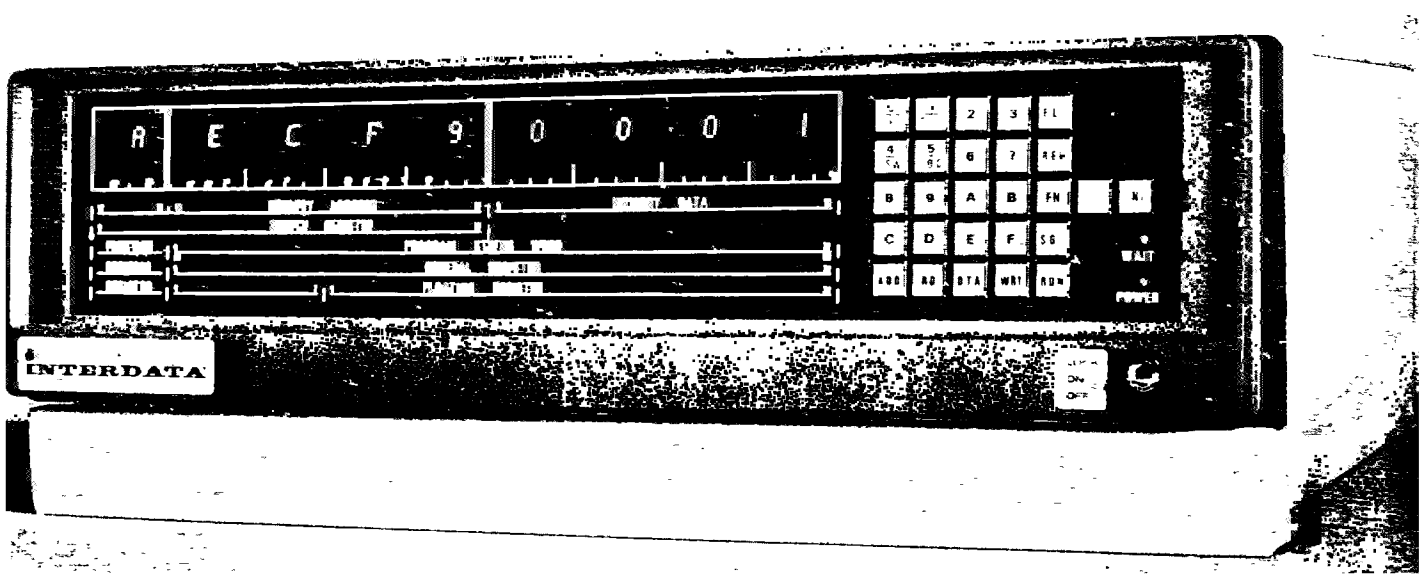
16 Bit Series
Reference Manual

INTERDATA®



16 Bit Series
Reference Manual

Publication Number 29-398R03

INFORMATION CONTAINED IN THIS
MANUAL IS SUBJECT TO DESIGN
CHANGE OR PRODUCT IMPROVEMENT

May 1975

TABLE OF CONTENTS

APPENDICES

ILLUSTRATIONS

TABLES

INDEX

CHAPTER 1

INTRODUCTION

INTRODUCTION

The INTERDATA 7/16 Processor is the first 16 bit, general purpose minicomputer to provide the instructional speed, I/O throughput and high level software and peripheral support of large systems. The 7/16 employs the latest techniques in logic design and system architecture. Included in the basic configuration are the 7/16 Processor, 8K bytes of memory, 16 general registers, 104 instructions, hardware interrupt vectoring for up to 255 devices; four high speed Direct Memory Access (DMA) channels, and power supply. A wide variety of standard, off the shelf options permit the user to tailor the system to meet both his present needs and future requirements. The overall efficiency of the 7/16 Processor makes it exceptionally well-suited for a wide variety of applications-from small dedicated processors to large multiuser systems.

MEMORY

The 7/16 main memory is built around core modules available in 8KB, 16KB, and 32KB versions to optimize system reliability and physical configurations. 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 in a single 15 inch printed circuit board, and occupies a single subassembly slot. The 8KB and 16KB memories have a 1.0 microsecond cycle time. The 32KB memory is available with either a 1.0 microsecond cycle time or a 750 nanosecond cycle time. Memory can be expanded by plugging in additional modules.

SYSTEM ARCHITECTURE

The INTERDATA 7/16 makes use of the powerful third generation architecture used by the other Processors in the INTERDATA family. 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 playing with "tricky code" to accomplish such basic functions as Exlucisve OR, multiple shifts, or byte processing.

The multi-accumulator architecture that INTERDATA pioneered in the minicomputer industry provides 16 general purpose registers for increased programming flexiblilty, and eliminates the needless accumulator housekeeping that is characteristic of machines with fewer accumulators. All 16 registers are available for use at the programmer's discretion. None of the 16 is dedicated to any specific purpose, such as index registers, stack pointers, program counter, or subroutine return pointers. The programmer is free to use the registers for storage of partical results, frequently used constants, loop management constants, or however else he sees fit.

The architectural design also provides 100% directly addressable memory, totally eliminating the time consuming design problems associated with paging and indirect addressing. Programmers can write straight-forward, simple, in-line code for the 7/16 without having to concern themselves with running out of base pages, and without having to waste memory with indirect address references.

INSTRUCTION SET

The instruction set for the 7/16 includes 104 individual instructions to provide the programmer with the tools he needs to write programs in as few steps as possible. The instruction uses both 16 and 32 bit instruction formats. It permits operations between any two general registers, between a general register and any memory location, between a general register and a 16 bit data constant carried in the instruction word, or between a general register and a four bit data constant contained in the instruction word.

The 7/16 includes a complete set of Arithmetic and Logical instructions. A complete set of Conditional Branch instructions permits branching to any location in memory without the use of skips. A full set of byte processing instructions simplifies handling of byte strings and, provides for more efficient use of available memory. The input/putput instructions permit operations between peripheral devices and general registers, or between devices and memory.

INPUT/OUTPUT SYSTEM

The INTERDATA 7/16 input/output system can handle up to 255 devices. High speed devices can operate at up to 2,000,000 bytes per second over the optional Selector Channel. Medium and low speed devices are connected to the standard multiplexor channel. These channels operate on a request-response basis to allow simple, reliable device controller design. INTERDATA offers a broad line of inexpensive peripherals for the 7/16 system that are both program and interface compatible with all members of the INTERDATA family. INTERDATA also offers standard, low-cost interface modules to aid the user in interface design.

SOFTWARE

Standard software available for the 7/16 includes: a symbolic Assembler, an interactive text Editor, an interactive Debug package, interactive FORTRAN, extended FORTRAN IV, utility programs, and the following four operating systems:

Basic Operating System (BOSS)
Disc Operating System (DOS)
Real Time Operating Systems (RTOS)
Mini Real Time Operating System (OS/16-MT)

In addition, the INTERDATA users' group, INTERCHANGE, has a large software library of its own that is available to 7/16 users.

OPTIONS AND PERIPHERALS

The 7/16 provides a flexible hardware system that can expand to meet the end user's requirements quickly and easily. As system demands and complexity increase, the 7/16 can be field expanded to provide the precise computational capability required. For example:

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 permits enabling or disabling memory writes into single or multiple 1K bytes blocks of memory under software control. The module interrupts the Processor when it detects a memory protect violation.

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

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

Automatic Loader provides a simple, single switch bootstrap load capability.

Turnkey Console provides switch control for power, initialize, and execution for the 7/16 in dedicated systems.

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

High Speed Arithmetic Logic Unit (HSALU) includes high speed signed multiply/divide, high speed hardware floating point, list processing instructions, privileged instructions detect, and improves standard instruction execution times up to 50%, depending on the individual instruction.

Stretch 32 field upgrades a 7/16 Processor to a software and I/O compatible 7/32 Processor capable of directly addressing 1 megabyte of memory and executing a full complement of 32 bit fullword instructions.

Other options include:

Selector Channel
TTY
Loader Storage Unit
Conformal Coating
Intertape Cassette System
Digital Multiplexor System
Analog to Digital System
Digital to Analog System
Universal Clock Module
Universal Clock Interface
I/O Bus Switch
IBM 360/370 Interface
Line Printers
Card Readers
Paper Tape Reader/Punch
Industry Compatible Magnetic Tapes
Disc Systems
Synchronous Data Set Interfaces
Asynchronous Data Set Interfaces

SUMMARY OF 7/16 FEATURES AND OPTIONS

The following chapters describe the 7/16 system including the High Speed ALU (HSALU) option. Of the features described, the 7/16 BASIC has:

Most Logical instructions
Most Fixed Point Arithmetic instructions
All Branch instructions
All Status Switching instructions
All I/O instructions

Not included in the 7/16 BASIC are list processing instructions, fixed point multiple and divide, Floating Point instructions, and the automatic I/O channel.

The interrupts associated with fixed point divide and floating point operations are not defined for the 7/16 BASIC. The simulate interrupt instruction can result only in an immediate interrupt. On a machine malfunction interrupt, the 7/16 BASIC Processor does not make a distinction between parity error on a data fetch and parity error on an instruction fetch.

CHAPTER 2

SYSTEM DESCRIPTION

The unique design characteristics of the INTERDATA 7/16 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 1 illustrates how the various elements of a 7/16 system are combined.

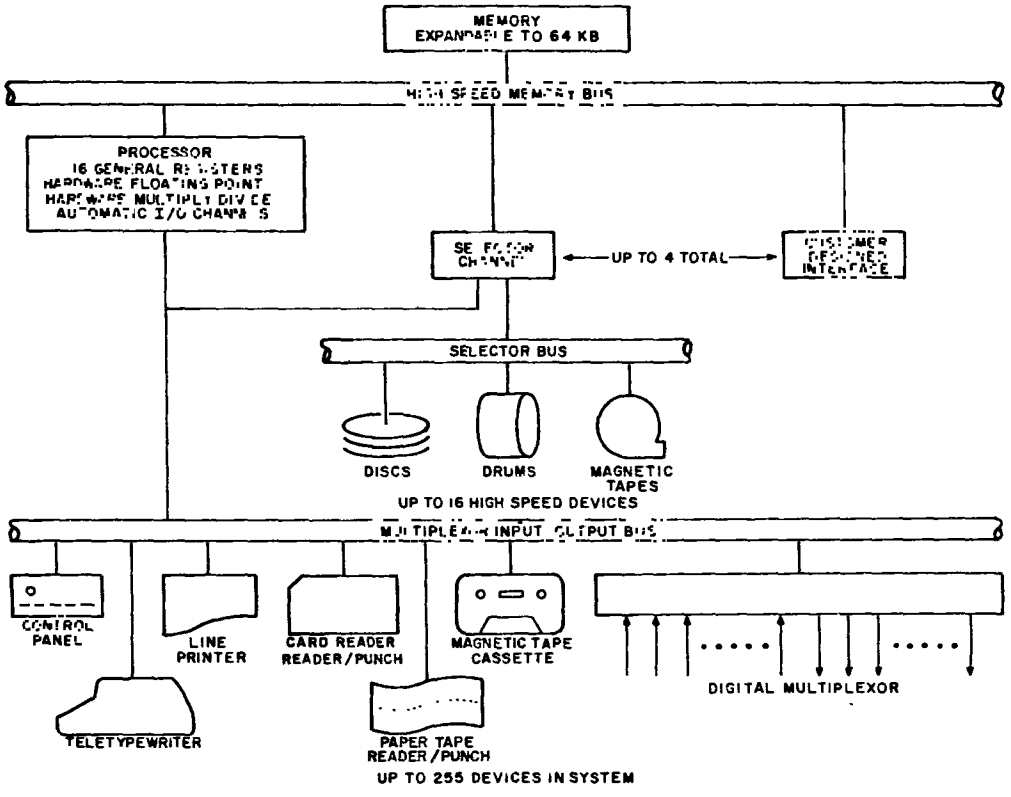


Figure 1. System Block Diagram

PROCESSOR

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
General Registers
Floating Point Registers (optional)
Floating Point Hardware (optional)
Signed multiply/divide hardware (optional)

Program Status Word

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

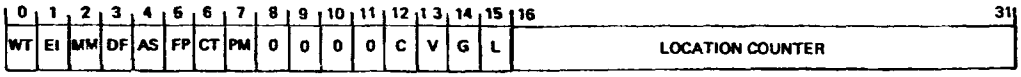


Figure 2. Program Status Word Format

Bits 0:15 are reserved for status information and for interrupt masks. Bits 16:31 contain the Location Counter. Unassigned Program Status Word bits must not be used, and must always be zero. Status information and interrupt mask bits are defined as follows:

Bit 0	Wait state
Bit 1	External interrupt mask
Bit 2	Machine malfunction interrupt mask
Bit 3	Fixed point divide fault interrupt mask
Bit 4	Automatic I/O and immediate interrupt mask
Bit 5	Floating point fault interrupt mask
Bit 6	Queue service interrupt mask
Bit 7	Protect mode
Bits 8:11	Not used, must be zero
Bits 12:15	Condition Code

Wait State

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

External Interrupt Mask

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 Auto Driver Channel.

Machine Malfunction Interrupt Mask

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

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

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.

Floating Point Fault Interrupt Mask

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

System Queue Service Interrupt Mask

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.

Protect Mode

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.

Condition Code

The bits in the Condition Code, Bits 12:15 of the Program Status Word, are set by the Processor to indicate the results of instruction execution. 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

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.

Floating Point Registers

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

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.

Reserved Memory Location

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

Location			Use
X'0000'	–	X'0021'	Reserved for Processor use
X'0022'	–	X'0023'	Register save pointer
X'0024'	–	X'0027'	Current PSW save area
X'0028'	–	X'002B'	Old PSW, Floating Point Fault
X;002C'	–	X'002F'	New PSW, Floating Point Fault
X'0030'	–	X'0033'	Old PSW, Illegal Instruction
X'0034'	–	X'0037	New PSW, Illegal Instruction
X'0038'	–	X'003B'	Old PSW, Machine Malfunction
X'003C'	–	X'003F'	New PSW, Machine Malfunction
X;0040'	–	X'0043'	Old PSW, External
X'0044'	–	X'0047'	New PSW, External
X'0048'	–	X'004B'	Old PSW, Fixed Point Fault
X'004C'	–	X'004F'	New PSW, Fixed Point Fault
X'0050'	–	X'007F'	Bootstrap Loader
X'0080'	–	X'0081'	System Queue Pointer
X'0082'	–	X'0085'	Old PSW, Channel Termination
X'0086'	–	X'0089'	New PSW, Channel Termination
X'008A'	–	X'008B'	System Queue Overflow Pointer
X'008C'	–	X;008F'	Old PSW, System Queue Overflow
X'0090'	–	X'0093'	New PSW, System Queue Overflow
X'0094'	–	X'0095'	Supervisor Call Argument Pointer
X'0096'	-	X'0099'	Old PSW, Supervisor Call
X'009A'	–	X'009B'	New Status, Supervisor Call
X'009C'	–	X'00BB'	New Location Counters, Supervisor Call
X'00BC'	–	X'00CF'	Reserved
X'00D0'	–	X'02CF'	Interrupt Service Table

These reserved locations play an important role in both interrupt and input/output processing. For details on these subjects refer to Chapters 7 and 8.

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 operation), 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 and 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, of which 8 bits are used for the sign and the exponent, and 24 bits are used for the fraction.

Logical Data

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

INSTRUCTION FORMATS

The INTERDATA instruction formats provide a concise method of representing required operations for easy interpretation by the Processor. There are four basic formats shown in Figure 3. The abbreviations used in Figure 3 have the following meanings:

OP	Operation Code
R1	First operand register
R2	Second operand register
N	A four bit immediate value
X2	Second operand index register
A2	Second operand direct address
I2	Second operand immediate value

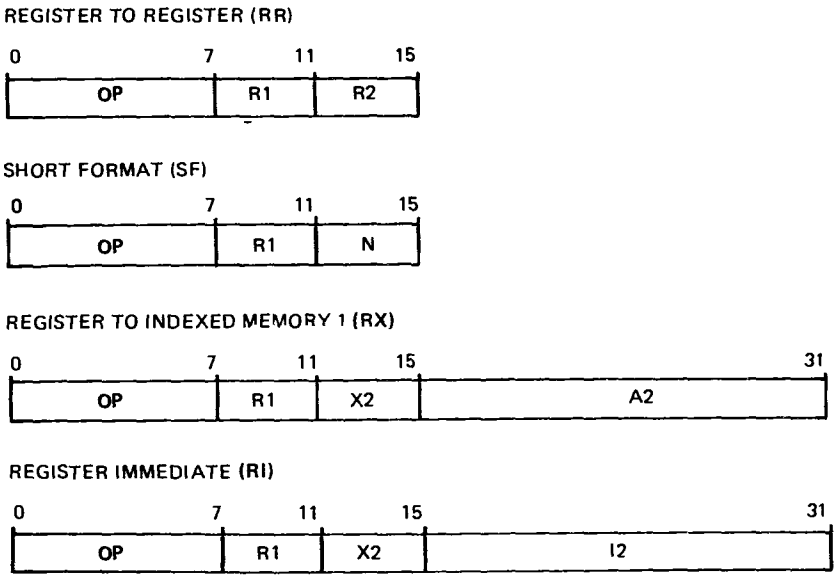


Figure 3. Instruction Formats

Most instructions in the 7/16 may be expressed in two or more formats, which provides flexibility in data organization and instruction sequencing.

In the examples accompanying each format description, it is assumed that proper values have been assigned to the symbols used in the assembler representation. Register specifications in these examples are expressed as absolute numbers to show the correspondence between the machine code format and the assembler notation. In actual practice, these numbers could be expressed symbolically.

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:

Assembler Notation		Machine Code
AHR	1,2	0A12

instructs the Processor to add the contents of Register 1 to the contents of Register 2, and store the result in Register 1.

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 used as the second operand. For example:

Assembler Notation		Machine Code
SIS	1,10	271A

instructs the Processor to subtract the quantity 10 from the contents of Register 1, and store the result in Register 1.

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 A2 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, to the contents of the A2 field. The A2 field may contain a maximum value of 65,535. For example:

Assembler Notation		Machine Code
SH	2,A2 (3)	4B233440

instructs the Processor to subtract from the contents of Register 2, the halfword quantity located in memory at the address obtained by adding X'3440' to the contents of Register 3. The result is stored in Register 2.

Register and Immediate Storage (RI) Format

This format represents a 32 bit instruction word. Bits 0:7 contain the operation code. Bits 8:11 contain the R1 field. Bits 12:15 contain the X2 field. Bits 16:31 contain the 16 bit immediate value, I2. In this format, the register specified by R1 contains the first operand. The second operand is obtained by adding the contents of the register specified by X2 to the value contained in the I2 field. For example:

Assembler Notation	Machine Code
AHI 1,I2 (2)	CA123444

instructs the Processor to add to the contents of Register 1 the quantity obtained by adding X'3444' to the contents of Register 2. The result replaces the contents of Register 1.

Branch Instruction Formats

The Branch instructions use the RR, SF, and RX formats. However, in the Conditional Branch instructions, the R1 field does not specify a register. Instead, it contains a mask value which is tested with the Condition Code. In the Short Form Branch instructions, the N field specifies the number of halfwords to be skipped.

CHAPTER 3

LOGICAL OPERATIONS

DATA FORMATS

The set of Logical instructions provide 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.

Logical data may be organized in 8 bit bytes, 16 bit halfwords, and 32 bit fullwords as shown in Figure 4.

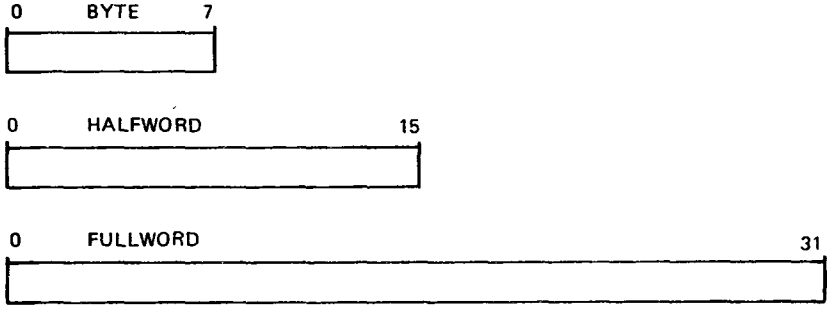


Figure 4. Logical Data

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	(logical product)
1 AND 0 = 0	
1 AND 1 = 1	
0 OR 0 = 0	
0 OR 1 = 1	(logical sum)
1 OR 0 = 1	
1 OR 1 = 1	
0 XOR 0 = 0	
0 XOR 1 = 1	(logical difference)
1 XOR 0 = 1	
1 XOR 1 = 0	

The list processing instructions manipulate a circular list as defined in Figure 5.

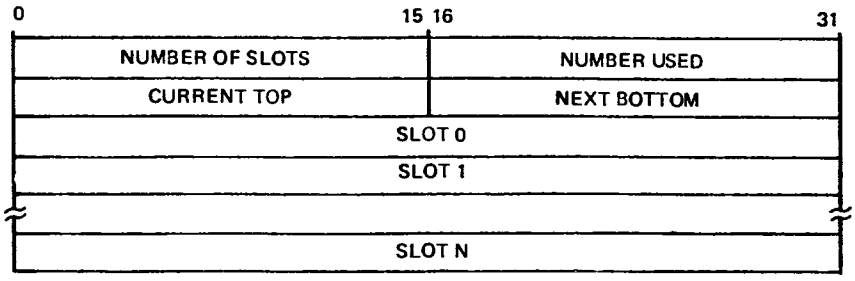


Figure 5. Circular List Definition

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. (Maximum slot designation is equal to 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 6.

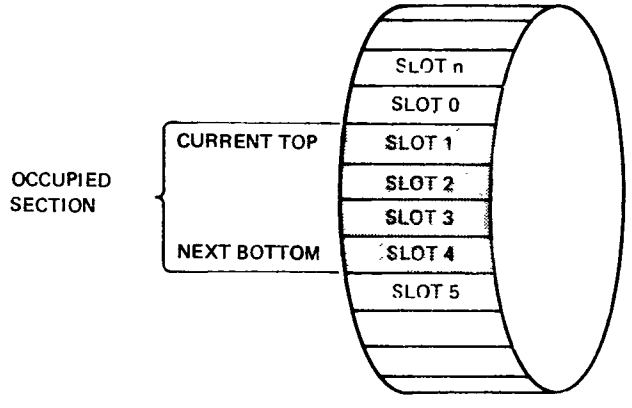


Figure 6. Circular List

LOGICAL INSTRUCTION FORMATS

The Logical instructions use the Register to Register (RR), the Short Format (SF), the Register and Indexed Storage (RX), and the Register and Immediate Storage (RI) formats.

LOGICAL INSTRUCTIONS

The instructions described in this section are:

LH	Load Halfword
LHR	Load Halfword Register
LHI	Load Halfword Immediate
LIS	Load Immediate Short
LCS	Load Complement Short
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
NH	AND Halfword
NHR	AND Halfword Register
NHI	AND Halfword Immediate
OH	OR Halfword
OHR	OR Halfword Register
OHI	OR Halfword Immediate
XH	Exclusive OR Halfword
XHR	Exclusive OR Halfword Register
XHI	Exclusive OR Halfword Immediate
THI	Test Halfword Immediate
CLH	Compare Logical Halfword
CLHR	Compare Logical Halfword Register
CLHI	Compare Logical Halfword Immediate
CLB	Compare Logical Byte
SLL	Shift Left Logical
SRL	Shift Right Logical
SLHL	Shift Left Halfword Logical
SLLS	Shift Left Logical Short
SRHL	Shift Right Halfword Logical
SRLS	Shift Right Logical Short
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

Instruction

Load Halfword
Load Halfword Register
Load Halfword Immediate
Load Immediate Short
Load Complement Short

Assembler Notation	Op-Code	Format
LH R1,A (X2)	48	RX
LHR R1, R2	08	RR
LHI R1, I2 (A2)	C8	RI
LIS R1, N	24	SF
LCS R1, N	25	SF

Operation

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

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 Notes

The Load Immediate Short instruction causes the four-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 four-bit second operand to be expanded to a 16 bit halfword with high order bits forced to zero. The two's complement value of this halfword replaces the contents of the register specified by R1.

When the load operations operate on fixed point data, the Condition Code indicates zero (no flags set), negative (L flag set), or positive (G flag set).

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

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

Instruction

Load Multiple

Assembler Notation		Op-Code	Format
LM R1,A (X2)	D	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.

Condition Code

Unchanged

Programming Note

The second operand must be located on a halfword boundary.

Instruction

Load Byte
Load Byte Register

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

Operation

The eight 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.

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

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

Operation

The two eight bit bytes contained in the register specified by R2 are exchanged and loaded into the registers specified by R1. Following execution of this instruction, the contents of R2 are unchanged.

Condition Code

Unchanged

Programming Note

R1 and R2 may specify the same register.

Instruction

Store Halfword

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 memory location specified by the effective address of the second operand.

Condition Code

Unchanged

Programming Note

The second operand location must be on a halfword boundary.

Instruction

Store Multiple

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.

Condition Code

Unchanged

Programming Note

The second operand location must be on a halfword boundary.

Instruction

Store Byte
Store Byte Register

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.

Condition Code

Unchanged

Programming Note

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

Instruction

AND Halfword
AND Halfword Register
AND Halfword Immediate

Assembler Notation		Op-Code	Format
NH	R1, A (X2)	44	RX
NHR	R1, R2	04	RR
NHI	R1, I2 (X2)	C4	RI

Operation

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

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.

Instruction

OR Halfword
OR Halfword Register
OR Halfword Immediate

Assembler Notation		Op-Code	Format
OH	R1, A (X2)	46	RX
OHR	R1, R2	06	RR
OHI	R1, I2 (X2)	C6	RI

Operation

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

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.

Instruction

Exclusive OR Halfword
Exclusive OR Halfword Register
Exclusive OR Halfword Immediate

Assembler Notation		Op-Code	Format
XH	R1, A (X2)	47	RX
XHR	R1, R2	07	RR
XHI	R1, I2 (X2)	C7	RI

Operation

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

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.

Instruction

Test Halfword Immediate

Assembler Notation		Op-Code	Format
THI	R1, I2 (X2)	C3	RI

Operation

Each bit in the 16 bit second operand is logically ANDed with the corresponding bit in the general register specified by R1. The contents of the register specified by R1 and the second operand are unchanged.

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

Instruction

Compare Logical Halfword
Compare Logical Halfword Register
Compare Logical Halfword Immediate

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

Operation

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

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.

Instruction

Compare Logical Byte

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

Operation

The byte quantity, contained in Bits 7:15 of the register specified by R1, is compared with the second operand. The result is indicated by the Condition Code settings. Neither operand is changed.

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

The state of the V flag is undefined.

Instruction

Shift Left Logical

Assembler Notation		Op-Code	Format
SLL	R1, I2 (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 Condtiion 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

Programming Notes

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.

When the first operand is 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 state of the C flag is undefined.

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

Instruction

Shift Right Logical

Assembler Notation		Op-Code	Format
SRL	R1,A (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

Programming Notes

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.

When the first operand is 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 state of the carry flag is undefined.

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

Instruction

Shift Left Halfword Logical
Shift Left Logical Short

Assembler Notation		Op-Code	Format
SLHL	R1, I2 (X2)	CD	RI
SLLS	R1, N	01	SF

Operation

The first operand, the contents of the register specified by R1, is shifted left the number of places specified by the second operand. Bits shifted out of Position 0 are shifted through the carry flag of the Condition Code, and then lost. The last bit shifted remains in the carry flag. Zeros are moved 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

Programming Notes

In the R1 format, the shift count is specified by the least significant four bits of the second operand.

In the SF format, the maximum shift count is 15.

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

When the first operand is 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 state of the C flag is undefined.

Instruction

Shift Right Halfword Logical
Shift Right Logical Short

Assembler Notation		Op-Code	Format
SRHL	R1, I2 (X2)	CC	RI
SRLS	R1, N	90	SF

Operation

The first operand, the contents of the register specified by R1, is shifted right the number of places specified by the second operand. Bits shifted out of Postion 15 are shifted through the carry flag of the Condition Code, and then 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

Programming Notes

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

In the SF format, the maximum shift count is 15.

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

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 state of the C flag is undefined.

Instruction

Rotate Left Logical

Assembler Notation		Op-Code	Format
RLL	R1, I2 (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.

Instruction

Rotate Right Logical

Assembler Notation		Op-Code	Format
RRL	R1,I2 (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.

Instruction

Add to Top of List
Add to Bottom of List

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 of the second operand. The number of slots 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 sucessfully
0	1	0	0	List overflow

Programming Notes

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

The second operand location must be on a halfword boundary.

Instruction

Remove from Top of List
Remove from Bottom of List

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

Operation

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

Condition Code

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

Programming Notes

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

The second operand location must be on a halfword boundary.

CHAPTER 4

BRANCHING

OPERATIONS

In normal operation, 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.

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 by the instruction as the second operand address.

Decision Making

The Conditional Branch instructions permit the program to make decisions based on previous results. In these instructions, the R1 field contains a four 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.

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 memory location 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) formats.

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
BRC	Branch on True Condition
BRCR	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

Instruction

Branch on False Condition
Branch on False Condition Register
Branch on False Condition Backward Short
Branch on False Condition Forward Short

Assembler Notation		Op-Code	Format
BFC	M1, A2 (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 taken to the second operand location. If any of the conditions tested is found to be true, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Notes

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.

Instruction

Branch on True Condition
Branch on True Condition Register
Branch on True Condition Backward Short
Branch on True Condition Forward Short

Assembler Notation		Op-Code	Format
BTC	M1,A2 (x2)	42	RX
BTCR	M1,A2 (X2)	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.

Condition Code

Unchanged

Programming Notes

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.

Instruction

Branch and Link
Branch and Link Register

Assembler Notation		Op-Code	Format
BAL	R1,A2 (X2)	41	RX
BALR	R1, R2	01	RR

Operation

The address of the next halfword location following the Branch and Link instruction word is placed in the register specified by R1, and branch is taken to the second operand location.

Condition Code

Unchanged

Programming Notes

The branch address is obtained before the register specified by R1 is changed. This allows either R1 and X2 or R1 and R2 to specify the same register.

The second operand location must be on a halfword boundary.

Instruction

Branch on Index Low or Equal

Assembler Notation		Op-Code	Format
BXLE	R1,A2 (X2)	C1	RX

Operation

Prior to the execution of this instruction, the register specified by R1 must contain a 16 bit starting index value. The register specified by R1+1 must contain a 16 bit increment value. The register specified by R1+2 must contain a 16 bit comparand (limit or final value). All values may be signed.

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.

Condition Code

Unchanged

Programming Notes

The branch address is obtained before the register specified by R1 is changed.

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.

Instruction

Branch on Index High

Assembler Notation		Op-Code	Format
BXH	R1,A2 (X2)	C0	RX

Operation

Prior to the execution of this instruction, the register specified by R1 must contain a 16 bit starting index value. The register specified by R1+1 must contain a 16 bit increment value. The register specified by R1+2 must contain a 16 bit comparand (limit or final value). All values may be signed.

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.

Condition Code

Unchanged

Programming Notes

The branch address is obtained before the register specified by R1 is changed.

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.

CHAPTER 5

FIXED POINT ARITHMETIC

DATA FORMATS

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

There are two formats for fixed point data: the halfword, and the 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. Refer to Figure 7.

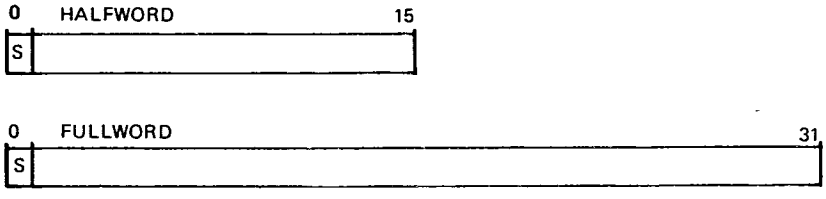


Figure 7. Fixed Point Data Words Formats

Positive values are represented in true binary form with a Sign bit of zero. Negative numbers 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:

Change all zeros to ones, and all ones to zeros

Add one

The quantity zero is represented with all bits zero.

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 result.

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 Storage (RI) formats.

FIXED POINT INSTRUCTIONS

The instructions described in this section are:

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

Instruction

Add Halfword
Add Halfword Register
Add Halfword Immediate
Add Immediate Short

Assembler Notation		Op-Code	Format
AH	R1,A2 (X2)	4A	RX
AHR	R1, R2	0A	RR
AHI	R1, I2 (X2)	CA	RI
AIS	R1,N	26	SF

Operation

The second operation is added algebraically to the contents of the register specified by R1. The result of this 16 bit addition replaces the contents of the register specified by R1.

Condition Code

C	V	G	L	
X	0	0	1	Result is zero
X	0	0	1	Result is not less than zero
X	0	1	0	Result is greater than zero
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

Programming Notes

The second operand for the Add Immediate Short instruction is obtained by expanding the four-bits 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.

Instruction

Add Halfword to Memory

Assembler Notation		Op-Code	Format
AHM	R1,A2 (X2)	61	RX

Operation

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

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

The second operand must be located on a halfword boundary.

Instruction

Add with Carry Halfword
Add with Carry Halfword Register

Assembler Notation		Op-Code	Format
ACH	R1, A2 (X2)	4E	RX
ACHR	R1, R2	0E	RR

Operation

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

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 Notes

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.

Instruction

Subtract Halfword
Subtract Halfword Register
Subtract Halfword Immediate
Subtract Immediate Short

Assembler Notation		Op-Code	Format
SH	R1,A2 (X2)	4B	RX
SHR	R1,R2	0B	RR
SHI	R1,I2 (X2)	CB	RI
SIS	R1,N	27	SF

Operation

The halfword second operand is subtracted algebraically from the contents of the register specified by R1. The result replaces the contents of the 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
X	1	X	X	Arithmetic overflow
1	X	X	X	Borrow

Programming Notes

The second operand for the Subtract Immediate Short instruction is obtained by expanding the four-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.

Instruction

Subtract with Carry Halfword
Subtract with Carry Halfword Register

Assembler Noatation		Op-Code	Format
SCH	R1,A2 (X2)	4F	RX
SCHR	R1,R2	0F	RR

Operation

The 16 bit second operand with the carry bit 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.

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
X	1	X	X	Carry

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.

Instruction

Compare Halfword
Compare Halfword Register
Compare Halfword Immediate

Assembler Notation		Op-Code	Format
CH	R1,A2 (X2)	49	RX
CHR	R1,R2	09	RR
CHI	R1, I2 (X2)	C9	RI

Operation

The first operand, contained in the register specified by R1, is compared algebraically to the 16 bit second operand. The result is indicated by the Condition Code settings. Neither operand is changed.

Condition Code

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

Programming Note

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

Instruction

Multiply Halfword
Multiply Halfword Register

Assembler Notation		Op-Code	Format
MH	R1,A2 (X2)	4C	RX
MHR	R1,R2	0C	RR

Operation

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

Condition Code

Unchanged

Programming Notes

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

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

Instruction

Multiply Halfword Unsigned
Multiply Halfword Unsigned Register

Assembler Notation		Op-Code	Format
MHU	R1,A2 (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. The second operand is unchanged.

Condition Code

Unchanged

Programming Notes

The R1 field must specify an even numbered register.

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

Instruction

Divide Halfword
Divide Halfword Register

Assembler Notation		Op-Code	Format
DH	R1,A2 (X2)	4D	RX
DHR	R1,R2	0D	RR

Operation

The 32 bit dividend contained in the registers specified by R1 and R1+1 is divided by the halfword divisor. The 16 bit signed remainder replaces the contents of the register specified in R1. The 16 bit signed quotient replaces the contents of the register specified by R1+1.

Condition Code

Unchanged

Programming Notes

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

If the divisor is zero, the instruction is aborted before the destination registers are changed. The fixed point fault interrupt is taken, if enabled by Bit 3 of the current PSW.

If the quotient cannot be expressed in a halfword (a Sign bit and 15 magnitude bits) the instruction is aborted before the destination registers are changed. The fixed point fault interrupt is taken, if enabled by Bit 3 of the current PSW.

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

Instruction

Shift Left Arithmetic

Assembler Notation		Op-Code	Format
SLA	R1,I2 (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 Notes

R1 must specify an even numbered register.

The least significant five bits of the second operand determine the shift count.

If the second operand specifies a shift count of zero, the state of the C flag is undefined.

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

Instruction

Shift Left Halfword Arithmetic

Assembler Notation		Op-Code	Format
SLHA	R1,I2 (X2)	CF	RI

Operation

Bits 1:15 of the first operand, contained in the register specified by R1, are shifted left the number of binary places specified by the second operand. The Sign bit (Bit 0) 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 Notes

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

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

If the second operand specifies a shift of zero places, the state of the C flag is undefined.

Instruction

Shift Right Arithmetic

Assembler Notation	Op-Code	Format
SRA	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 and 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 Notes

R1 must specify an even numbered register.

The least significant five bits of the second operand determine the shift count.

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

If the second operand specifies a shift count of zero, the state of the C flag is undefined.

Instruction

Shift Right Halfword Arithmetic

Assembler Notation		Op-Code	Format
SRHA	R1,I2 (X2)	CE	RI

Operation

Bits 1:15 of the first operand, contained in the register specified by R1, are shifted right the number of binary places specified by the second operand. The Sign bit (Bit 0) remains unchanged and is propagated right as many places as specified by the second operand. Bits shifted out of Position 15 are shifted through the carry flag, and then 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 state of the C flag indicates the state of the last bit shifted.

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

If the second operand specifies a shift of zero places, the state of the C flag is undefined.

CHAPTER 6

FLOATING POINT ARITHMETIC

DATA FORMATS

The Floating Point Arithmetic instructions provide a means for rapid manipulation of the scientific data expressed as floating point numbers. In addition to the usual operations of add, subtract, multiply, divide, and compare, the floating point set includes instructions for loading and storing floating point operands.

Floating point data is expressed in excess-64 notation. Each floating point number consists of a Sign bit, an exponent field, and a fraction, as shown in Figure 8.

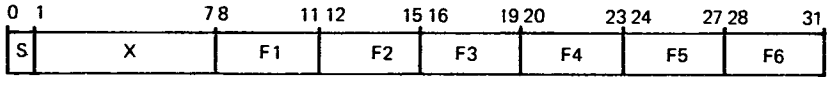


Figure 8. Floating Point Data Format

This form of representation requires 32 bits. The Sign bit indicates whether the floating point value is positive or negative. The exponent field indicates the power of 16 by which the fraction is to be multiplied to produce the floating point value. In excess-64 notation, an exponent field of 64 (X'40') indicates that the fraction is to be multiplied by 16^0. An exponent field of 63, (X'3F') indicates that the fraction is to be multiplied by 16^{-1}. An exponent field of 65 (X'41') indicates that the fraction is to be multiplied by 16^1. Floating point numbers may range in absolute value from 5.4×10^{-79} through 7.5×10^{75}.

Normalization

The process of normalizing floating point quantities allows values to be represented with the greatest possible precision. In the normalization process, the floating point fraction is shifted left one hexadecimal digit (four-bits) at a time until the most significant hexadecimal digit of the fraction is non-zero. The exponent is decremented by one for each hexadecimal digit shifted.

Exponent Overflow and Underflow

Exponent overflow results when a floating point operation produces an exponent greater than 63 (exponent field greater than 127 or X'7F'). On overflow, the floating point result is forced to the maximum absolute value, X'7FFF FFFF'. The Sign bit is not changed, and may be either zero or one.

Exponent underflow results when a floating point operation, including normalization, produces an exponent less than -64, (exponent field less than zero). On underflow, the result is forced to true zero, X'0000 0000'.

Conversion from Decimal

The process of converting a decimal number into the excess-64 notation 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.

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

3. Combine the hexadecimal integer and fraction.

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

4. Shift the radix point.

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

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 bits for the fraction.

   42B66. = 0100 0010 1011 0110 0110 0000 0000 0000

FLOATING POINT INSTRUCTIONS FORMATS

The Floating Point instruction use the Register to Register (RR), and the Register and Indexed Storage (RX) instruction formats. In all the RR format instructions, the R1 and R2 fields specify one of the floating point registers. There are eight floating point registers numbered 0, 2, 4, 6, 8, 10, 12, and 14. In the RX 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 instructions described in this section are:

LE	Load
LER	Load Register
STE	Store
AE	Add
AER	Add Register
SE	Subtract
SER	Subtract Register
CE	Compare
CER	Compare Register
ME	Multiply
MER	Multiply Register
DE	Divide
DER	Divide Register

Instruction

Load
Load Register

Assembler Notation		Op-Code	Format
LE	R1,A2(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.

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 Notes

Normalization 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 L flags are reset, and if enabled by Bit 5 of the current PSW, the arithmetic fault interrupt is taken.

If the fraction of the second operand is zero, the result is forced to zero, and the C, V, G, and L flags are reset.

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

Instruction

Store

Assembler Notation		Op-Code	Format
STE	R1,A2(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.

Condition Code

Unchanged

Programming Note

The second operand must be located on a fullword boundary.

Instruction

Add
Add Register

Assembler Notation		Op-Code	Format
AE	R1,A2(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 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 register specified by R1.

If the addition of fractions does not produce a carry, the result is normalized. The normalized result is replaced by the contents of the register specified by R1.

Condition Code

C	V	G	L	
0	X	0	0	Floating point result is zero
0	X	0	1	Floating point result is less than zero
0	X	1	0	Floating point result is greater than zero
0	1	X	X	Exponent overflow
0	1	0	0	Exponent underflow

Programming Notes

When the addition of the fraction 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 the L flag, is set in the Condition Code, and if enabled by Bit 5 of the current PSW, the floating point 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 floating point fault interrupt is taken.

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

Instruction

Subtract
Subtract Register

Assembler Notation		Op-Code	Format
SE	R1, Z2(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 incremented by one for each hexadecimal shift until the two exponents are equal. The fractions are then subtracted algebraically.

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

Condition Code

C	V	G	L	
0	X	0	0	Floating point result is zero
0	X	0	1	Floating point result is less than zero
0	X	1	0	Floating point result is greater than zero
0	1	X	X	Exponent overflow
0	1	0	0	Exponent underflow

Programming Notes

When the subtraction of the fractions produce 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 the L flag, is set in the Condition Code, and if enabled by Bit 5 of the current PSW, the floating point 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 floating point fault interrupt is taken.

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

Compare
Compare Register

Assembler Notation		Op-Code	Format
CE	R1,A2(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.

Condition Code

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

Programming Note

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

Instruction

Multiply
Multiply Register

Assembler Notation		Op-Code	Format
ME	R1,A2(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. 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 register specified by R1.

Condition Code

C	V	G	L	
0	X	0	0	Floating point result is zero
0	X	0	1	Floating point result is less than zero
0	X	1	0	Floating point result is greater than zero
X	1	X	X	Exponent overflow
X	1	0	0	Exponent underflow

Programming Notes

The addition of exponents may produce exponent overflow. In this case, the result is forced to the maximum value, $\pm$X'7FFF FFFF'. The V flag in the Condition Code is set, along with the G or the L flag, depending upon the sign of the result. A floating point fault interrupt is taken, if enabled by Bit 5 of the current PSW.

The addition of exponents and 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 the L flags are always reset, and if enabled by Bit 5 of the current PSW, the floating point fault interrupt is taken.

If the first operand or the second operand is zero, the result is forced to zero and C, V, G and L flags are reset.

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

Instruction

Divide
Divide Register

Assembler Notation		Op-Code	Format
DE	R1,A2(X2)	6D	RX
DER	R1,R2	2D	RR

Operation

The exponent of the second operand is subtracted from the exponent of the first operand to produce the exponent of the result. This exponent is converted back to excess-64 notation.

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

Condition Code

C	V	G	L	
0	X	0	0	Floating point result is zero
0	X	0	1	Floating point result is less than zero.
0	X	1	0	Floating point result is greater than zero
0	1	X	X	Exponent overflow
0	1	0	0	Exponent underflow
1	1	0	0	Divisor equal to 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 the V flags are set in the Condition Code. The G and the L are reset. If enabled by Bit 5 of the current PSW, the floating point fault interrupt is taken.

The subtraction of exponents may produce exponent overflow. In this case, the result is forced to the maximum value, $\pm$X'7FFF FFFF'. The V flag in the Condition Code is set, along the G or the L flag, depending on the sign of the result. A floating point fault interrupt is taken if enabled by Bit 5 of the current PSW.

The subtraction of exponents and 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 the L flags are always reset, and if enabled by Bit 5 of the current PSW, the floating point fault interrupt is taken.

If the first operand is zero, the result is forced to zero, and the C, V, G and L flags are reset.

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

CHAPTER 7

STATUS SWITCHING AND INTERRUPTS

STATUS SWITCHING AND INTERRUPTS

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 singly

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 9.

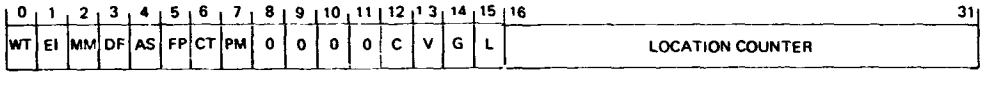


Figure 9. Program Status Word Format

Bits 0:15 of the PSW are reserved for status definitions. Note that Bits 8:11 are not currently assigned specific functions. These bits must always be zero. 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	(WT)	Wait state
Bit 1	(EI)	External interrupt mask
Bit 2	(MM)	Machine malfunction interrupt mask
Bit 3	(DF)	Fixed point fault interrupt mask
Bit 4	(AS)	Automatic I/O and immediate interrupt mask
Bit 5	(FP)	Floating point fault interrupt mask
Bit 6	(CT)	Queue service interrupt mask
Bit 7	(PM)	Protect mode

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.

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 INTERDATA 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 location 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
0	0	1	0	Parity error on instruction read
0	1	0	0	Parity error on data read

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'. The contents of the general registers are saved in 16 successive halfword locations starting at the address specified in memory location X'0022' - X'0023'.

When power returns, the Processor restores the PSW 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.

During write operations to memory, the parity bit of each memory word is set to maintain odd parity. The parity bit is recomputed on each memory read. If the computed bit is not equal to the bit read out of memory, the Processor takes a machine malfunction interrupt, setting the G or the L flag to indicate error on instruction read or on data read.

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 has been changed. The current PSW is saved in memory locations X'0028' - X'0031'. The new PSW, contained in memory locations X'0032' - X'0035', 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 requests 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 device 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'0D00', and extends through location X'02CF'. (Chapter 8, Input/Output Operations, contains detailed descriptions of the make-up of this table and its use in both interrupt driver 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. If the location contains an even value, 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 address of the routine is placed in the service pointer table locations for each device.

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.

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

This 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.

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 instructions.

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 enabled. When the Processor executes a Supervisor Call instruction, it:

Saves the current PSW in memory locations X'0096' — X'099'.

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 Processors:

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 terminators 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 are described in this section as:

LPSW	Load Program Status Word
EPSR	Exchange Program Status Register
SINT	Simulate Interrupt
SVC	Supervisor Call

Instruction

Load Program Status Word

Assembler Notation		Op-Code	Format
LPSW	0,A2 (X2)	C2	RX

Operation

The 32 bit second operand becomes the current Program Status Word.

Condition Code

Determined by the new PSW

Programming Notes

The R1 field of this instruction is not used by the Processor, and it should be zero or blank.

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

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.

Condition Code

Determined by the new status

Programming Notes

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

Assembler Notation		Op-Code	Format
SINT	0,I2(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 Notes

The R1 field of this instruction must contain zero.

This instruction is a privileged operation.

Instruction

Supervisor Call

Assembler Notation		Op-Code	Format
SVC	R1, A3(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 value found at the resulting address becomes the new Location Counter.

Condition Code

Determined by the new status

Programming Note

The second operand must be located on a halfword boundary.

CHAPTER 8

INPUT OUTPUT OPERATIONS

INPUT/OUTPUT OPERATIONS

Input/output operations, as defined for the INTERDATA 7/16, 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 I/O, or Multiplexor Bus. Data transfers to or from external devices may be performed in the byte mode, the halfword mode, or the burst mode. Byte and halfword transfers require Processor intervention, either programmed or automatic, for each item transferred. Burst mode transfers, which require a Selector Channel, proceed independently of the Processor.

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 may be attached directly to the I/O Bus or they may be attached to the I/O Bus indirectly, through a Selector Channel. 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 may result in either an immediate interrupt or an automatic channel operation. The Processor chooses between these two options according to information contained in the interrupt service pointer table.

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 acknowledge 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, or the entry reserved for the device requesting attention.

If the entry in the table is even (Bit 15 equals zero) the Processor takes an immediate interrupt, and 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.

At the time the Processor transfers control to the software routines, 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:

AI	Acknowledge Interrupt
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
Acknowledge Interrupt Register

Assembler Notation		Op-Code	Format
AI	R1,A2(X2)	DF	RX
AIR	R1,R2	9F	RR

Operation

The address of the interrupting device replaces the contents of the register specified by R1. The eight 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.

Condition Code

C	V	G	L	
1	X	X	X	Device busy
X	1	X	X	Examine status or time-out
X	X	1	X	End of medium
X	X	X	1	Device unavailable

Programming Notes

The Condition Code settings described above assume standard INTERDATA device controllers.

These instructions are privileged operations.

Instruction

Sense Status
Sense Status Register

Assembler Notation		Op-Code	Format
SS	R1, A2(X2)	DD	RX
SSR	R1, R2	9D	RR

Operation

Bits 8:15 of the register specified by R1 contain the eight bit device address. The device is addressed, and the eight 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.

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 Notes

The Condition Code interpretations of status assume standard INTERDATA 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.

Instruction

Output Command
Output Command Register

Assembler Notation		Op-Code	Format
OC	R1,A2(X2)	DE	RX
OCR	R1,R2	9E	RR

Operation

Bits 8:15 of the register specified by R1 contain the eight bit device address. The Processor addresses the device and transmits an eight bit command byte from the second operand location to the device. Neither operand is changed.

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, Bits 8:15 of the register specified by R2 contain the device command.

These instructions are privileged operations.

Instruction

Read Data
Read Data Register

Assembler Notation

		Op-Code	Format
RD	R1,A2(X2)	DB	RX
RDR	R1,R2	9B	RR

Operation

Bits 8:15 of the register specified by R1 contain the eight bit device address. The Processor addresses the device. The device responds by transmitting an eight-bit data byte. This byte is placed in the second operand location.

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 eight 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.

Instruction

Read Halfword
Read Halfword Register

Assembler Notation		Op-Code	Format
RH	R1,A2(X2)	D9	RX
RHR	R1,R2	99	RR

Operation

Bits 8:15 of the register specified by R1 contain the eight 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 eight bit bytes in successive operations.

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; 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.

Instruction

Read Block

Assembler Notation		Op-Code	Format
RB	R1,A2(Z2)	D7	RX

Operation

Bits 8:15 of the register specified by R1 contain the eight 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 eight 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

Programming Notes

The Condition Code interpretations of status assume standard INTERDATA device 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, the one data byte is transmitted.

The Processor is in a non-interruptable state during the transfer.

This instruction is a privileged operation.

Instruction

Read Block Register

Assembler Notation		Op-Code	Format
RBR	R1,R2	97	RR

Operation

Bits 8:15 of the register specified by R1 contain the eight 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 eight 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

Programming Notes

The maximum value for R1 is 14.

The Condition Code interpretations of status assume standard INTERDATA 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.

Instruction

Write Data
Write Data Register

Assembler Notation		Op-Code	Format
WD	R1,A2(X2)	DA	RX
WDR	R1,R2	9A	RR

Operation

Bits 8:15 of the register specified by R1 contain the eight bit device address. The Processor addresses the device, and transmits an eight bit data byte from the second operand location to the device. Neither operand is changed.

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 byte is taken from Bits 8:15 of the register specified by R2.

These instructions are privileged operations.

Instruction

Write Halfword
Write Halfword Register

Assembler Notation		Op-Code	Format
WH	R1,A2(X2)	D8	RX
WHR	R1,R2	98	RR

Operation

Bits 8:15 of the register specified by R1 contain the eight 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 eight bit data bytes in successive operations.

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; 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 privileged operations.

Instruction

Write Block

Assembler Notation		Op-Code	Format
WB	R1,A2(X2)	D6	RX

Operation

Bits 8:15 of the register specified by R1 contain the eight 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 eight 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 Notes

The Condition Code interpretations of status assume standard INTERDATA 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.

Instruction

Write Block Register

Assembler Notation		Op-Code	Format
WBR	R1,R2	96	RR

Operation

Bits 8:15 of the register specified by R1 contain the eight 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 eight 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 Notes

The maximum value for R2 is 14.

The Condition Code interpretations of status assume standard INTERDATA 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.

Instruction

Autoload

Assembler Notation		Op-Code	Format
AL	0,A2(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'.

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 Notes

The R1 field of this instruction must be zero.

The Condition Code interpretations of status assume standard INTERDATA device controllers.

This instruction is a privileged operation.

CONTROL OF I/O OPERATIONS

The design of the 7/16 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.

INTERDATA 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. 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 can 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 7/16 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 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 hardware malfunction or by improper use of the simulate interrupt instruction.)

The next step is to prepare the device for the 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 servcie 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 transfer, and
4. Restore the registers, and
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 INTERDATA hardware allows interrupts to be disabled at either the device level or the Processor level, nesting of interrupts is both possible and practical.

Software Vectoring

Software vectoring of interrupts is provided for compatibility with previous INTERDATA 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 and get back the device address.
3. Transfer to an appropriate subrouting, 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 complete for memory cycles.

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 communications 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 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. 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 intergral 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 private 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.

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, this 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. 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 Figure 10.

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 of an immediate interrupt software routine. If Bit 15 of the entry is one, then the entry minus one is the address 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 11.

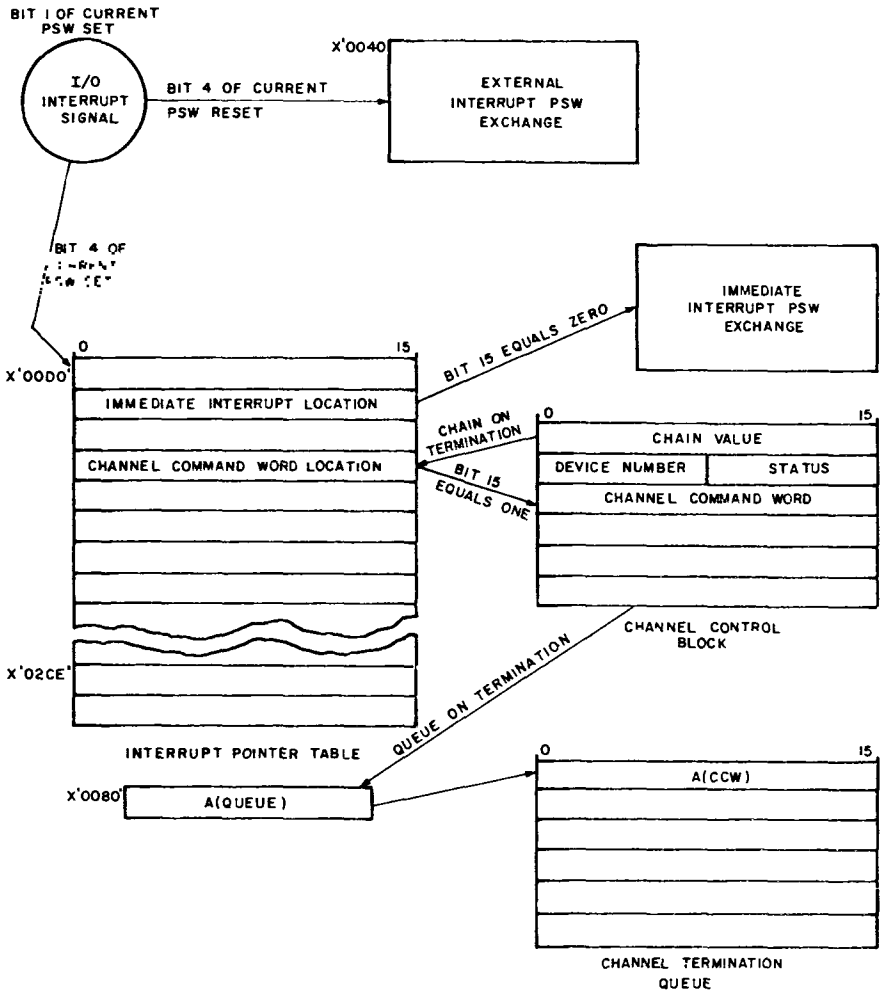


Figure 10. I/O Channel Operation Block Diagram

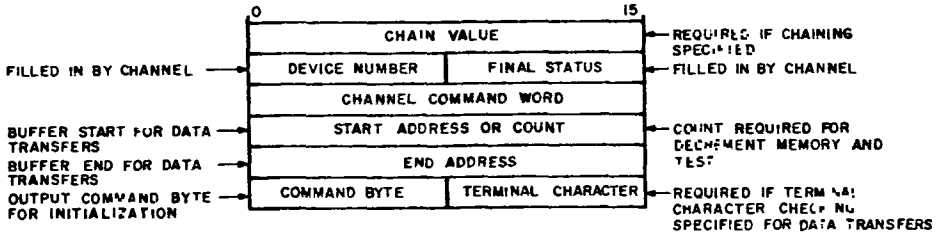


Figure 11. Channel Control Block

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 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 detects the presence of an interrupt signal from a peripheral device, 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.

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 Procsesor. 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 12.

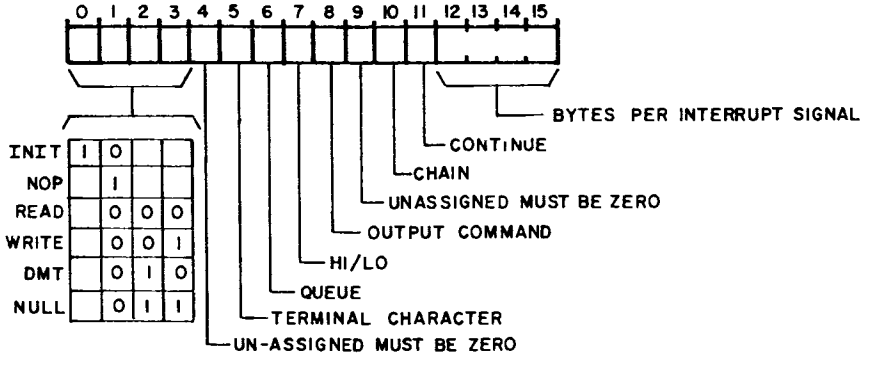


Figure 12. Bit Configuration For Channel Command Word

Initialization

Bits 0 (INIT) 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 13.

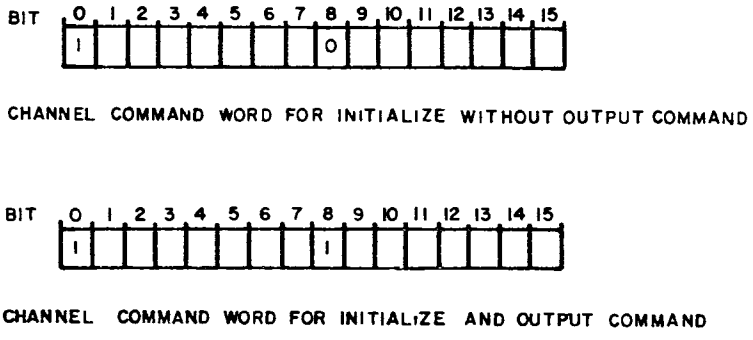


Figure 13. Channel Command for Initialize and Output Commands

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 14.

For all Read/Write operations, Bits 12 through 15 must contain the number of bytes to be transferred on each interrupt signal. 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 controls 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.

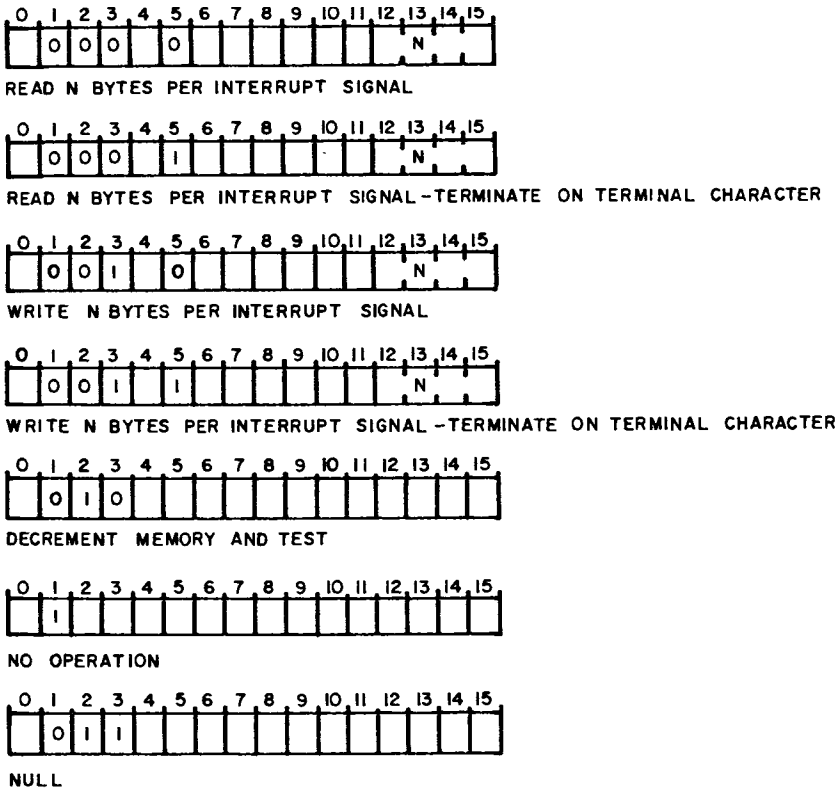


Figure 14. Channel Command Words for I/O Operation

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 operator". Subsequent interrupt signals from the device causes 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 is 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 15.

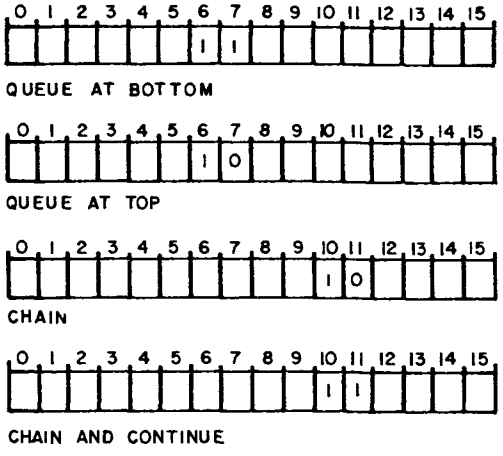


Figure 15. Channel Command Words for Termination

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 9

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 he programmed as an I/O device hy 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 16, is a RETMA standard $5\frac{1}{4}$" 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.

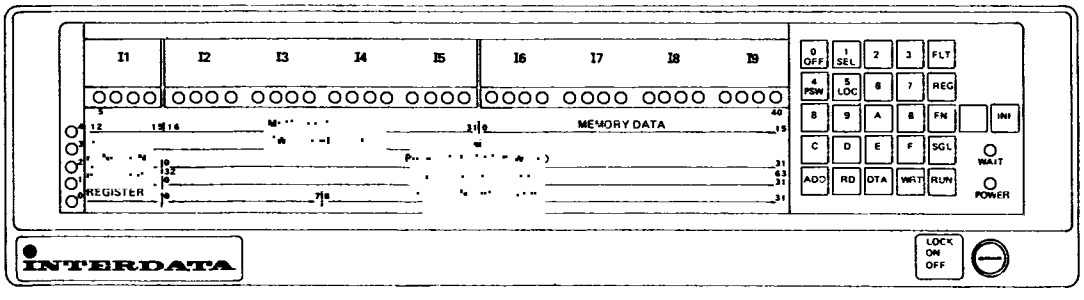


Figure 16. Hexadecimal Display Panel

Internal to the Hexadecimal Display Panel are five eight-hit hyte Display Registers, D1 through D5, that hold data output from the Processor, and a 20-hit Switch Register that holds data input from The Hexadecimal Keyboard. Refer to Figure 17.

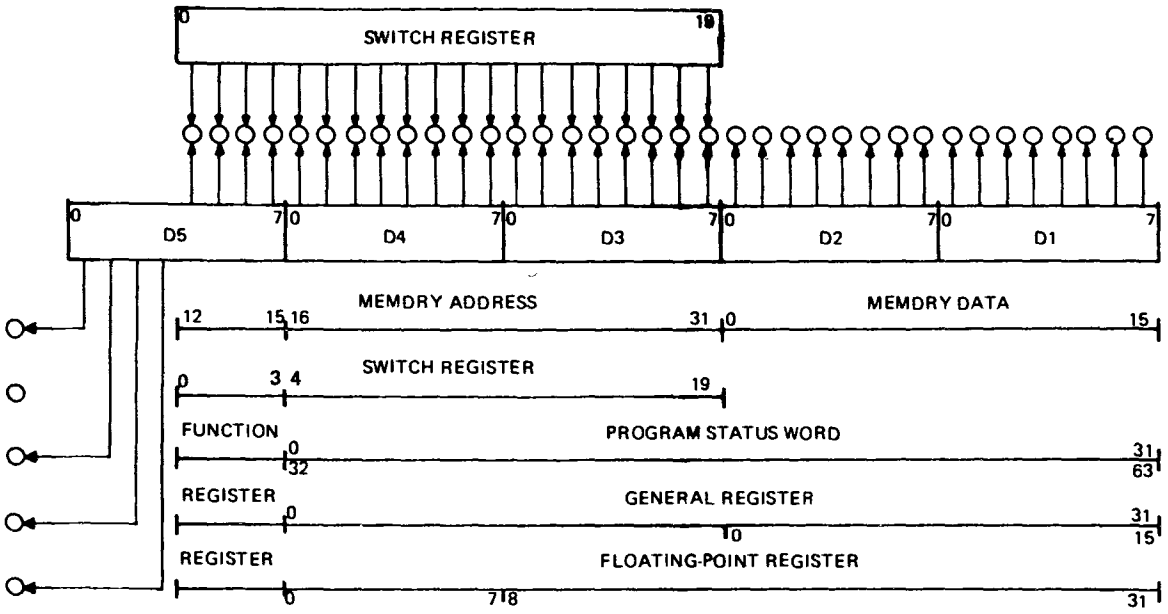


Figure 17. Display Registers and Indicators

Associated with each of Display Registers D1 through D4 are eight indicator lamps that provide a hinary read-out and two optional hexadecimal read-out indicators. Associated with the least significant four hits of Display Register D5 are four indicator lamps for hinary display and one optional hexadecimal read-out indicator.

The most significant four hits 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 hy 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 17, the most significant 20-bits of the display show the contents of Display Registers D3 and D4 and the least significant four hits of Display Register D5 (Bits 4:7); or the contents of the 20-hit Switch Register. When the Switch Register is being displayed, the lamp next to the Switch Register diagram is illuminated. 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-hits of the display again show the contents of Display Registers D3 and D4 and the least significant four hits of Display Register D5 (Bits 4:7).

The methods of displaying the Switch Register and the other diagrammed items are discussed later.

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. Only INT switch is active.

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 illuminates. 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 illuminated and a Hexadecimal 5 in 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. (If the Memory Access Controller is enabled then the relocated value of the Location Counter is the effective address of the memory location.) The halfword data read is output to Display Registers D1 and D2. The Location Counter is incremented hy two and output to Display Registers D3 and D4 and the least significant four bits of Display Register D5 (a 20-bit value). The lamp next to the Memory Address/Memory Data diagram is illuminated.

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 hy the Location Counter. (If the Memory Access Controller is enabled then the relocated value of the Location Counter is the effective address of the memory location.) 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 and the least significant four bits of Display Register D5. The lamp next to the Memory Address/Memory Data diagram is illuminated.
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 illuminated and the number of the displayed register is output to the top four display lamps.
EXAMINE 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 numher 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 illuminated 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 douhle 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).

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 mode assumed when power is re-applied (RUN or SGL) depends upon the presence or absence of the Automatic Restart option. See Power Fail.

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. Select the memory read start address as in Address a Memory Location.
2. Depress the Read (RD) key. The address read from, plus two, and the data read from memory are displayed.
3. Repeat from Step 2 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. Select the memory write start address as in Address a Memory Location.
2. Depress the Data (DTA) key. The Switch Register is cleared and displayed.

3. Enter the data to be written from the Hexadecimal Keyboard.

4. Depress the Write (WRT) key. The address written into, plus two, and the data written are displayed.

5. Repeat from Step 2 to write different data into successive locations or from Step 4 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.

NOTE

The General Register displayed is from the register set specified hy the current Program Status Word.

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.

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 least significant 16 bits (Bits 16-31) of the Status field:

1. Depress the Data (DATA) key.

2. Enter the data (to be written into bits 16-31 of the PSW) from the Hexidecimal keyboard.

3. Depress the Function (FN) key.

4. Depress Hexadecimal key 1. The Processor halts and copies the 16 bits of the Switch register in bits 16-31 of the PSW. The modified PSW is then 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 hy 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 hy the micro-program.

If the Hexadecimal Display Panel was in the Run mode, and the Automatic Restart option is present and if the Machine Malfunction Interrupt Enable hit 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, or if the Automatic Restart option is not present, the value of the Location Counter is output to the display registers, the WAIT lamp on the console is illuminated and the Halt mode is entered.

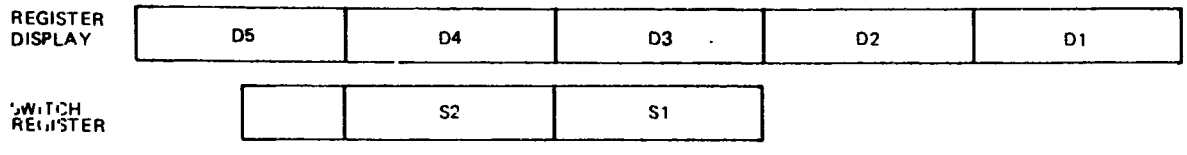
Power failure and operation of the Initialize key are indistinguishable to the Micro-Program. Consequently, operation of the Initialize key should he considered carefully when the Machine Malfunction Interrupt is enabled.

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.

DATA FORMAT

A byte or a halfword can he transferred to or from the Display using a WD, WH, WDR, WHR, or RD, RH, RDR, RHR instruction. Refer to Figure 18.



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 18. Hexadecimal Display Panel Data Transfers

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 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 18.

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'.

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 illuminates 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 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
```

TABLE 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	} Single or Halt
Floating Register	X	1	0	1	X	X	X	X	} Single or Halt
Function	X	1	0	0	X	X	X	X	} Single or Halt

		0	1	2	3	4	5	6	7	
General Register	0	0	X	X	X	1	0	0	0	Floating Register 0
	1	1	X	X	X	1	0	0	0	
	2	0	X	X	X	1	0	0	1	2
	3	1	X	X	X	1	0	0	1	
	4	0	X	X	X	1	0	1	0	4
	5	1	X	X	X	1	0	1	0	
	6	0	X	X	X	1	0	1	1	6
	7	1	X	X	X	1	0	1	1	
	8	0	X	X	X	1	1	0	0	8
	9	1	X	X	X	1	1	0	0	
	A	0	X	X	X	1	1	0	1	A
	B	1	X	X	X	1	1	0	1	
	C	0	X	X	X	1	1	1	0	C
	D	1	X	X	X	1	1	1	0	
	E	0	X	X	X	1	1	1	1	Floating Register E
General Register	F	1	X	X	X	1	1	1	1	

		0	1	2	3	4	5	6	7	
Function	0	0	X	X	X	0	0	0	0	Console Interrupt
	1	1	X	X	X	0	0	0	0	Modify PSW Bits
	2	0	X	X	X	0	0	0	1	
	3	1	X	X	X	0	0	0	1	
	4	0	X	X	X	0	0	1	0	PSW
	5	1	X	X	X	0	0	1	0	LOC
	6	0	X	X	X	0	0	1	1	
	7	1	X	X	X	0	0	1	1	
	8	0	X	X	X	0	1	0	0	
	9	1	X	X	X	0	1	0	0	
	A	0	X	X	X	0	1	0	1	
	B	1	X	X	X	0	1	0	1	
	C	0	X	X	X	0	1	1	0	
	D	1	X	X	X	0	1	1	0	
	E	0	X	X	X	0	1	1	1	
Function	F	1	X	X	X	0	1	1	1	

COMMAND

Normal	1	0	0	0	0	0	0	0
Incremental	0	1	0	0	0	0	0	0

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 sequence 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.

APPENDIX 1

INSTRUCTION SUMMARY - ALPHABETICAL

INSTRUCTION	OP-CODE	MNEMONIC	PAGE NO.
Acknowledge Interrupt	DF	AI	84
Acknowledge Interrupt Register	9F	AIR	84
Add Halfword	4A	AH	45
Add Halfword Immediate	CA	AHI	45
Add Halfword Register	0A	AHR	45
Add Halfword Memory	61	AHM	46
Add Immediate Short	26	AIS	45
Add to Bottom of List	65	ABL	35
Add to Top of List	64	ATL	35
Add with Carry Halfword	4E	ACH	46
Add with Carry Halfword Register	0E	ACHR	47
AND Halfword	44	NH	23
AND Halfword Immediate	C4	HNI	23
AND Halfword Register	04	NHR	23
Autoload	D5	AL	95
Branch on Index High	C0	BXH	42
Branch on Index Low or Equal	C1	BXLE	41
Branch and Link	41	BAL	40
Branch and Link Register	01	BALR	40
Branch on False Condition	43	BFC	38
Branch on False Condition Register	03	BFCR	38
Branch on True Condition	42	BTC	39
Branch on True Condition Register	02	BRCR	39
Branch on False Condition Backward Short	22	BFBS	38
Branch on False Condition Forward Short	23	BFFS	38
Branch on True Condition Backward Short	20	BTBS	39
Branch on True Condition Forward Short	21	BTFS	39
Compare Halfword	49	CH	50
Compare Halfword Immediate	C9	CHI	50
Compare Halfword Register	09	CHR	50
Compare Logical Byte	D4	CLB	28
Compare Logical Halfword	45	CLH	27
Compare Logical Halfword Immediate	C5	CLHI	27
Compare Logical Halfword Register	05	CLHR	27
Divide Halfword	4D	DH	53
Divide Halfword Register	0D	DHR	54
Exchange Byte Register	94	EXBR	19
Exchange Program Status Register	95	EPSR	77
Exclusive OR Halfword	47	XH	27
Exclusive OR Halfword Immediate	C7	XHI	27
Exclusive OR Halfword Register	07	XHR	27

INSTRUCTION	OP-CODE	MNEMONIC	PAGE NO.
Floating Point Add	6A	AE	64
Floating Point Add Register	2A	AER	64
Floating Point Compare	69	CE	66
Floating Point Compare Register	29	CER	66
Floating Point Divide	6D	DE	68
Floating Point Divide Register	2D	DER	68
Floating Point Load	68	LE	62
Floating Point Load Register	28	LER	62
Floating Point Multiply	6C	ME	67
Floating Point Multiply Register	2C	MER	67
Floating Point Store	60	STE	63
Floating Point Subtract	6B	SE	65
Floating Point Subtract Register	2B	SER	65
Load Byte	D3	LB	18
Load Byte Register	93	LBR	18
Load Complement Short	25	LCS	16
Load Halfword	48	LH	16
Load Halfword Immediate	C8	LHI	16
Load Halfword Register	08	LHR	16
Load Immediate Short	24	LIS	16
Load Multiple	D1	LM	17
Load Program Status Word	C2	LPSW	76
Multiply Halfword	4C	MH	51
Multiply Halfword Register	0C	MHR	51
Multiply Halfword Unsigned	DC	MHU	52
Multiply Halfword Unsigned Register	9C	MHUR	52
OR Halfword	46	OH	24
OR Halfword Immediate	C6	OHI	24
OR Halfword Register	06	OHR	24
Output Command	DE	OC	86
Output Command Register	9E	OCR	86
Read Block	D7	RB	89
Read Block Register	97	RBR	90
Read Data	DB	RD	87
Read Data Register	9B	RDR	87
Read Halfword	D9	RH	88
Read Halfword Register	99	RHR	88
Rotate Left Logical	EB	RLL	33
Rotate Right Logical	EA	RRL	34

INSTRUCTION	OP-CODE	MNEMONIC	PAGE NO.
Remove from Bottom of List	67	RBL	36
Remove from Top of List	66	RTL	36
Sense Status	DD	SS	85
Sense Status Register	9D	SSR	85
Shift Left Arithmetic	EF	SLA	54
Shift Left Halfword Arithmetic	CF	SLHA	55
Shift Left Halfword Logical	CD	SLHL	30
Shift Left Logical	ED	SLL	29
Shift Left Logical Short	91	SLLS	32
Shift Right Arithmetic	EE	SRA	56
Shift Right Halfword Arithmetic	CE	SRHA	57
Shift Right Halfword Logical	CC	SRHL	31
Shift Right Logical	EC	SRL	31
Shift Right Logical Short	90	SRLS	32
Simulate Interrupt	E2	SINT	78
Store Byte	D2	STB	22
Store Byte Register	92	STBR	22
Store Halfword	40	STH	20
Store Multiple	D0	STM	21
Subtract Halfword	4B	SH	48
Subtract Halfword Immediate	CB	SHI	48
Subtract Halfword Register	0B	SHR	48
Subtract Immediate Short	27	SIS	48
Subtract with Carry Halfword	4F	SCH	49
Subtract with Carry Halfword Register	0F	SCHR	49
Supervisor Call	E1	SVC	79
Test Halfword Immediate	C3	THI	26
Write Block	D6	WB	93
Write Block Register	96	WBR	94
Write Data	DA	WD	91
Write Data Register	9A	WDR	91
Write Halfword	D8	WH	92
Write Halfword Register	98	WHR	93

APPENDIX 2

INSTRUCTION SUMMARY - NUMERICAL

OP-CODE	MNEMONIC	INSTRUCTION	PAGE NO.
01	BALR	Branch and Link Register	40
02	BTCR	Branch on True Conditon Register	39
03	BFCR	Branch on False Condition Register	38
04	NHR	AND Halfword Register	23
05	CLHR	Compare Logical Halfword Register	27
06	OHR	OR Halfword Register	24
07	XHR	Exclusive OR Register	27
08	LHR	Load Halfword Register	16
09	CHR	Compare Halfword Register	50
0A	AHR	Add Halfword Register	45
0B	SHR	Subtract Halfword Register	48
0C	MHR	Multiply Halfword Register	51
0D	DHR	Divide Halfword Register	54
0E	ACHR	Add with Carry Halfword Register	47
0F	SCHR	Subtract with Carry Halfword Register	49
20	BRBS	Branch on True Condition Backward Short	39
21	BRFS	Branch on True Condition Forward Short	39
22	BFBS	Branch on False Condition Backward Short	38
23	BFFS	Branch on False Condition Forward Short	38
24	LIS	Load Immediate Short	16
25	LCS	Load Complement Short	16
26	AIS	Add Immediate Short	45
27	SIS	Subtract Immediate Short	48
28	LER	Floating Point Load Register	62
29	CER	Floating Point Compare Register	66
2A	AER	Floating Point Add Register	64
2B	SER	Floating Point Subtract Register	65
2C	MER	Floating Point Multiply Register	67
2D	DER	Floating Point Divide Register	68
40	SHT	Store Halfword	20
41	BLA	Branch and Link	40
42	BTC	Branch on True Condition	39
43	BFC	Branch on False Condition	38
44	NH	AND Halfword	23
45	CLH	Compare Logical Halfword	27
46	OH	OR Halfword	24
47	XH	Exclusive OR Halfword	27
48	LH	Load Halfword	16
49	CH	Compare Halfword	50
4A	AH	Add Halfword	45
4B	SH	Subtract Halfword	48
4C	MH	Multiply Halfword	51
4D	DH	Divide Halfword	53

OP-CODE	MNEMONIC	INSTRUCTION	PAGE NO.
4E	ACH	Add with Carry Halfword	46
4F	SCH	Subtract with Carry Halfword	49
60	STE	Floating Point Store	63
61	AHM	Add Halfword Memory	46
64	ATL	Add to Top of List	35
65	ABL	Add to Bottom of List	35
66	RTL	Remove from Top of List	36
67	RBL	Remove from Bottom of List	36
68	LE	Floating Point Load	62
69	CE	Floating Point Compare	66
70			
6A	AE	Floating Point Add	64
6B	SE	Floating Point Subtract	65
6C	ME	Floating Point Multiply	67
6D	DE	Floating Point Divide	68
90	SRLS	Shift Right Logical Short	32
91	SLLS	Shift Left Logical Short	32
92	STBR	Store Byte Register	22
93	LBR	Load Byte Register	18
94	EXBR	Exchange Byte Register	19
95	EPSR	Exchange Program Status Register	77
96	WBR	Write Block	94
97	RBR	Read Block Register	90
98	WHR	Write Halfword Register	93
99	RHR	Read Halfword Register	88
9A	WDR	Write Data Register	91
9B	RDR	Read Data Register	87
9C	MHUR	Multiply Halfword Unsigned Register	52
9D	SSR	Sense Status Register	85
9E	OCR	Output Command Register	86
9F	AIR	Acknowledge Interrupt Register	84
C0	BXH	Branch on Index High	42
C1	BXLE	Branch on Index Low or Equal	41
C2	LPSW	Load Program Status Word	76
C3	THI	Test Halfword Immediate	26
C4	NHI	AND Halfword Immediate	23
C5	CLHI	Compare Logical Halfword Immediate	27
C6	OHI	OR Halfword Immediate	24
C7	XHI	Exclusive OR Halfword Immediate	27
C8	LHI	Load Halfword Immediate	16

OP-CODE	MNEMONIC	INSTRUCTION	PAGE NO.
C9	CHI	Compare Halfword Immediate	50
CA	AHI	Add Halfword Immediate	45
CB	SHI	Subtract Halfword Immediate	48
CC	SRHL	Shift Right Halfword Logical	31
CD	SLHL	Shift Left Halfword Logical	30
CE	SRHA	Shift Right Halfword Arithmetic	57
CF	SLHA	Shift Left Halfword Arithmetic	55
D0	STM	Store Multiple	21
D1	LM	Load Multiple	17
D2	STM	Store Byte	22
D3	LB	Load Byte	18
D4	CLB	Compare Logical Byte	28
D5	AL	Auto Load	95
D6	WB	Write Block	93
D7	RB	Read Block	89
D8	WH	Write Halfword	92
D9	RH	Read Halfword	88
DA	WD	Write Data	91
DB	RD	Read Data	87
DC	MHU	Multiply Halfword Unsigned	52
DD	SS	Sense Status	85
DE	OC	Output Command	86
DF	AI	Acknowledge Interrupt	84
E1	SVC	Supervisor Call	79
E2	SINT	Simulate Interrupt	78
EA	RRL	Rotate Right Logical	34
EB	RRL	Rotate Left Logical	33
EC	SRL	Shift Right Logical	31
ED	SLL	Shift Left Logical	29
EE	SRA	Shift Right Arithmetic	56
EF	SLA	Shift Left Arithmetic	54

APPENDIX 3

EXTENDED BRANCH MNEMONICS

INSTRUCTION	OP-CODE (M1)	MNEMONIC	OPERANDS
Branch on Carry	428	BC	A(X2)
Branch on Carry RR	028	BCR	R2
Branch on No Carry	438	BNC	A(X2)
Branch on No Carry RR	038	BNCR	R2
Branch on Equal	433	BE	A(X2)
Branch on Equal RR	033	BER	R2
Branch on Not Equal	423	BNE	A(X2)
Branch on Not Equal RR	029	BNER	R2
Branch on Low	428	BL	A(X2)
Branch on Low RR	028	BLR	R2
Branch on Not Low	438	BNL	A(X2)
Branch on Not Low RR	038	BNLR	R2
Branch on Minus	421	BM	A(X2)
Branch on Minus RR	021	BMR	R2
Branch on Not Minus	431	BNM	A(X2)
Branch on Not Minus RR	031	BNMR	R2
Branch on Plus	422	BP	A(X2)
Branch on Plus RR	022	BPR	R2
Branch on Not Plus	432	BNP	A(X2)
Branch on Not Plus RR	032	BNPR	R2
Branch on Overflow	424	BO	A(X2)
Branch on Overflow RR	024	BOR	R2
Branch Unconditional	430	B	A(X2)
Branch on Unconditional RR	030	BR	R2
Branch on Zero	433	BZ	A(X2)
Branch on Zero RR	033	BZR	R2
Branch on Not Zero	423	BNZ	A(X2)
Branch on Not Zero RR	023	BNZR	R2
No Operation	420	NOP	
No Operation RR	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)

INSTRUCTION	OP-CODE (M1)	MNEMONIC	OPERANDS
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)
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 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 4
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

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 TO DECIMAL CONVERSION TABLE

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

HEXADECIMAL ADDITION TABLE

	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 TABLE

	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
π	3.14159 26535 89793	3.243F 6A89
π^{-1}	0.31830 98861 83790	0.517C C1B7
$\sqrt{\pi}$	1.77245 38509 05516	1.C5BF 891C
$\mathrm{Ln}\,\pi$	1.14472 98858 49400	1.250D 048F
e	2.71828 18284 59045	2.B7E1 5163
e^{-1}	0.36787 94411 71442	0.5E2D 58D9
$\sqrt{e}$	1.64872 12707 00128	1.A612 98E2
$\log_{10}e$	0.43429 44819 03252	0.6F2D EC55
$\log_{2}e$	1.44269 50408 88963	1.7154 7653
γ	0.57721 56649 01533	0.93C4 67E4
$\mathrm{Ln}\,\gamma$	-0.54953 93129 81645	-0.8CAE 9BC1
$\sqrt{2}$	1.41421 35623 73095	1.6A09 E668
Ln2	0.69314 71805 59945	0.B172 17F8
$\log_{10}2$	0.30102 99956 63981	0.4D10 4D42
$\sqrt{10}$	3.16227 76601 68379	3.298B 075C
Ln10	2.30258 50929 94046	2.4D76 3777

ALPHABETICAL INDEX