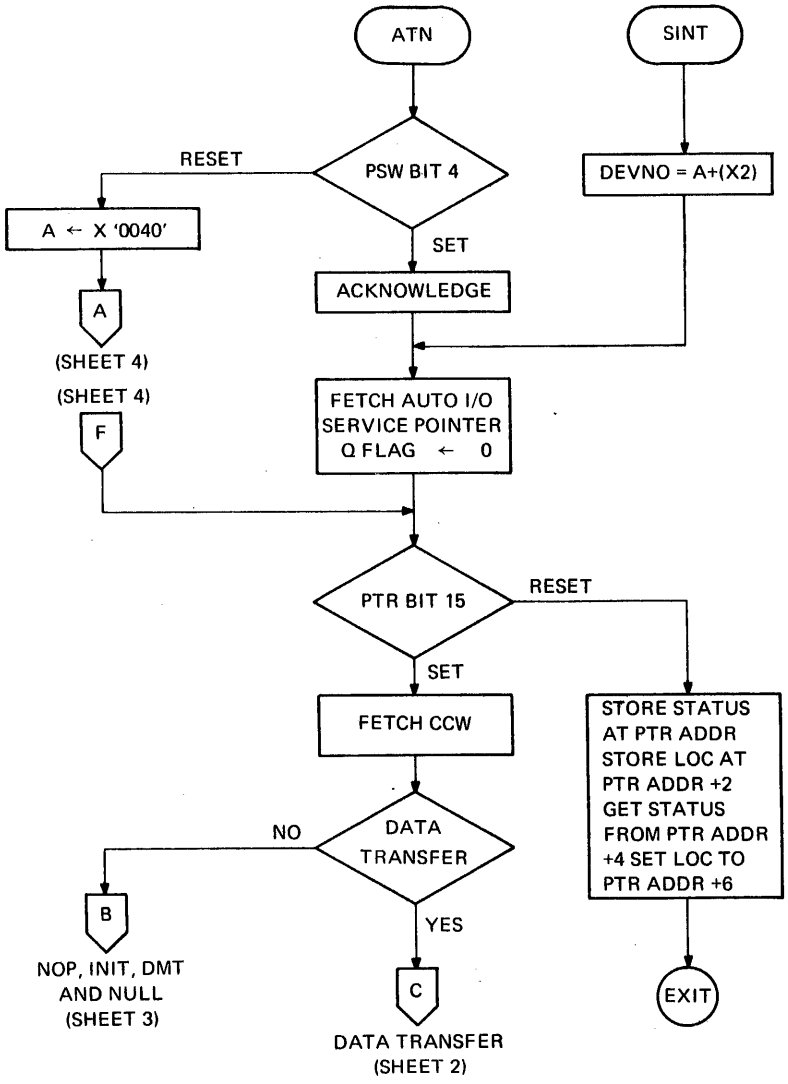
AIC = ANALOG INPUT CONTROLLER
AOC = ANALOG OUTPUT CONTROLLER
DIO = DIGITAL I/O CONTROLLER
QSA = QUAD SYNCHRONOUS ADAPTER
ULI = UNIVERSAL LOGIC INTERFACE

CAROUSEL ASCII/HEX CONVERSION TABLE

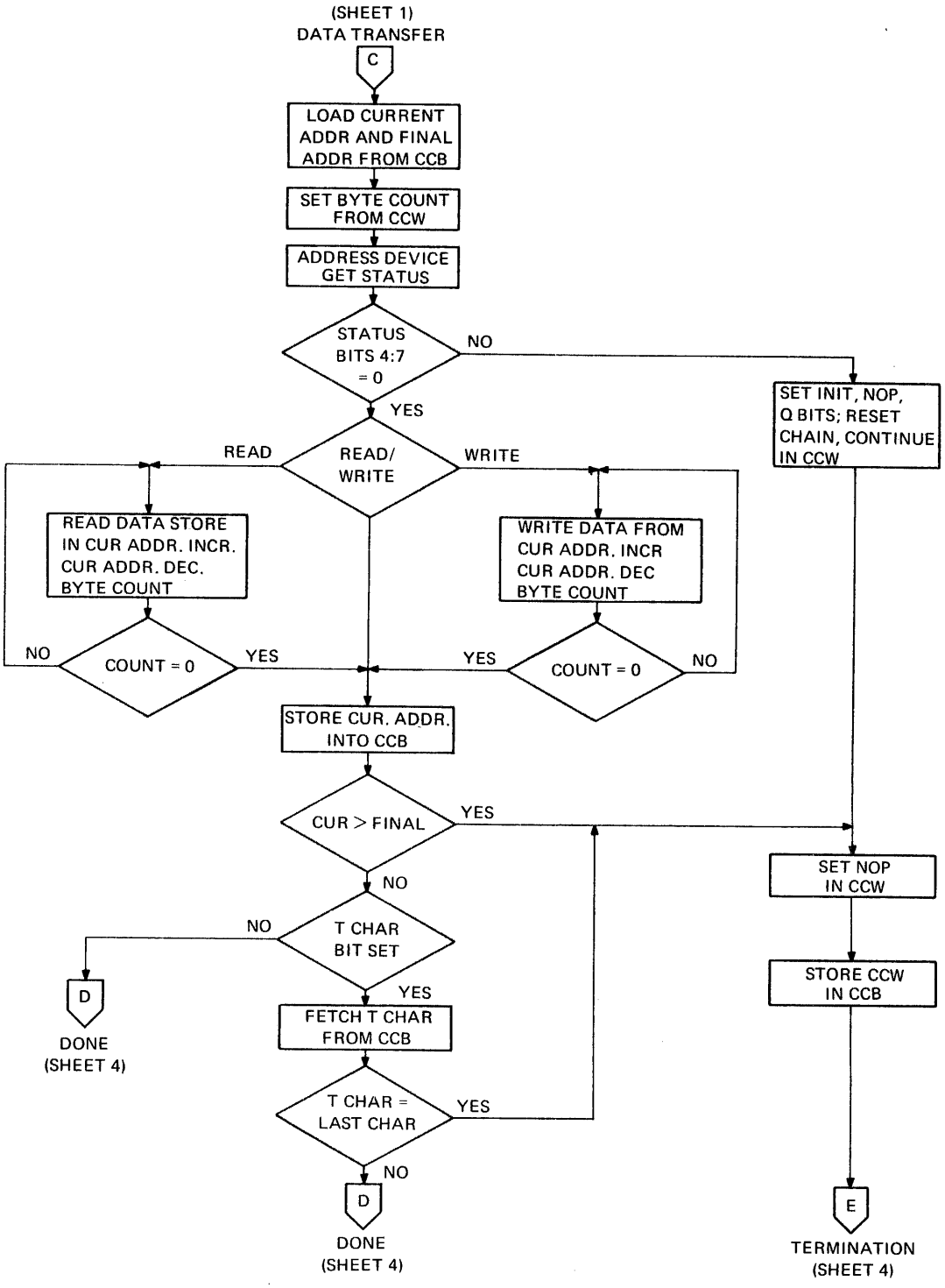
BITS				b_6 b_5 b_4	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_3	b_2	b_1	b_0	MSD / LSD	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SPACE	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	A	LF	SUB	*	:	J	Z	j	z
1	0	1	1	B	VT	ESC	+	;	K	[	k	{
1	1	0	0	C	FF	FS	,	<	L	\	l	¦
1	1	0	1	D	CR	GS	–	=	M	]	m	}
1	1	1	0	E	SO	RS	.	>	N	^	n	~
1	1	1	1	F	SI	US	/	?	O	—	o	DEL

NUL	Null	DLE	Data link escape
SOH	Start of heading	DC1-3	Device control
STX	Start of text	DC4	Device stop
ETX	End of text	NAK	Negative acknowledge
EOT	End of transmission	SYN	Synchronous idle
ENQ	Enquiry	ETB	End of transmission block
ACK	Acknowledge	CAN	Cancel
BEL	Audible signal	EM	End of medium
BS	Backspace	SUB	Start of special sequence
HT	Horizontal tabulation	ESC	Escape
LF	Line feed	FS	File separator
VT	Vertical tabulation	GS	Group separator
FF	Form feed	RS	Record separator
CR	Carriage return	US	Unit separator
SO	Shift out	SP	Space
SI	Shift in	DEL	Delete/Idle

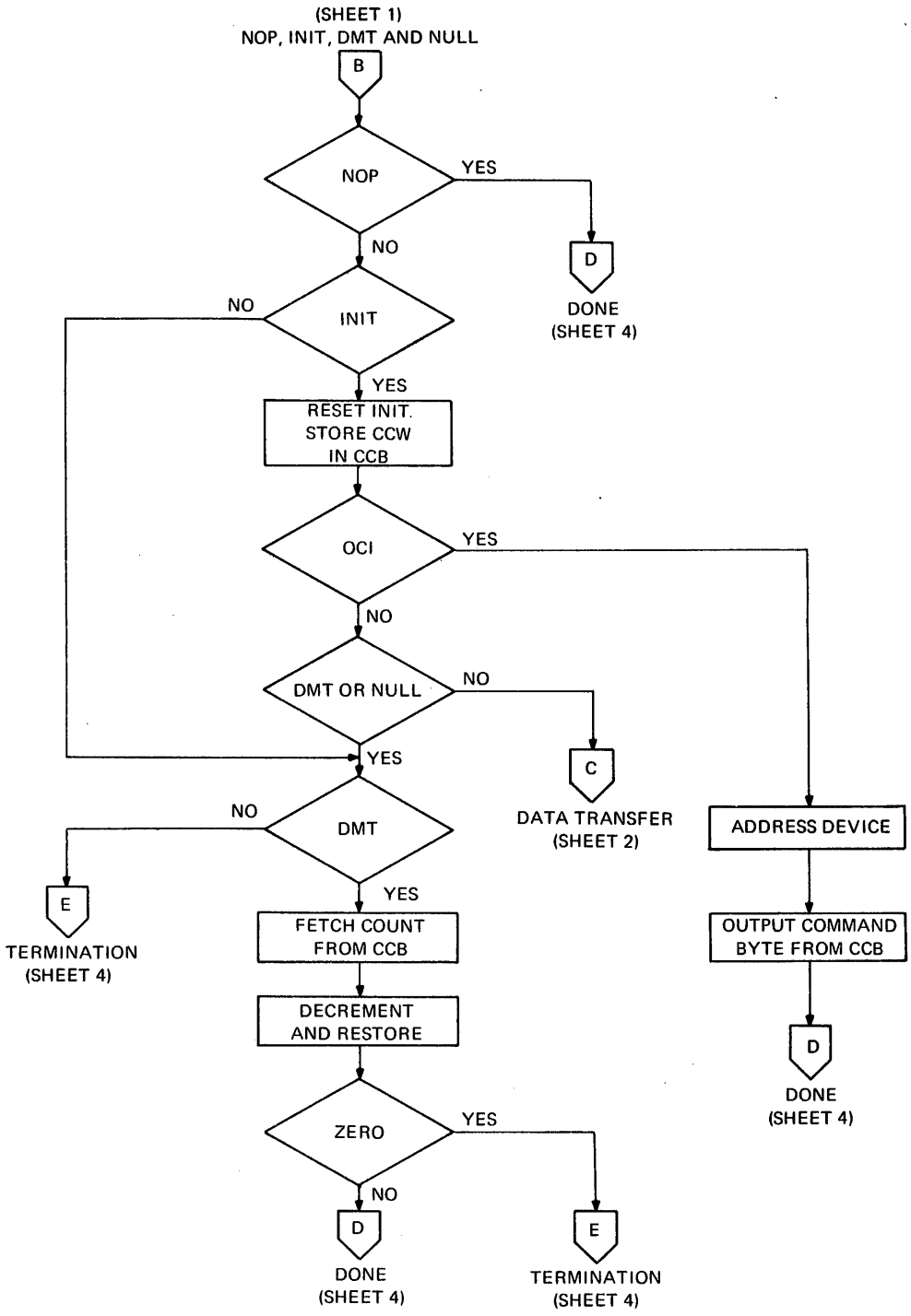
APPENDIX 8
AUTOMATIC I/O OPERATION



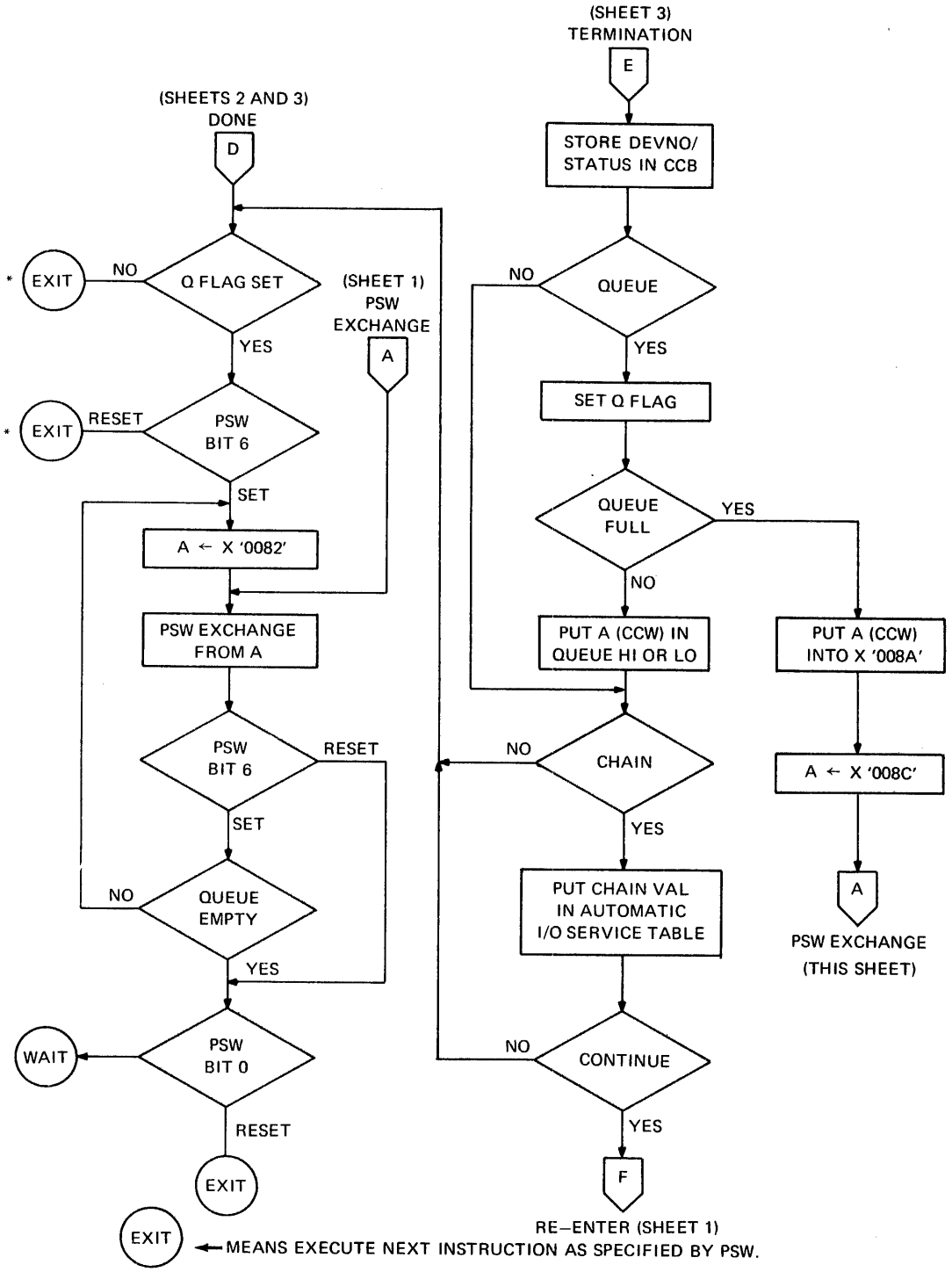
Sheet 1 of 4


(SHEET 1)
DATA TRANSFER
C
LOAD CURRENT ADDR AND FINAL ADDR FROM CCB
SET BYTE COUNT FROM CCW
ADDRESS DEVICE GET STATUS
STATUS BITS 4:7 = 0
NO
YES
SET INIT, NOP, Q BITS; RESET CHAIN, CONTINUE IN CCW
READ/ WRITE
READ
WRITE
READ DATA STORE IN CUR ADDR. INCR. CUR ADDR. DEC. BYTE COUNT
WRITE DATA FROM CUR ADDR. INCR CUR ADDR. DEC BYTE COUNT
NO
COUNT = 0
YES
YES
COUNT = 0
NO
STORE CUR. ADDR. INTO CCB
CUR > FINAL
YES
NO
SET NOP IN CCW
NO
T CHAR BIT SET
YES
STORE CCW IN CCB
D
DONE
(SHEET 4)
FETCH T CHAR FROM CCB
T CHAR = LAST CHAR
YES
NO
D
DONE
(SHEET 4)
E
TERMINATION
(SHEET 4)

Sheet 2 of 4


(SHEET 1)
NOP, INIT, DMT AND NULL
B
NOP
YES
NO
D
DONE
(SHEET 4)
INIT
NO
YES
RESET INIT.
STORE CCW
IN CCB
OCI
YES
NO
DMT OR NULL
NO
YES
C
DATA TRANSFER
(SHEET 2)
ADDRESS DEVICE
DMT
NO
YES
E
TERMINATION
(SHEET 4)
FETCH COUNT
FROM CCB
DECREMENT
AND RESTORE
OUTPUT COMMAND
BYTE FROM CCB
D
DONE
(SHEET 4)
ZERO
YES
NO
D
DONE
(SHEET 4)
E
TERMINATION
(SHEET 4)

Sheet 3 of 4


(SHEET 3)
TERMINATION
E
STORE DEVNO/
STATUS IN CCB
NO
QUEUE
YES
SET Q FLAG
QUEUE
FULL
YES
NO
PUT A (CCW) IN
QUEUE HI OR LO
PUT A (CCW)
INTO X '008A'
NO
CHAIN
YES
PUT CHAIN VAL
IN AUTOMATIC
I/O SERVICE TABLE
A ← X '008C'
A
PSW EXCHANGE
(THIS SHEET)
NO
CONTINUE
YES
F
RE–ENTER (SHEET 1)
(SHEETS 2 AND 3)
DONE
D
* EXIT
NO
Q FLAG SET
YES
(SHEET 1)
PSW
EXCHANGE
A
* EXIT
RESET
PSW
BIT 6
SET
A ← X '0082'
PSW EXCHANGE
FROM A
PSW
BIT 6
RESET
SET
NO
QUEUE
EMPTY
YES
WAIT
PSW
BIT 0
RESET
EXIT
EXIT
← MEANS EXECUTE NEXT INSTRUCTION AS SPECIFIED BY PSW.
* - IF INTERRUPT SIGNAL IS PRESENT, FIRMWARE WILL SERVICE IT
BEFORE EXECUTION OF THE NEXT INSTRUCTION.

Sheet 4 of 4

Illegal Instruction 7.25
Machine Malfunction 8.75
Normal I/O Interrupt 8.25
Immediate I/O Interrupt 8.75

Automatic Input/Output Interrupt Service Times

		NOP	NULL	DMT	OCI	READ	WRITE	BAD STATUS
	BASE	8.25	13.75	12.25	14.50	22.50+2.50n	22.25+2.25n	26.00
	INIT	-	2.50	2.50	-	2.50	2.50	2.50
	TCHAR No Match	-	-	-	-	2.50	2.50	-
1	TCHAR Match	-	-	-	-	1.25	1.25	-
1	COUNT=0 or CUR=FINAL	-	-	2.75	-	2.00	2.00	-
2	QUEUE.HI	-	9.50	9.50	-	9.50	9.50	-
2	QUEUE LW	-	9.75	9.75	-	9.75	9.75	-
2	CHAIN	-	3.00	3.00	-	3.00	3.00	-
2	CONT	-	Next -6.25	Next -6.25	-	Next -6.25	Next -6.25	-
2	QSVC INT	6.25	6.25	6.25	6.25	6.25	6.25	6.25

1. Reason for Termination
2. Termination Procedure

All times are given in microseconds. To determine the execution time of a particulat interrupt, add to the base time, the time for each pertinant option. For example: a Write of one character using a Termination Character test (TCHAR) with no match takes 22.50 Base
2.50 +2.50n (n=characters)
2.50 (TCHAR no match)
27.50

INDEX

PUBLICATION COMMENT FORM

Please use this postage-paid form to make any comments, suggestions, criticisms, etc. concerning this publication.

From _____ Date _____

Title _____ Publication Title _____

Company _____ Publication Number _____

Address _____

_____

_____

FOLD FOLD

Check the appropriate item.

☐ Error Page No. _____ Drawing No. _____

☐ Addition Page No. _____ Drawing No. _____

☐ Other Page No. _____ Drawing No. _____

Explanation:

FOLD FOLD

CUT ALONG LINE

Fold and Staple

No postage necessary if mailed in U.S.A.

STAPLE

STAPLE

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 22 OCEANPORT, N.J.

POSTAGE WILL BE PAID BY ADDRESSEE

PERKIN-ELMER

Computer Systems Division
2 Crescent Place
Oceanport, NJ 07757

TECH PUBLICATIONS DEPT. MS 322A

FOLD

FOLD

STAPLE

STAPLE