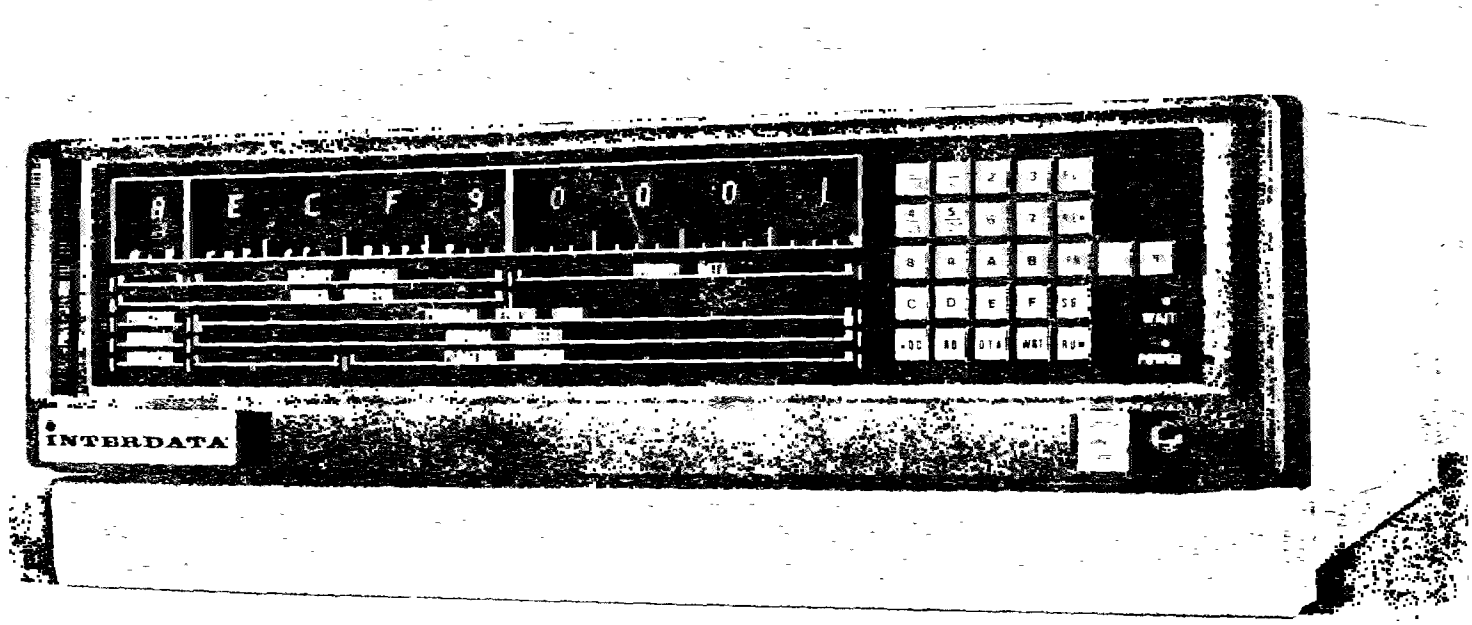
32 Bit Series
Reference Manual

INTERDATA®



32 Bit Series
Reference Manual

Publication Number 29-365R01

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

June 1974

TABLE OF CONTENTS

ILLUSTRATIONS

APPENDICIES

CHAPTER 1
SYSTEM DESCRIPTION

Aware of the growing need for minicomputers with improved performance and increased memory capacity, INTERDATA has combined the field proven and a reliable technologies used in the Models 70, 74, and 80 with its traditional concern for programming simplicity and economy to produce the extended series architecture. This new architecture is a logical step in the evolution of the INTERDATA family of minicomputers. Through the use of 32 bit general registers and revised instruction formats, it provides fullword data processing power and direct memory addressing up to a limit of 16 million bytes. Figure 1 shows the interrelationship of the various elements contained in this advanced system.

MEMORY SYSTEM

The basic system contains 32KB of memory. Memory may be expanded in either 16KB or 32KB increments up to the maximum allowed. Storage addressing is consecutive, starting at zero, for each eight bit byte. Memories may be set to maintain odd parity on the halfword level, one parity bit for each 16 data bits.

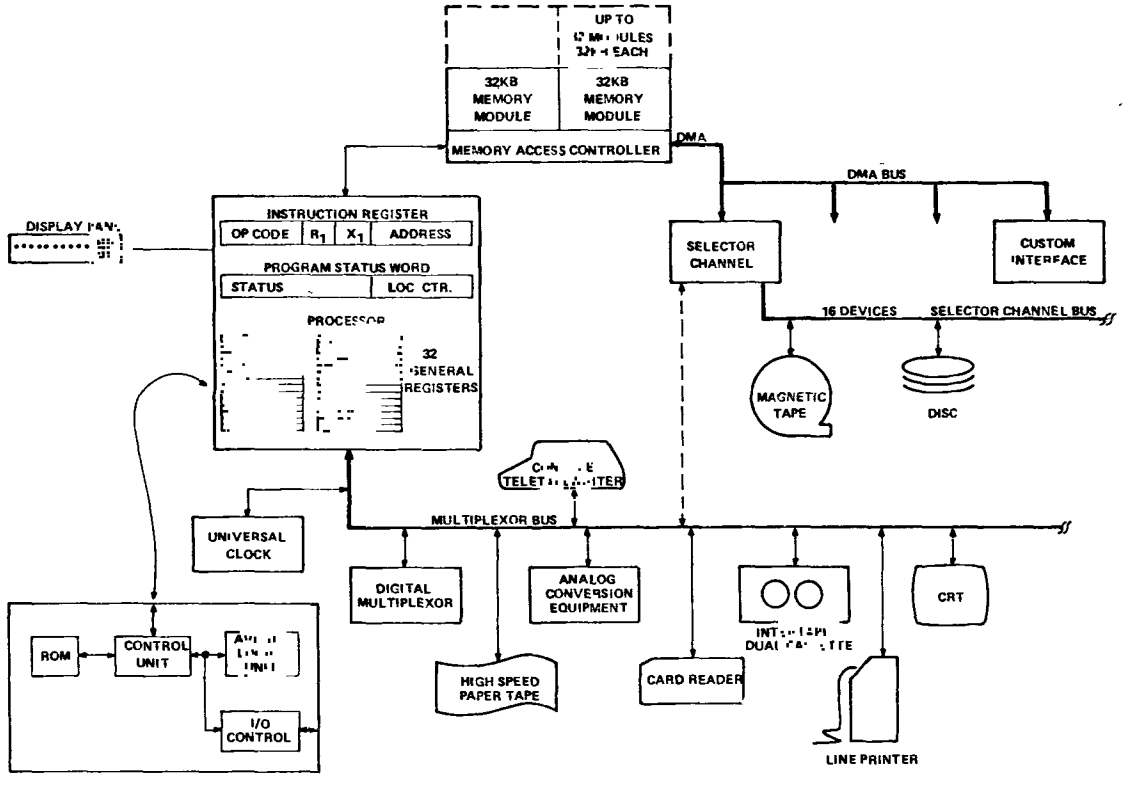


Figure 1. System Diagram

Direct Memory Access

Direct Memory Access devices may be added to the system. Direct Memory Access devices allow the operation of high speed peripherals (e.g., discs or drums), directly to or from memory. This enables the user to perform simultaneous processing together with the high speed data transfer. The DMA port uses a memory cycle stealing technique.

Selector Channel

The Selector Channel is a standard Direct Memory Access device that allows the connection of high speed peripheral devices directly to memory. To use the Selector Channel, the program initializes the device itself, sends a start and a final memory address to the Selector Channel, and commands the channel to start. The Processor, at this point, can proceed to another function. When the Selector Channel terminates the transfer, it generates a hardware interrupt to the Processor.

Relocation and Protection

The memory access and protect controller provides hardware relocation, segmentation, and protection of programs. Programs may be divided up into as many as 16 segments. Segments may be located anywhere in available memory. The memory access and protect controller provides direct address translation, that is, program addresses are automatically converted into real memory addresses.

The memory access and protect controller provides protection in two ways. First, it isolates the currently running program from all others in the system. The currently running program cannot reference, for any reason, memory locations outside its preassigned areas. Second, within its assigned areas, the program may be prevented from executing instructions or writing into memory.

MULTIPLEXOR INPUT/OUTPUT BUS

All medium and low speed devices connect to the Multiplexor Bus. This is a request/response bus, consisting of 30 lines: 16 bidirectional data lines, 8 control lines, 5 test lines, and 1 initialize line. Interrupt detection and automatic hardware vectoring for each of 1,023 devices are standard.

PERIPHERALS

A complete line of standard, off-the-shelf, peripheral devices is available with the system. All system modules and device controllers previously designed for other INTERDATA Processors are plug compatible with the Multiplexor Bus. These field-proven designs enable the user to select the devices or modules required for his specific application. The following are examples of what is available.

Digital Multiplexor

The digital multiplexor provides an economical set of modular blocks to monitor or control digital lines. A single controller, augmented with input and output modules of 128 lines each, provides the capability for monitoring 2,048 inputs and controlling 2,048 outputs. The digital multiplexor uses a biased core technique for input sampling. This technique insures absolute DC isolation from the sense contact, excellent common mode transient response and DC offset capability, which make the digital multiplexor particularly well suited for reliable use in noise contaminated environments.

Intertape Cassette System

The Intertape cassette system provides dual drive transports, capable of transferring data at a rate of 1,000 characters per second. This reliable and inexpensive unit makes an ideal substitute for paper tape equipment. With a storage capacity of 500,000 bytes per cassette, hardware read-after-write check, and longitudinal redundancy check, the Intertape cassette system is ideal for low speed auxiliary storage.

Industry Compatible Magnetic Tape Systems

Nine track, industry compatible magnetic tape systems are available in both 800 and 1,600 bits per inch (bpi) densities. These units operate at 45 inches per second (ips). The 800 bpi unit includes hardware read-after-write and cyclic redundancy checking hardware. The 1,600 bpi version includes read-after-write check and phase encoded formatter. Transfer rate for the 800 bpi version is 36,000 characters per second. Transfer rate for the 1,600 bpi version is 72,000 characters per second.

Removable Cartridge Disc System

The removable cartridge disc system is a reliable and inexpensive mass storage system, capable of providing 2.5 or 5.0 megabytes of storage per unit. Up to four disc drivers can operate on each controller, providing a maximum storage capacity of 20.0 megabytes per system. Average access time is 70 milliseconds and the transfer rate is 180,000 bytes per second.

Alphanumeric Display Terminals

Several types of alphanumeric display terminals are available. Display units provide a 1,920 character display (24 lines x 80 characters), standard 64 character ASCII subset, complete Processor and operator cursor control, and a full range of editing features with both message and character modes. Units operate at 110 to 9,600 baud.

Data Communications Equipment

A complete line of character buffered adapters is available to service Bell 103, 201, 202, and 301 data sets, as well as the 801 automatic dialer. This enables the Processor to accommodate applications requiring either synchronous or asynchronous communications.

High Speed Paper Tape System

The high speed paper tape system provides a 300 character per second reader and a 75 character per second punch. These units can be provided individually, or as a combined package using the same controller.

System Modules

A complete line of system modules provides the user with a simple and convenient means of creating special interfaces. These general purpose interface modules greatly reduce or eliminate special design effort. Standard modules are available to handle 8 bit or 16 bit parallel input or output, manual data entry, and decimal indicators.

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
Hardware multiply and divide
Floating point hardware

Program Status Word

The 64 bit Program Status Word (PSW), shown below, defines the state of the Processor at any given time.

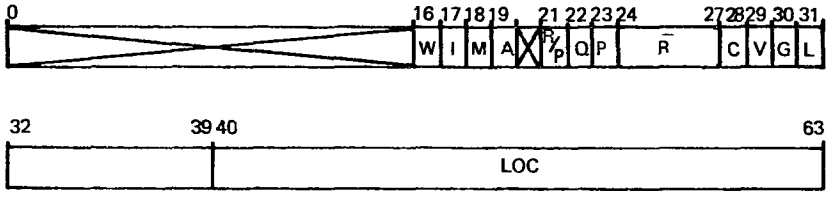


Figure 2. Program Status Word

Bits 0:31 are reserved for status information and interrupt masks. Bits 40:63 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 16	Wait state
Bit 17	Immediate interrupt mask
Bit 18	Machine malfunction interrupt mask
Bit 19	Arithmetic fault interrupt mask
Bit 21	Relocation/protection interrupt mask
Bit 22	System queue service interrupt mask
Bit 23	Protect mode
Bits 24:27	Register set select bits
Bits 28:31	Condition Code

Wait State

When this bit is set, the Processor halts normal program execution. It is still responsive to machine malfunction and immediate interrupts, if enabled.

Immediate Interrupt Mask

This bit 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 it is reset, the requests are queued. This bit also controls the Auto Driver Channel.

Machine Malfunction Interrupt Mask

This bit controls interrupts generated when power fails, when power returns, and when parity checking indicates a memory parity error.

Arithmatic Fault Interrupt Mask

This bit controls internal interrupts caused by arithmetic faults -- fixed-point quotient overflow, floating point overflow or underflow. If it is set, the interrupt is taken. If it is reset, the error condition is ignored.

Relocation Protection Interrupt Mask

This bit serves two purposes. It enables the memory access and protect controller so that program addresses are automatically relocated. It also enables the relocation/protection interrupt, which is generated by the memory access and protect controller.

System Queue Service Interrupt Mask

This bit controls the interrupt generated when the system queue requires service.

Protect Mode

This bit describes an operational state of the Processor. If it is set, the Processor is in the protect mode, and only non-privileged instructions may be executed. If this bit is reset, the Processor is in the Supervisor mode, and the currently running program may execute any legal instruction.

Register Set Select

Bits 24:27 of the Program Status Word are used to designate the current register set. All 32 bit series machines must have at least two register sets. Register sets are numbered 0 through 15. If fewer than 16 sets are implemented, the last set is always numbered 15.

Condition Code

Bits 28:31 of the Program Status Word contain the Condition Code. As part of the execution of certain instructions, the state of the Condition Code may be changed. The state of the Condition Code following these instructions indicates the nature of the result. Not all instructions affect the Condition Code. The state of the Condition Code may be tested with Conditional Branch instructions.

Location Counter

The Location Counter controls the sequencing of instruction execution. In normal sequential operation, the Location Counter contains the address of the next instruction to be executed. The instruction is fetched from memory. While the instruction is being executed, the Location Counter is incremented by either two, or four, or six, depending on the length of the instruction. Upon completion of instruction execution, the next instruction is fetched from the location specified by the incremented Location Counter, and the process is repeated.

This sequential mode of operation is altered by Branch instructions and by interrupts. Branch instructions cause the Location Counter to be replaced by a new value derived from the instruction. Interrupts cause the entire Program Status Word to be replaced by a new Program Status Word.

General Registers

In the current implementation, there are two sets of general registers. Each set contains 16 registers. Each register is 32 bits wide. The sets are numbered 0 and 15. Register set selection is determined by the state of Bits 24:27 of the current Program Status Word. Registers 1 through 15 of either set may be used as index registers.

When interrupts occur, the Processor loads pertinent information into preselected registers of register set 0, the supervisor set. The details of this operation are described in Chapter 6. Register set 15, the user set, does not have any specific functional assignments.

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 either in memory or in a pair of general registers belonging to register set 0. It loads information related to the interrupt condition in other registers of set 0. It loads a new Program Status Word from a memory location reserved for the specific interrupt condition. (The immediate interrupt is an exception to the rule. The status portion of the new Program Status Word, Bits 0:31, is forced to a preset value. The Location Counter is loaded from a memory location reserved for the interrupting device.) Refer to Chapter 6 for details on interrupt processing.

Reserved Memory Locations

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

Location			Use
X'000000'	-	X'00001F'	Reserved
X'000020'	-	X'000027'	Machine malfunction interrupt old PSW
X'000028'	-	X'00002F'	Not used, must be zero
X'000030'	-	X'000037'	Illegal instruction interrupt new PSW
X'000038'	-	X'00003F'	Machine malfunction interrupt new PSW
X'000040'	-	X'000047'	Not used, must be zero
X'000048'	-	X'00004F'	Arithmetic fault interrupt new PSW
X'000050'	-	X'00007F'	Bootstrap loader and device definition table
X'000080'	-	X'000083'	System queue pointer
X'000084'	-	X'000085'	Current PSW save pointer
X'000086'	-	X'000087'	Register save pointer
X'000088'	-	X'00008F'	System queue service interrupt new PSW
X'000090'	-	X'000097'	Relocation/protection interrupt new PSW
X'000098'	-	X'00009B'	Supervisor call new status
X'00009C'	-	X'0000BB'	Supervisor call interrupt new location counters
*X'0000BC'	-	X'0000CF'	Not used, must be zero
X'0000D0'	-	X'0002CF'	Interrupt service pointer table
X'0002D0'	-	X'0004CF'	Expanded interrupt service pointer table
X'0004D0'	-	X'0008CF'	Expanded interrupt service pointer table

*Used by Micro-Program

These reserved locations play an important role in both interrupt and input/output processing. For details on these subjects refer to Chapters 6 and 7. In addition to the above, certain locations are reserved for use by the Memory Access Controller. Refer to Chapter 8 for details.

Processor Operations

Fixed point arithmetic and logical operation are performed between:

The contents of two fullword registers.

The contents of a fullword register and the contents of a fullword located in memory.

The contents of a fullword 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.

In fixed point arithmetic and logical operations between a fullword register and a halfword operand in memory, the halfword operand is expanded to a fullword by propagating the most significant bit into the high order bits before the operation is started. This permits the use of halfword to fullword operations with consistent results, and it provides for space economy in that small values do not have to be contained in fullword locations.

Arithmetic operations on fixed point halfword quantities may produce results that are not entirely consistent with the results that would be obtained in a 16 bit Processor. Where this is a problem, the convert to halfword value instruction adjusts both the result and the Condition Code so that they are correct and consistent with the same operations in a 16 bit Processor.

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. Following floating point operations, the Condition Code is set to indicate the nature of the result.

DATA FORMATS

The Processor performs logical and arithmetic operations on single bits, 8 bit bytes, 16 bit halfwords, 32 bit fullwords, and 64 bit double words. This data may represent a fixed point number, a floating point number, or logical information.

Fixed Point Data

Fixed point arithmetic operands are either 16 bit halfwords or 32 bit fullwords. In multiply and divide operations, 64 bit operands are manipulated. Fixed point data is treated as 15 bit signed integers in the halfword format, and as 31 bit signed integers in the fullword format. Positive numbers are expressed 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. The numerical value of zero is represented with all bits zero. Refer to Chapter 4 for details on fixed point data representation.

Floating Point Data

A floating point number consists of a signed exponent and a signed fraction. The quantity expressed by this number 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 eight bits are used for the sign and exponent, and 24 bits are used for the fraction. Refer to Chapter 5 for details on floating point data representation.

Logical Data

Logical operations manipulate 8 bit bytes, 16 bit halfwords, and 32 bit fullwords. In addition, it is possible to perform logical operations on single bits located in bit arrays. Refer to Chapter 2 for details on logical data representation.

INSTRUCTION FORMATS

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

OP	Operation code
R1	First operand register
R2	Second operand register
N	A four bit immediate value
X2	Second operand single index register
D2	Second operand displacement
FX2	Second operand first index register
SX2	Second operand second index register
A2	Second operand direct address
I2	Second operand immediate value

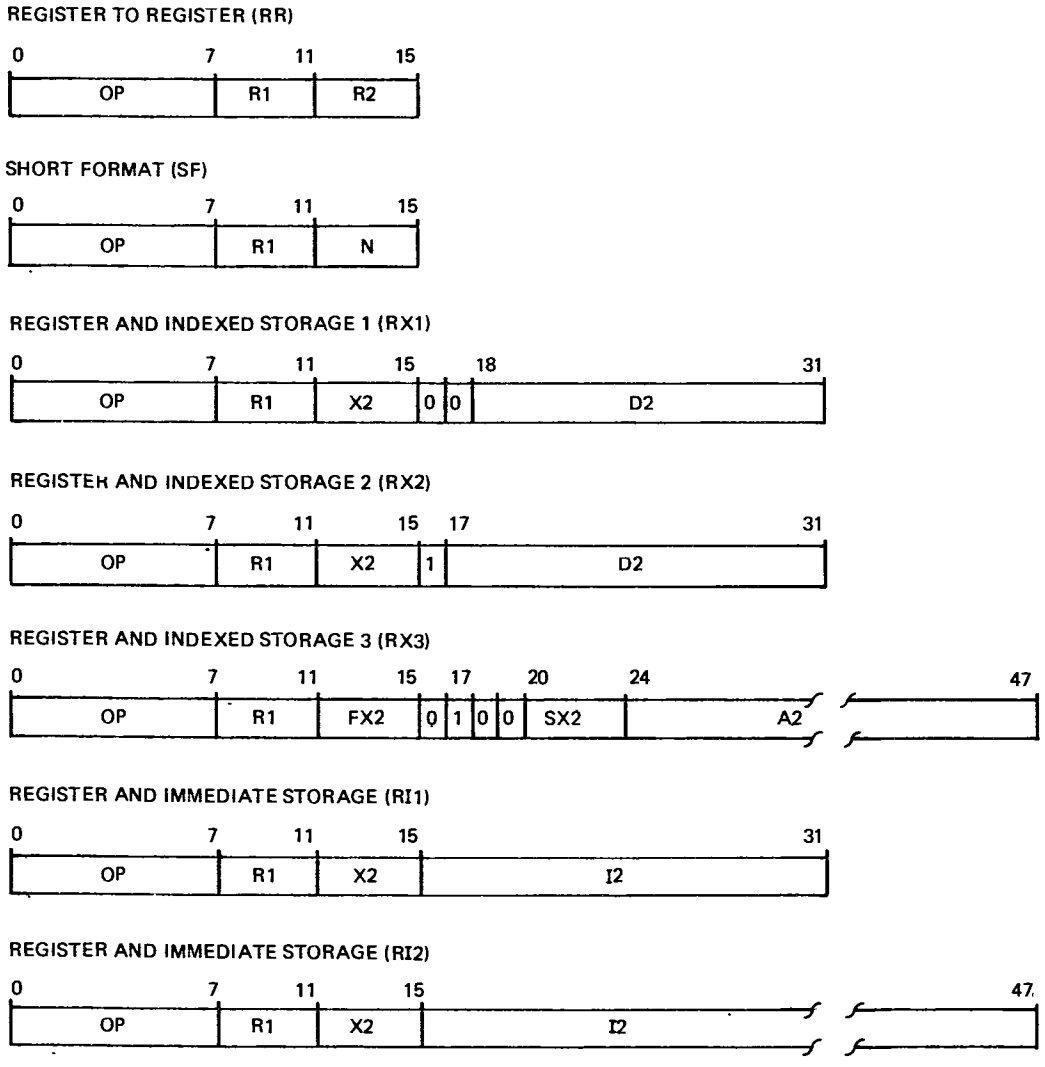


Figure 3. Instruction Formats

Most instructions in the extended series 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
AR	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 One (RX1) 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, Bits 16 and 17 must be zero, and Bits 18:31 contain the D2 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 14 bit displacement contained in the D2 field. The displacement is always positive. For example:

Assembler Notation		Machine Code
S	2, D2(3)	5B233400

instructs the Processor to subtract the fullword contents of the memory location, whose address is obtained by adding X'3400' to the contents of index Register 3, from the contents of Register 2. The result replaces the contents of Register 2.

Register and Indexed Storage Two (RX2) Format

This format provides relative addressing capability in a 32 bit instruction word. Bits 0:7 contain the operation code. Bits 8:11 contain the R1 specification. Bits 12:15 contain the X2 specification. Bit 16 must always be one. Bits 17:31 contain the relative displacement, D2.

In this format, the register specified by R1 contains the first operand. The second operand is located in memory at the address obtained by adding to the incremented Location Counter the sum of the contents of the index register specified by X2 and the contents of the D2 field. The D2 field may contain either a positive or a negative number. Negative numbers are expressed in two's complement notation. For example:

Assembler Notation	Machine Code
A 6,D2 (4)	5A648040

instructs the Processor to add to the contents of Register 6 the fullword quantity found in memory at the address obtained by adding to the incremented Location Counter the sum of X'0040' and the contents of the index register, Register 4. The result replaces the contents of Register 6.

Register and Indexed Storage Three (RX3) Format

This is a 48 bit format in which double indexing is permitted. Bits 0:7 contain the operation code. Bits 8:11 contain the R1 specification. Bits 12:15 contain the first index specification, FX2. Bit 16 must be zero. Bit 17 must be one. Bits 18:19 must be zero. Bits 20:23 contain the second index specification, SX2. Bits 24:47 contain a 24 bit address, A2.

In general, the first operand is contained in the register specified by R1. The second operand is located in memory. Its memory address is obtained by adding the contents of the first index register to the contents of the second index register, and then adding this result ot the contents of the A2 field. For example:

Assembler Notation	Machine Code
S 7,A2 (9,4)	5B7944000420

instructs the Processor to subtract from the contents of Register 7 the fullword quantity located in memory at the address obtained by adding the contents of the first index register, Register 9, to the contents of the second index register, Register 4, and then adding this result to the address quantity, X'000420'. The result of the operation replaces the contents of Register 7.

NOTE

Second level indexing is allowed only if first level indexing is also specified.

Register and Immediate Storage One (RI1) Format

This format represents a 32 bit instruction word. Bits 0:7 contain the operation code. Bits 8:11 contain the R1 specification. 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 index register specified by X2 to the value contained in the I2 field. Before adding the immediate value to the contents of the index register, the 16 bit immediate value is expanded to a 32 bit fullword quantity by propagating the most significant bit through the high order bits. For example:

Assembler Notation	Machine Code
AHI 4,I2 (2)	CA423444

instructs the Processor to add to the contents of Register 4 the quantity obtained by adding X'00003444' to the contents of Register 2. The result replaces the contents of Register 4.

Register and Immediate Storage Two (RI2) Format

This is a 48 bit instruction format. Bits 0:7 contain the operation code. Bits 8:11 contain the R1 specification. Bits 12:15 contain the X2 specification. Bits 16:47 contain the 32 bit immediate value, I2.

The first operand is contained in the register specified by R1. The second operand is obtained by adding the contents of the index register, specified by X2, to the 32 bit immediate value contained in the I2 field. For example:

Assembler Notation	Machine Code
AI 3,I2 (2)	FA3224647318

instructs the Processor to add to the contents of Register 3 the value obtained by adding Y'24647318' to the contents of Register 2. The result replaces the contents of Register 3.

Branch Instruction Formats

The Branch instructions use the RR, SF, and all variations on the RX formats. However, in the Conditional Branch instructions, the R1 field does not specify a register. Instead, it contains a mask value (labeled M1 in the instruction descriptions), which is tested with the Condition Code.

Programming Note

When working with the INTERDATA Common Assembler Language (CAL) assembler, it is not necessary to specify explicitly either RX1 or RX2 or RX3 format. The assembler chooses the most economical format and supplies the required bits in the machine code. When double indexing is required, the assembler always chooses RX3 format.

CHAPTER 2
LOGICAL OPERATIONS

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, list processing, translate, and cyclic redundancy checks.

DATA FORMATS

Logical data may be organized as bytes, halfwords, fullwords, or bit arrays of up to 2^{31} bits as shown in Figure 4.

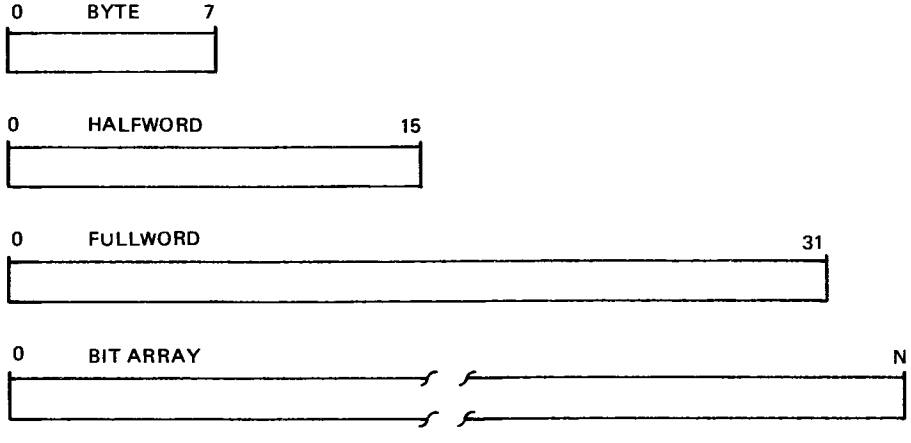


Figure 4. Logical Data

OPERATIONS

In logical operations between the contents of a general register and a halfword operand, the halfword operand is expanded to a fullword before starting the operation. The halfword is expanded by propagating the most significant bit through Bits 15:0 of the fullword.

Boolean Operations

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

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

Translation

The translate instruction is used to translate a character directly, or to effect an unconditional branch to a special translate subroutine. Associated with the translate instruction is a translation table. The entries in the table are halfwords as shown in Figure 5.

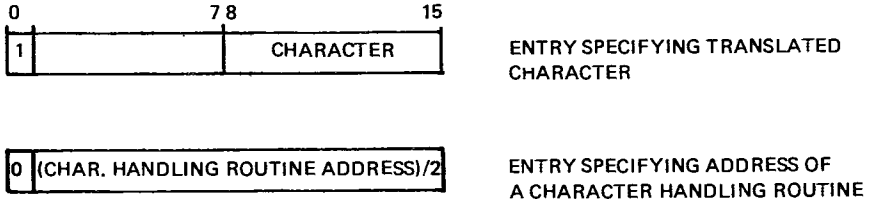


Figure 5. Translation Table Entry

The character to be translated is a byte of logical data. This unsigned quantity is doubled and used as an index into the table. If the corresponding entry has a one in bit Position zero, then Bits 8:15 contain the character to be substituted for the data character. If there is a zero in bit Position zero, then Bits 1:15 contain the address, divided by two, of the translate routine. When the translate instruction results in a branch, this value is doubled to produce the address of the routine. Because this result is a 16 bit address, the software routine must be located in the first 64KB of the program. (The program may reside anywhere in memory if it is relocated by the relocation and protection module.) The translate table may contain up to 256 entries. However, if the data characters are always less than eight bits, fewer entries are required.

The list processing instructions manipulate a circular list defined as follows:

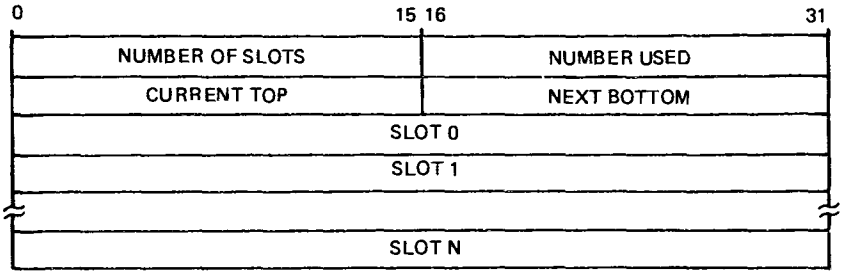


Figure 6. Circular List Definition

The first two fullwords contain the list parameters. Immediately following the parameter block is the list itself. The first fullword 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 65,535 fullword slots may be specified. (Maximum slot designation is equal to X'FFFE'.)

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

The third and fourth halfwords 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 7.

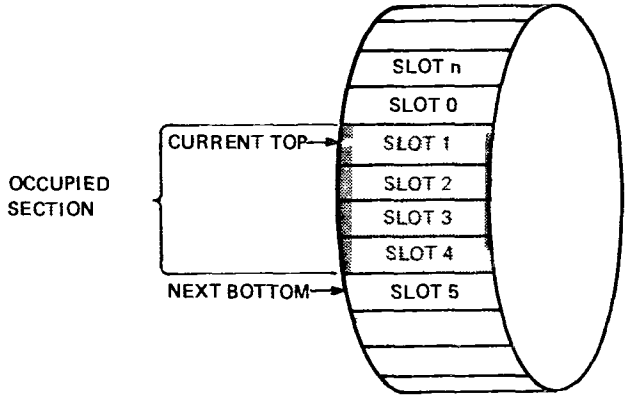


Figure 7. Circular List

LOGICAL INSTRUCTION FORMATS

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

LOGICAL INSTRUCTIONS

The instructions described in this section are:

L	Load
LR	Load Register
LI	Load Immediate
LIS	Load Immediate Short
LCS	Load Complement Short
LH	Load Halfword
LHI	Load Halfword Immediate
LA	Load Address
LHL	Load Halfword Logical
LM	Load Multiple
LB	Load Byte
LBR	Load Byte Register
EXHR	Exchange Halfword Register
EXBR	Exchange Byte Register
ST	Store
STH	Store Halfword
STM	Store Multiple
STB	Store Byte
STBR	Store Byte Register
CL	Compare Logical
CLR	Compare Logical Register
CLI	Compare Logical Immediate
CLH	Compare Logical Halfword
CLHI	Compare Logical Halfword Immediate
CLB	Compare Logical Byte
N	AND
NR	AND Register
NI	AND Immediate
NH	AND Halfword
NHI	AND Halfword Immediate
O	OR
OR	OR Register
OI	OR Immediate
OH	OR Halfword
OHI	OR Halfword Immediate
X	Exclusive OR
XR	Exclusive OR Register
XI	Exclusive OR Immediate
XH	Exclusive OR Halfword
XHI	Exclusive OR Halfword Immediate
TI	Test Immediate
THI	Test Halfword Immediate
SLL	Shift Left Logical
SLLS	Shift Left Logical Short
SRL	Shift Right Logical
SRLS	Shift Right Logical Short
SLHL	Shift Left Halfword Logical
SLHLS	Shift Left Halfword Logical Short
SRHL	Shift Right Halfword Logical
SRHLS	Shift Right Halfword Logical Short
RLL	Rotate Left Logical
RRL	Rotate Right Logical
TS	Test and Set

TBT	Test Bit
SBT	Set Bit
CBT	Complement Bit
RBT	Reset Bit
CRC12	Cyclic Redundancy Check Modulo 12
CRC16	Cyclic Redundancy Check Modulo 16
TLATE	Translate
ATL	Add to Top of List
ABL	Add to Bottom of List
RTL	Remove from Top of List
RBL	Remove from Bottom of List

Instructions

Load
Load Register
Load Immediate
Load Immediate Short
Load Complement Short

Assembler Notation		Op-Code	Format
L	R1,D2 (X2)	58	RX1, RX2
L	R1,A2 (FX2, SX2)	58	RX3
LR	R1,R2	08	RR
LI	R1,I2 (X2)	F8	RI2
LIS	R1,N	24	SF
LCS	R1,N	25	SF

Operation

The second operand replaces the contents of the register specified in 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 32 bit fullword with high order bits forced to zero. This fullword 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 32 bit fullword with high order bits forced to zero. The two's complement value of this fullword replaces the contents of the register specified by R1.

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

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

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

Instructions

Load Halfword
Load Halfword Immediate

Assembler Notation		Op-Code	Format
LH	R1,D2 (X2)	48	RX1, RX2
LH	R1,A2 (FX2,SX2)	48	RX3
LHI	R1,I2 (X2)	C8	RI1

Operation

The halfword second operand is expanded to a fullword by propagating the most significant bit through Bits 15:0. This fullword 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

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

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

Instruction

Load Address

Assembler Notation		Op-Code	Format
LA	R1,D2 (X2)	E6	RX1, RX2
LA	R1,A2 (FX2,SX2)	E6	RX3

Operation

The effective address of the second operand (24 bits) replaces Bits 8:31 of the register specified by R1. Bits 0:7 of the register specified by R1 are forced to zero.

Condition Code

Unchanged

Instruction

Load Halfword Logical

Assembler Notation		Op-Code	Format
LHL	R1, D2 (X2)	73	RX1, RX2
LHL	R1, A2 (FX2, SX2)	73	RX3

Operation

The halfword second operand replaces Bits 16:31 of the register specified by R1. Bits 0:15 of the register specified by R1 are forced to zero.

Condition Code

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

Programming Note

The second operand must be located on a halfword boundary.

Instruction

Load Multiple

Assembler Notation		Op-Code	Format
LM	R1,D2 (X2)	D1	RX1, RX2
LM	R1,A2 (FX2,SX2)	D1	RX3

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 fullword from memory. The process stops when Register 15 has been loaded.

Condition Code

Unchanged

Programming Note

The second operand must be located on a fullword boundary.

Instructions

Load Byte
Load Byte Register

Assembler Notation		Op-Code	Format
LB	R1,D2 (X2)	D3	RX1,RX2
LB	R1,A2 (FX2,SX2)	D3	RX3
LBR	R1,R2	93	RR

Operation

The eight bit second operand replaces the least significant bits (Bits 24:31) of the register specified by R1. Bits 0:23 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 24:31) of the register specified by R2.

Instruction

Exchange Halfword Register

Assembler Notation		Op-Code	Format
EXHR	R1,R2	34	RR

Operation

Bits 0:15 of the register specified by R2 replace Bits 16:31 of the register specified by R1. Bits 16:31 of the register specified by R2 replace Bits 0:15 of the register specified by R1.

Condition Code

Unchanged

Programming Note

If R1 equals R2, the two halfwords contained within the register are exchanged.

Instruction

Exchange Byte Register

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

Operation

The two eight bit bytes contained in Bits 16:31 of the register specified by R2 are exchanged and loaded into Bits 16:31 of the register specified by R1. Bits 0:15 of the register specified by R1 are unchanged. The register specified by R2 is unchanged.

Condition Code

Unchanged

Programming Note

R1 and R2 may specify the same register.

Instruction

Store

Assembler Notation		Op-Code	Format
ST	R1,D2 (X2)	50	RX1,RX2
ST	R1,A2 (FX2,SX2)	50	RX3

Operation

The 32 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 fullword boundary.

Instruction

Store Halfword

Assembler Notation		Op-Code	Format
STH	R1, D2 (X2)	40	RX1, RX2
STH	R1, A2 (FX2, SX2)	40	RX3

Operation

Bits 16:31 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, D2 (X2)	D0	RX1, RX2
STM	R1, A2 (FX2, SX2)	D0	RX3

Operation

The fullword 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 fullword boundary.

Store Byte
Store Byte Register

Assembler Notation		Op-Code	Format
STB	R1, D2 (X2)	D2	RX1, RX2
STB	R1, A2 (FX2, SX2)	D2	RX3
STBR	R1, R2	92	RR

Operation

The least significant eight bits (Bits 24:31) 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 24:31 of the register specified by R2. Bits 0:23 of the register are unchanged.

Instructions

Compare Logical
Compare Logical Register
Compare Logical Immediate

Assembler Notation		Op-Code	Format
CL	R1,D2 (X2)	55	RX1,RX2
CL	R1,A2 (FX2,SX2)	55	RX3
CLR	R1,R2	05	RR
CLI	R1,I2 (X2)	F5	RI2

Operation

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

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

Compare Logical Halfword
Compare Logical Halfword Immediate

Assembler Notation		Op-Code	Format
CLH	R1, D2 (X2)	45	RX1, RX2
CLH	R1, A2 (FX2, SX2)	45	RX3
CLHI	R1, I2 (X2)	C5	RI1

Operation

The halfword second operand is expanded to a fullword by propagating the most significant bit through Bits 15:0. The first operand, the contents of the register specified by R1, is compared to this fullword. 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
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 formats, the second operand must be located on a halfword boundary.

Instruction

Compare Logical Byte

Assembler Notation		Op-Code	Format
CLB	R1,D2 (X2)	D4	RX1, RX2
CLB	R1,A2 (FX2,SX2)	D4	RX3

Operation

The byte quantity, contained in Bits 24:31 of the register specified by R1, is compared with the second operand. 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
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

AND
AND Register
AND Immediate

Assembler Notation		Op-Code	Format
N	R1, D2 (X2)	54	RX1, RX2
N	R1, A2 (FX2, SX2)	54	RX3
NR	R1, R2	04	RR
NI	R1, I2 (X2)	F4	RI2

Operation

The logical product of the 32 bit second operand and the contents of the register specified by R1 replaces the contents of the register specified by R1. The 32 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 formats, the second operand must be located on a fullword boundary.

AND Halfword
AND Halfword Immediate

Assembler Notation		Op-Code	Format
NH	R1,D2 (X2)	44	RX1,RX2
NH	R1,A2 (FX2,SX2)	44	RX3
NHI	R1,I2 (X2)	C4	RI1

Operation

The halfword second operand is expanded to a fullword by propagating the most significant bit through Bits 15:0. The logical product of these 32 bit quantity and the contents of the register specified by R1 replaces the contents of the register specified by R1. The 32 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 formats, the second operand must be located on a halfword boundary.

Instructions

OR
OR Register
OR Immediate

Assembler Notation		Op-Code	Format
O	R1,D2 (X2)	56	RX1,RX2
O	R1,A2 (FX2,SX2)	56	RX3
OR	R1,R2	06	RR
OI	R1,I2 (X2)	F6	RI2

Operation

The logical sum of the 32 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 formats, the second operand must be located on a fullword boundary.

Instructions

OR Halfword
OR Halfword Immediate

Assembler Notation		Op-Code	Format
OH	R1, D2 (X2)	46	RX1, RX2
OH	R1, A2 (FX2, SX2)	46	RX3
OHI	R1, I2 (X2)	C6	RI1

Operation

The halfword second operand is expanded to a fullword by propagating the most significant bit through Bits 15:0. The logical sum of this 32 bit quantity and the contents of the register specified by R1 replaces the contents of the register specified by R1. The 32 bit sum is form 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 formats, the second operand must be located on a halfword boundary.

Instructions

Exclusive OR
Exclusive OR Register
Exclusive OR Immediate

Assembler Notation		Op-Code	Format
X	R1,D2 (X2)	57	RX1,RX2
X	R1,A2 (FX2,SX2)	57	RX3
XR	R1,R2	07	RR
XI	R1,I2 (X2)	F7	RI2

Operation

The logical difference of the 32 bit second operand and the contents of the register specified by R1 replaces the contents of the register specified by R1. The 32 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 formats, the second operand must be located on a fullword boundary.

Instructions

Exclusive OR Halfword
Exclusive OR Halfword Immediate

Assembler Notation		Op-Code	Format
XH	R1, D2 (X2)	47	RX1, RX2
XH	R1, A2 (FX2, SX2)	47	RX3
XHI	R1, I2 (X2)	C7	RI1

Operation

The halfword second operand is expanded to a fullword by propagating the most significant bit through Bits 15:0. The logical difference of this 32 bit quantity and the contents of the register specified by R1 replaces the contents of the register specified by R1. The 32 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 formats, the second operand must be located on a halfword boundary.

Instruction

Test Immediate

Assembler Notation		Op-Code	Format
TI	R1,I2 (X2)	F3	RI2

Operation

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

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

Test Halfword Immediate

Assembler Notation		Op-Code	Format
THI	R1,I2 (X2)	C3	RI1

Operation

The halfword second operand is expanded to a fullword by propagating the most significant bit through Bits 15:0. Each bit in this quantity is logically ANDed with the corresponding bit contained in the register specified by R1. Neither operand is changed.

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

Instructions

Shift Left Logical
Shift Left Logical Short

Assembler Notation		Op-Code	Format
SLL	R1,I2 (X2)	ED	RI1
SLLS	R1,N	11	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 shifted into Position 31.

Condition Code

C	V	G	L	
X	0	0	0	Result is zero
X	0	0	0	Result is not zero
X	0	1	0	Result is not zero

Programming Notes

In the RI formats, the shift count is specified by the least significant five 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 out of Position 0.

When the register specified by R1 contains fixed point data, the L flat 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.

Instructions

Shift Right Logical
Shift Right Logical Short

Assembler Notation		Op-Code	Format
SRL	R1,R2 (X2)	EC	RI1
SRLS	R1, N	10	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 Position 31 are shifted through the carry flag on 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 five 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 out of Position 31.

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.

Shift Left Halfword Logical
Shift Left Halfword Logical Short

Assembler Notation		Op-Code	Format
SLHL	R1,I2 (X2)	CD	RI1
SLHLS	R1, N	91	SF

Operation

Bits 16:31 of the register specified by R1 are shifted left the number of places specified by the second operand. Bits shifted out of Position 16 are shifted through the carry flag and lost. The last bit shifted remains in the carry flag. Zeros are shifted into Position 31. Bits 0:15 of the first operand remain unchanged.

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 out of Position 16.

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.

Instructions

Shift Right Halfword Logical
Shift Right Halfword Logical Short

Assembler Notation		Op-Code	Format
SRHL	R1,I2 (X2)	CC	RI1
SRHLS	R1, N	90	SF

Operation

Bits 16:31 of the register specified by R1 are shifted right the number of places specified by the second operand. Bits shifted out of Position 31 are shifted through the carry flag and lost. The last bit shifted remains in the carry flag. Zeros are shifted into Position 16. Bits 0:15 of the first operand remain unchanged.

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 out of Position 31.

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	RI1

Operation

The 32 bit first operand, contained in the register specified by R1, is shifted left, end around, the number of positions specified by the second operand. Bits shifted out of Position 0 are shifted into Position 31.

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

Progrmming Notes

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

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.

Instruction

Rotate Right Logical

Assembler Notation		Op-Code	Format
RRL	R1, I2 (X2)	EA	RI1

Operation

The 32 bit first operand, contained in the register specified by R1, is shifted right, end around, the number of positions specified by the second operand. Bits shifted out of Position 31 are shifted into Position 0.

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 Notes

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

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.

Instruction

Test and Set

Assembler Notation		Op-Code	Format
TS	D2 (X2)	E0	RX1, RX2
TS	A2 (FX2, SX2)	E0	RX3

Operation

The halfword second operand is read from memory and, on the same cycle, written back with the most significant bit set. The most significant bit of the second operand is tested. The Condition Code reflects the state of this bit at the time of the memory read.

Condition Code

C	V	G	L	
X	X	X	0	Most significant bit reset
X	X	X	1	Most significant bit set

Programming Notes

The Test and Set instruction provides for the synchronization of Processors in multiprocessor systems. While one Processor is reading and modifying the second operand, no other Processor (or device), is allowed to access the second operand location.

The second operand must be located on a halfword boundary.

Instruction

Test Bit

Assembler Notation		Op-Code	Format
TBT	R1,D2 (X2)	74	RX1,RX2
TBT	R1,A2 (FX2,SX2)	74	RX3

Operation

The second operand address points to a bit array starting on a halfword boundary. The value contained in the register specified by R1 is the bit displacement into the array. The bit is located and tested. The test does not change the bit.

Condition Code

C	V	G	L	
0	0	0	0	Tested bit is zero
0	0	1	0	Tested bit is one

Instruction

Set Bit

Assembler Notation		Op-Code	Format
SBT	R1, D2 (X2)	75	RX1, RX2
SBT	R1, A2 (FX2, SX2)	75	RX3

Operation

The second operand address points to a bit array starting on a halfword boundary. The value contained in the register specified by R1 is the bit displacement into the array. The bit is located and set to one.

Condition Code

C	V	G	L	
0	0	0	0	Previous state of bit was zero
0	0	1	0	Previous state of bit was one

Instruction

Complement Bit

Assembler Notation		Op-Code	Format
CBT	R1,D2 (X2)	77	RX1,RX2
CBT	R1,A2 (FX2,SX2)	77	RX3

Operation

The second operand address points to a bit array starting on a halfword boundary. The value contained in the register specified by R1 is the bit displacement into the array. The bit is located and complemented.

Condition Code

C	V	G	L	
0	0	0	0	Previous state of bit was zero
0	0	1	0	Previous state of bit was one

Instruction

Reset Bit

Assembler Notation		Op-Code	Format
RBT	R1,D2 (X2)	76	RX1,RX2
RBT	R1,A2 (FX2,SX2)	76	RX3

Operation

The second operand address points to a bit array starting on a halfword boundary. The value contained in the register specified by R1 is the bit displacement into the array. The bit is located and forced to zero.

Condition Code

C	V	G	L	
0	0	0	0	Previous state of bit was zero
0	0	1	0	Previous state of bit was one

Cyclic Redundancy Check Modulo 12
Cyclic Redundancy Check Modulo 16

Assembler Notation		Op-Code	Format
CRC12	R1,D2 (X2)	5E	RX1,RX2
CRC12	R1,A2 (FX2,SX2)	5E	RX3
CRC16	R1,D2 (X2)	5F	RX1,RX2
CRC16	R1,A2 (FX2,SX2)	5F	RX3

Operation

These instructions are used to generate either a 12 bit or a 16 bit Cyclic Redundancy Check (CRC) character. The register specified by R1 contains, in Bits 24:31, the next data character to be included in the CRC. The second operand is the accumulated CRC. The polynomial used for the 12 bit CRC generation is:

$$X^{12}+X^{11}+X^{3}+X^{2}+X+1$$

The polynomial used for the 16 bit CRC generation is:

$$X^{16}+X^{15}+X^{2}+1$$

Condition Code

Unchanged

Programming Note

The second operand must be located on a halfword boundary.

Instruction

Translate

Assembler Notation		Op-Code	Format
TLATE	R1,D2 (X2)	E7	RX1,RX2
TLATE	R1,A2 (FX2,SX2)	E7	RX3

Operation

The least significant bits (Bits 24:31) of the register specified by R1 contain the character to be translated. The fullword location specified by the second operand address contains the address of a translation table. The table is made up of 256 halfwords. The character contained in the register specified by R1 is used as an index into the table.

If Bit 0 of the table entry corresponding to the index character is one, then Bits 8:15 of the table entry replace the index character, and the next sequential instruction is executed.

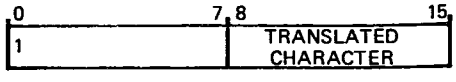
If Bit 0 of the table entry is zero, then Bits 1:15 of the table entry contain the address, divided by two, of a special handling routine. In this case, no translation takes place. The address contained in Bits 1:15 is shifted left by one, (multiplied by two). This address replaces the current Location Counter, thereby effecting an unconditional branch.

Condition Code

Unchanged

Programming Note

The first instruction of the handling routine must be located in the low 64KB of address space.



Instructions

Add to Top of List
Add to Bottom of List

Assembler Notation		Op-Code	Format
ATL	R1,D2 (X2)	64	RX1,RX2
ATL	R1,A2 (FX2,SX2)	64	RX3
ABL	R1,D2 (X2)	65	RX1,RX2
ABL	R1,A2 (FX2,SX2)	65	RX3

Operation

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

Condition Code

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

Programming Notes

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

The second operand location must be on a fullword boundary.

Remove from Top of List
Remove from Bottom of List

Assembler Notation		Op-Code	Format
RTL	R1,D2 (X2)	66	RX1,RX2
RTL	R1,A2 (FX2,SX2)	66	RX3
RBL	R1,D2 (X2)	67	RX1,RX2
RBL	R1,A2 (FX2,SX2)	67	RX3

Operation

The 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 and the instruction terminates with the overflow flag set in the Condition Code. This condition is referred to as list underflow. If underflow does not occur, the number of slots used tally is decremented by one, the appropriate pointer is changed, and the element is extracted and placed in the register specified by R1.

Condition Code

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

Programming Notes

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

The second operand location must be on a fullword boundary.

CHAPTER 3
BRANCHING

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

OPERATIONS

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

Decision Making

The Conditional Branch instructions permit the program to make the 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 instruction immediately following the Branch instruction is saved in the register specified by R1.

BRANCH INSTRUCTION FORMATS

The Branch instructions use the Register to Register (RR), the Short Form (SF), and the Register and Indexed Storage (RX) 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
BTC	Branch on True Condition
BTCR	Branch on True Condition Register
BTBS	Branch on True Condition Backward Short
BTFS	Branch on True Condition Forward Short
BAL	Branch and Link
BALR	Branch and Link Register
BXLE	Branch on Index Low or Equal
BXH	Branch on Index High

Branch on 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,D2 (X2)	43	RX1,RX2
BFC	M1,A2 (FX2,SX2)	43	RX3
BFCR	M1,R2	03	RR
BFBS	M1,N	22	SF
BFFS	M1,N	23	SF

Operation

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

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.

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,D2 (X2)	42	RX1,RX2
BTC	M1,A2 (FX2,SX2)	42	RX3
BTCR	M1,R2	02	RR
BTBS	M1,N	20	SF
BTFS	M1,N	21	SF

Operation

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

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 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,D2 (X2)	41	RX1,RX2
BAL	R1,A2 (FX2,SX2)	41	RX3
BALR	R1,R2	01	RR

Operation

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

Condition Code

Unchanged

Programming Notes

The second operand location must be on a halfword boundary.

The branch address is obtained before the register specified by R1 is changed. R1 may specify the same register as X2, FX2, SX2, or R2,

Instruction

Branch on Index Low or Equal

Assembler Notation		Op-Code	Format
BXLE	R1,D2 (X2)	C1	RX1,RX2
BXLE	R1,A2 (FX2,SX2)	C1	RX3

Operation

Prior to execution of this instruction, the register specified by R1 must contain a 32 bit starting index value. The register specified by R1+1 must contain a 32 bit increment value. The register specified by R1+2 must contain a 32 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 incremented index value replaces the contents of the register specified by R1.

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

The second operand location must be on a halfword boundary.

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

Instruction

Branch on Index High

Assembler Notation		Op-Code	Format
BXH	R1,D2 (X2)	C0	RX1,RX2
BXH	R1,A2 (FX2,SX2)	C0	RX3

Operation

Prior to execution of this instruction, the register specified by R1 must contain a 32 bit starting index value. The register specified by R1+1 must contain a 32 bit increment value. The register specified by R1+2 must contain a 32 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 incremented index value replaces the contents of the register specified by R1.

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

The second operand location must be on a halfword boundary.

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

CONDITION CODE

Most Fixed Point Arithmetic instructions affect the Condition Code. (The exceptions are the Multiply and Divide.) The Condition Code indicates the effect of the operation on the 32 bit 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 (RI) instruction formats.

FIXED POINT INSTRUCTIONS

The fixed point instructions described in this section are:

A	Add
AR	Add Register
AI	Add Immediate
AIS	Add Immediate Short
AH	Add Halfword
AHI	Add Halfword Immediate
AM	Add to Memory
AHM	Add Halfword to Memory
S	Subtract
SR	Subtract Register
SI	Subtract Immediate
SIS	Subtract Immediate Short
SH	Subtract Halfword
SHI	Subtract Halfword Immediate
C	Compare
CR	Compare Register
CI	Compare Immediate
CH	Compare Halfword
CHI	Compare Halfword Immediate
M	Multiply
MR	Multiply Register
MH	Multiply Halfword
MHR	Multiply Halfword Register
D	Divide
DR	Divide Register
DH	Divide Halfword
DHR	Divide Halfword Register
SLA	Shift Left Arithmetic
SLHA	Shift Left Halfword Arithmetic
SRA	Shift Right Arithmetic
SRHA	Shift Right Halfword Arithmetic
CHVR	Convert to Halfword Value Register

Instructions

Add
Add Register
Add Immediate
Add Immediate Short

Assembler Notation		Op-Code	Format
A	R1, D2 (X2)	5A	RX1, RX2
A	R1, A2 (FX2, SX2)	5A	RX3
AR	R1, R2	0A	RR
AI	R1, I2 (X2)	FA	RI2
AIS	R1, N	26	SF

Operation

The second operand is added algebraically to the contents of the register specified by R1. The result of this 32 bit addition 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 then 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 bit data field, N, to a 32 bit fullword by forcing the high order bits to zero.

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

Instructions

Add Halfword
Add Halfword Immediate

Assembler Notation		Op-Code	Format
AH	R1, D2 (X2)	4A	RX1, RX2
AH	R1, A2 (FX2, SX2)	4A	RX3
AHI	R1, I2 (X2)	CA	RI1

Operation

The 16 bit second operand is expanded to a 32 bit fullword by propagating the most significant bit through Bits 15:0 of the fullword. The fullword operand is added to the fullword 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	Carry

Instruction

Add to Memory

Assembler Notation		Op-Code	Format
AM	R1,D2 (X2)	51	RX1,RX2
AM	R1,A2 (FX2,SX2)	51	RX3

Operation

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

Instruction

Add Halfword to Memory

Assembler Notation		Op-Code	Format
AHM	R1,D2 (X2)	61	RX1,RX2
AHM	R1,A2 (FX2,SX2)	61	RX3

Operation

The second operand is expanded to a fullword by propagating the most significant bit through Bits 15:0. This fullword is added algebraically to the contents of the register specified by R1. The 32 bit result is truncated to 16 bits by removing the most significant bits (Bits 0:15). The 16 bit result replaces the contents of the memory location specified by the effective address of the second operand.

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.

Instructions

Subtract
Subtract Register
Subtract Immediate
Subtract Immediate Short

Assembler Notation		Op-Code	Format
S	R1,D2 (X2)	5B	RX1,RX2
S	R1,A2 (FX2,SX2)	5B	RX3
SR	R1,R2	0B	RR
SI	R1,I2 (X2)	FB	RI2
SIS	R1,N	27	SF

Operation

The fullword 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 Note

The second operand for the Subtract Immediate Short instruction is obtained by expanding the four bit data field, N, to a 32 bit fullword by forcing the high order bits to zero.

Subtract Halfword
Subtract Halfword Immediate

Assembler Notation		Op-Code	Format
SH	R1,D2 (X2)	4B	RX1,RX2
SH	R1,A2 (FX2,SX2)	4B	RX3
SHI	R1,I2 (X2)	CB	RI1

Operation

The 16 bit second operand is expanded to a 32 bit fullword by propagating the most significant bit through Bits 15:0. This fullword is subtracted 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

Instructions

Compare
Compare Register
Compare Immediate

Assembler Notation		Op-Code	Format
C	R1, D2 (X2)	59	RX1, RX2
C	R1, A2 (FX2, SX2)	59	RX3
CR	R1, R2	09	RR
CI	R1, I2 (X2)	F9	RI2

Operation

The first operand, contained in the register specified by R1, is compared algebraically to the 32 bit second operand. The result is indicated by the Condition Code setting. Neither operand is changed.

Condition Code

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

Instructions

Compare Halfword-
Compare Halfword Immediate

Assembler Notation		Op-Code	Format
CH	R1,D2 (X2)	49	RX1,RX2
CH	R1,A2 (FX2,SX2)	49	RX3
CHI	R1,I2 (X2)	C9	RI1

Operation

The halfword second operand is expanded to a fullword by propagating the most significant bit through Bits 15:0. This fullword is compared algebraically with the first operand, the contents of the register specified by R1. The result is indicated by the Condition Code setting. Neither operand is changed.

Condition Code

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

Instructions

Multiply
Multiply Register

Assembler Notation		Op-Code	Format
M	R1, D2 (X2)	5C	RX1, RX2
M	R1, A2 (FX2, SX2)	5C	RX3
MR	R1, R2	1C	RR

Operation

The fullword first operand, contained in the register specified by R1 + 1, is multiplied by the fullword second operand. The 64 bit result is stored in 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 formats, the second operand must be located on a fullword boundary.

Instructions

Multiply Halfword
Multiply Halfword Register

Assembler Notation		Op-Code	Format
MH	R1,D2 (X2)	4C	RX1,RX2
MH	R1,A2 (FX2,SX2)	4C	RX3
MHR	R1,R2	0C	RR

Operation

The first operand, contained in Bits 16:31 of the register specified by R1, is multiplied by the 16 bit second operand, taken from memory or from Bits 16:31 of the register specified by R2. The 32 bit result replaces the contents of the register specified by R1.

Condition Code

Unchanged

Programming Note

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

Instructions

Divide
Divide Register

Assembler Notation		Op-Code	Format
D	R1, D2 (X2)	5D	RX1, RX2
D	R1, A2 (FX2, SX2)	5D	RX3
DR	R1, R2	1D	RR

Operation

The 64 bit dividend contained in the register specified by R1 and the register specified by R1+1 is divided by the fullword divisor. The 32 bit signed remainder replaces the contents of the register specified by R1. The 32 bit 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.

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

The quotient overflow causes an arithmetic fault interrupt (if enabled by Bit 19 of the PSW) if the value of the quotient would be greater than X'7FFFFFFF' or less than (more negative than) X'80000000'. On a divide fault, the operand registers are unchanged.

Divide Halfword
Divide Halfword Register

Assembler Notation		Op-Code	Format
DH	R1,D2 (X2)	4D	RX1,RX2
DH	R1,A2 (FX2,SX2)	4D	RX3
DHR	R1,R2	0D	RR

Operation

The 32 bit dividend contained in the register specified by R1 is divided by a 16 bit divisor taken from memory or from Bits 16:31 of the register specified by R2. The 16 bit remainder is expanded to a fullword by propagating the Sign bit through Bits 15:0 and is stored in the register specified by R1. The 16 bit quotient is expanded to a fullword by propagating the Sign bit through Bits 15:0 and is stored in the register specified by R1+1.

Condition Code

Unchanged

Programming Notes

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

Before starting the divide operation, the divisor is checked. If it is zero, the operation is aborted and the arithmetic fault interrupt taken, if enabled by Bit 19 of the current PSW.

The quotient overflow causes an arithmetic fault interrupt (if enabled by Bit 19 of the PSW) if the value of the quotient is greater than X'7XXX' or less than (more negative than) X'8000'.

Instruction

Shift Left Arithmetic

Assembler Notation		Op-Code	Format
SLA	R1, I2 (X2)	EF	RI1

Operation

Bits 1:31 of the first operand, contained in the register specified by R1, are shifted left the number of 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 31.

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.

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

Instruction

Shift Left Halfword Arithmetic

Assembler Notation		Op-Code	Format
SLHA	R1,I2 (X2)	CF	RI1

Operation

Bits 17:31 of the register specified by R1 are shifted left the number of places specified by the second operand. Bit 16 of the register, the halfword Sign bit, remains unchanged. Bits shifted out of Position 17 are shifted through the carry flag and then lost. The last bit shifted remains in the carry flag. Zeros are shifted into Position 31. Bits 0:15 of the first operand register remain 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

Programming Notes

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

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	R1,I2 (X2)	EE	RI1

Operation

Bits 1:31 of the first operand, contained in the register specified by R1, are shifted right the number of places specified by the second operand. The Sign bit (Bit 0), remains unchanged and is propagated right as many positions as specified by the second operand. Bits shifted out of Position 31 are shifted through the carry flag and lost. The last bit shifted remains in the carry flag.

Condition Code

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

Programming Notes

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

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

Instruction

Shift Right Halfword Arithmetic

Assembler Notation		Op-Code	Format
SRHA	R1,I2 (X2)	CE	RI1

Operation

Bits 17:31 of the register specified by R1 are shifted right the number of places specified by the second operand. Bit 16 of the register, the halfword Sign bit, remains unchanged and is propagated right the number of positions specified by the second operand. Bits shifted out of Position 31 are shifted through the carry flag and lost. The last bit shifted remains in the carry flag. Bits 0:15 of the first operand register remain 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

Programming Notes

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

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

Instruction

Convert to Halfword Value Register

Assembler Notation	Op-Code	Format
CHVR R1,R2	12	RR

Operation

The halfword second operand, (Bits 16:31) of the register specified by R2), is expanded to a fullword by propagating the most significant bit (Bit 16) through Bits 15:0. This fullword replaces the contents of the register specified by R1.

Condition Code

C	V	G	L	
X	X	0	0	Result is zero
X	X	0	1	Result is less than zero
X	X	1	0	Result is greater than zero
X	1	X	X	Source operand cannot be represented by a 16 bit signed number
1	X	X	X	Carry flag was set in previous Condition Code
0	X	X	X	Carry flag was reset in previous Condition Code

Programming Note

The V flag is set when Bits 0:15 of the second operand are not the same as Bit 16 of the second operand. (In this case, the G and L flags reflect the algebraic value of Bits 16:31 of the second operand.)

Execution of this instruction following halfword operations guarantees results identical with those that would be obtained if the program were run on an INTERDATA 16 bit machine. For example, assume that location A in memory contains the halfword value of X'7FFF' (decimal 32767) then,

```
LH     R1,A        R1 contains X'00007FFF'
AIS    R1,1        R1 contains X'00008000'
```

Following the add operation, the Condition Code is:

C	V	G	L
0	0	1	0

indicating a result greater than zero, which is correct for fullword operations. If the same sequence were executed on a 16 bit Processor, as:

```
LH    R1,A        R1 contains X'7FFF'
AIS   R1,1        R1 contains X'8000'
```

Following this, the Condition Code in the halfword processor is:

C	V	G	L
0	1	0	1

indicating overflow and a negative result. Going back to the original sequence and adding the Convert to Halfword Value instruction produces the following:

```
LH    R1,A        R1 contains X'00007FFF'
AIS   R1,1        R1 contains X'00008000'
CHVR  R1,R1       R1 contains X'FFFF8000'
```

Following this sequence, the Condition Code is:

C	V	G	L
0	1	0	1

which is identical to that of the 16 bit Processor, and can be tested in the same manner.

CHAPTER 5
FLOATING POINT ARITHMETIC

The Floating Point Arithmetic instructions provide a means for rapid manipulation of 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, and instructions for converting from floating point to fixed point and from fixed point to floating point.

OATA FORMATS

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

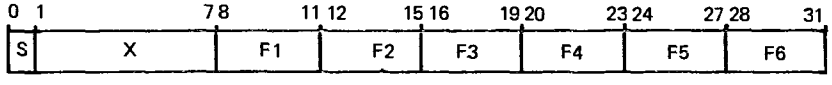


Figure 9. Floating Point Data Format

This form of representation requires 32 bits. The Sign bit (S) indicates whether the floating point value is positive or negative. The exponent field (X) 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). 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 Oecimal

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

1. Separate the decimal integer from the decimal fraction:

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

2. Convert each part to hexadecimal:

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

3. Combine the hexadecimal integer and fraction:

$$B6.6_{16} = (B6.6 \times 10^{0})_{16}$$

4. Shift the radix point:

$$(B6.6 \times 10^{0})_{16} = (.B66 \times 10^{2})_{16}$$

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

CONDITION CODE

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

FLOATING POINT INSTRUCTION FORMATS

The Floating Point instructions use the Register to Register (RR), and the Register and Indexed Storage (RX) instruction formats. In all of the RR formats, except for Fix and Float, the R1 and the R2 fields specify one of the floating point registers. There are eight floating point registers, numbered 0, 2, 4, 6, 8, 10, 12, and 14. In the RX formats, 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
LME	Load Multiple
STE	Store
STME	Store Multiple
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
FXR	Fix Register
FLR	Float Register

Load
Load Register

Assembler Notation		Op-Code	Format
LE	R1,D2 (X2)	68	RX1,RX2
LE	R1,A2 (FX2,SX2)	68	RX3
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 event, the result is forced to zero, X'0000 0000', the V flag in the Condition Code is set, the G and L flags are reset, and if enabled by Bit 19 of the current PSW, the arithmetic fault interrupt is taken.

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

Instruction

Load Multiple

Assembler Notation		Op-Code	Format
LME	R1, D2 (X2)	72	RX2, RX2
LME	R1, A2 (FX2, SX2)	72	RX3

Operation

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

Condition Code

Unchanged

Programming Notes

Values loaded into the floating point registers are not normalized.

The second operand must be located on a fullword boundary.

Instruction

Store

Assembler Notation		Op-Code	Format
STE	R1,D2 (X2)	60	RX1,RX2
STE	R1,A2 (FX2,SX2)	60	RX3

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

Store Multiple

Assembler Notation		Op-Code	Format
STME	R1,D2 (X2)	71	RX1,RX2
STME	R1,A2 (FX2,SX2)	71	RX3

Operation

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

Condition Code

Unchanged

Programming Note

The second operand must be located on a fullword boundary.

Add
Add Register

Assembler Notation		Op-Code	Format
AE	R1,D2 (X2)	6A	RX1,RX2
AE	R1,A2 (FX2,SX2)	6A	RX3
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 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 addition of the fractions produces a carry, incrementing the exponent of the result by one may produce exponent overflow. In this case, the result is forced to the maximum value, ±X'7FFF FFFF', the V flag, along with the G or L flag is set in the Condition Code, and if enabled by Bit 19 of the current PSW, the arithmetic fault interrupt is taken.

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

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

Subtract
Subtract Register

Assembler Notation		Op-Code	Format
SE	R1,D2 (X2)	6B	RX1,RX2
SE	R1,A2 (FX2,SX2)	6B	RX3
SER	R1,R2	2B	RR

Operation

The exponents of the two operands are compared. If the exponents differ, the fraction with the smaller exponent is shifted right hexadecimally (four bits at a time), and its exponent is incremented by one for each hexadecimal shift until the two exponents are equal. The 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 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 subtraction of fractions does not produce a carry, the result is normalized. The normalized result replaces the contents of the register specified by R1.

Conditon 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 produces a carry, incrementing the exponent of the result by one may produce exponent overflow. In this case, the result is forced to the maximum value, ±X'7FFF FFFF', the V flag, along with the G or L flag is set in the Condition Code, and if enabled by Bit 19 of the current PSW, the arithmetic fault interrupt is taken.

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

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

Instructions

Compare
Compare Register

Assembler Notation		Op-Code	Format
CE	R1,D2 (X2)	69	RX1,RX2
CE	R1,D2 (FX2,SX2)	69	RX3
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 is equal to second operand
1	X	0	1	First operand is less than second operand
0	X	1	0	First operand is greater than second operand

Programming Note

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

Multiply
Multiply Register

Assembler Notation		Op-Code	Format
ME	R1,D2 (X2)	6C	RX1,RX2
ME	R1,A2 (FX2,SX2)	6C	RX3
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
0	1	X	X	Exponent overflow
0	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, ±X'7FFF FFFF'. The V flag in the Condition Code is set, along with either the G or the L flag, depending on the sign of the result. An arithmetic fault interrupt is taken, if enabled by Bit 19 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 L flags are always reset, and if enabled by Bit 19 of the current PSW, the arithmetic fault interrupt is taken.

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

Instructions

Divide
Divide Register

Assembler Notation		Op-Code	Format
DE	R1,D2 (X2)	6D	RX1,RX2
DE	R1,A2 (FX2,SX2)	6D	RX3
DER	R1,R2	2D	RR

Operation

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

The 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 of the Condition Code are set. The G and L flags are reset. If enabled by Bit 19 of the current PSW, the arithmetic fault interrupt is taken.

The subtraction of exponents may produce exponent overflow. In this case, the result is forced to the maximum value, ±X'7FFF FFFF'. The V flag in the Condition Code is set, along with either the G or the L flag, depending on the sign of the result. An arithmetic fault interrupt is taken, if enabled by Bit 19 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 L flags are always reset, and if enabled by Bit 19 of the current PSW, the arithmetic fault interrupt is taken.

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

Instruction

Fix Register

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

Operation

R1 specifies one of the general purpose registers. R2 specifies one of the floating point registers. The floating point number contained in the floating point register is converted to an integer value by truncating. The result is stored in the register specified by R1.

Condition Code

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

Programming Notes

On fullword overflow, the result is forced to the maximum value, ±X'7FFF FFFF'.

On halfword overflow, the result is not changed.

Instruction

Float Register

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

Operation

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

Condition Code

C	V	G	L	
0	0	0	0	Result is zero
0	0	0	1	Result is less than zero
0	0	1	0	Result is greater than zero

Programming Note

The result in R1 is normalized.

CHAPTER 6
STATUS SWITCHING AND INTERRUPTS

At any given time, the Processor may be in either the Stop mode 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 any memory location

Change any memory location

Examine the contents of any general register

Examine the current PSW

Execute instructions singly

The translation from the Stop mode to the Run mode requires operator intervention at the display console.

Once the Processor has been put in the Run mode, the current PSW 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 supervisor and protect modes

Vary the normal sequential execution of instructions

PROGRAM STATUS WORD

The Program Status Word is a 64 bit double word.

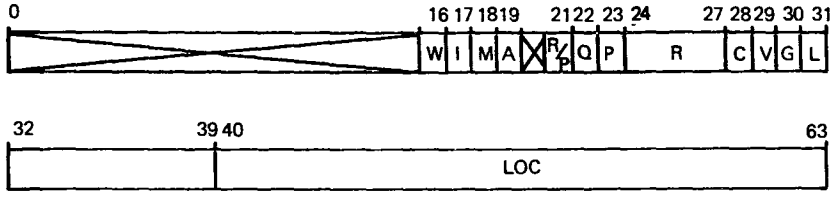


Figure 10. Program Status Word

Bits 0:15 of the PSW are not currently used, and must be zero. Bits 16:27 are reserved for status definition and interrupt masks. Note that Bit 20 is not currently assigned a specific function. This bit must always be zero. Bits 28:31 are reserved for the Condition Code. Bits 32:39 are not used, and must be zero. Bits 40:63 are reserved for the Location Counter. The status and interrupt bits are interpreted as follows:

Bit 16 (W)	Wait state
Bit 17 (I)	Immediate interrupt enable
Bit 18 (M)	Machine malfunction interrupt enable
Bit 19 (A)	Arithmetic fault interrupt enable
Bit 21 (RP)	Relocation/protection enable
Bit 22 (Q)	System queue service interrupt enable
Bit 23 (P)	Protect mode
Bits 24:27 (R)	Register set selection

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 0:31) 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 16 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 and immediate interrupts, if they are enabled. If the Processor is put in the Wait state with these interrupts disabled, only operator intervention from the display console can force the Processor out of the Wait state.

Protect Mode

When Bit 23 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 23 of the current PSW is reset, the Processor is in the Supervisor mode. Programs running in this mode may execute any legal instruction.

Register Set Selection

Bits 24:27 of the current PSW control register set selection. If these bits are reset, general register set 0 is used. If these bits are set, general register set 15 is used.

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
Immediate, Auto Driver Channel	17
Console	17
Machine Malfunction	18
Arithmetic Fault	19
System Queue Service	22
Protect Mode Violation	23
Relocation/Protection	21
Supervisor Call	none
Simulated	none
Illegal Instruction	none

Interrupts occur at various times during processing. The immediate, console, and machine malfunction interrupts occur between the execution of instructions or after completion of an auto driver channel operation. The relocation/protection interrupt occurs after the execution of an instruction. The system queue service, arithmetic fault, supervisor call, and simulated interrupts occur during the execution of instructions. The illegal instruction and protect mode violation interrupts occur before the execution of the improper instruction.

The interrupt procedure is based on the concepts of old, current, and new Program Status Words. The current PSW, contained in a 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.

With one exception (the machine malfunction interrupt), when the current PSW becomes the old PSW it is saved in a pair of registers belonging to register set 0. The machine malfunction old PSW is stored in a reserved memory location. Again with one exception, when a new PSW becomes the current PSW, it is loaded from a reserved memory location. The exception is the immediate interrupt. On an immediate interrupt, the current status is forced to a predetermined value. The current Location Counter is loaded from the interrupt service pointer table.

The new Program Status Word for any interrupt should, if possible, disable interrupts of its own class, and should, to avoid the overhead of saving registers, specify register set 0.

Immediate Interrupt

The immediate interrupt is used for I/O control. Through this mechanism, external devices can request and obtain Processor service. Bit 17 of the current PSW controls the immediate interrupt. If this bit is set, the Processor is responsive to device requests. If this bit is reset, requests are queued until the Processor is able to recognize them. When the Processor recognizes a request from a device it:

Saves the current PSW in Registers 0 and 1 of general register set 0.

Loads the status portion (Bits 0:31) of the current PSW with a value of X'00002000'.

Acknowledges the request and obtains the device number and status from the device. The device number is placed in Register 2 of the register set 0. The status is placed in Register 3.

Adds two times the device number to X'0000D0' (the starting location of the interrupt service pointer table) to obtain the address within the table that corresponds to the interrupting device. For the immediate interrupt, the value in the table must be even. The value in the table becomes the current Location Counter.

In setting up the registers for the immediate interrupt service routine, the Processor loads the device number and status into the least significant bits of Registers 2 and 3. The most significant bits are forced to zero.

Note that the current PSW for immediate interrupts disables the immediate interrupt and specifies register set 0. If it is desired to run the interrupt routine with interrupts enabled, the routine must save the information contained in Registers 0:4, and should switch to register set 15.

Console Interrupt

The console interrupt is a special case of the immediate interrupt. It too is controlled by Bit 17 of the current PSW. If this bit is set, a console interrupt is generated by:

Depressing the Function key on the console

Depressing 0

The effect of the console interrupt is to cause an immediate interrupt, as described previously, from device X'001'.

Simulated Interrupt

The Simulate Interrupt instruction simulates an immediate interrupt. When this instruction is executed, the Processor goes through the immediate interrupt procedure as if a request for service had been received from an external device. The current PSW is saved, and the current PSW loaded just as for the immediate interrupt. The device is addressed, and the status returned in Register 3. The address from the interrupt service pointer table is placed in Register 4. The state of Bit 17, immediate interrupt enable, has no effect on this interrupt. It is always enabled.

Machine Malfunction Interrupt

Bit 18 of the current PSW controls the machine malfunction interrupt. This interrupt occurs on a memory parity error, on the detection of primary power failure, and during the restart procedure after power has been restored. When a machine malfunction interrupt occurs, the current PSW is saved in memory location X'000020'. The new PSW from memory location X'000038' becomes the current PSW. The Condition Code of the new PSW as stored in memory must contain zeros. After the interrupt is taken, the state of the Condition Code indicates the specific cause of the interrupt.

Condition Code states are:

C	V	G	L	
0	0	0	0	Power restore
X	0	0	1	Power failure
0	0	1	0	Parity error on instruction fetch
X	1	0	0	Parity error on data fetch
1	X	0	X	Parity error or power failure during auto driver channel operation

Power failure occurs when the primary power fail detector senses a low voltage, when the Initialize key (INI) 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 at the memory location specified by the contents of location X'000084'; and it saves both sets of general registers, starting with register set 0, at the location specified by the contents of memory location X'000086'.

When power returns, the Processor restores the PSW and the general registers from their save areas. If Bit 18 of the restored PSW is set, the Processor takes another machine malfunction interrupt, this time with no bits set in the Condition Code of the current PSW.

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 V or the G flag to indicate error on data fetch or instruction fetch.

If a machine malfunction interrupt condition arises during an auto driver channel operation, the PSW, current at the time the channel was activated, becomes the old machine malfunction PSW. Register 4 of register set 0 contains the address of the Channel Command Block. The C flag of the current PSW is set along with either the L flag or the V flag to indicate either power failure or parity error on a data fetch.

Arithmetic Fault Interrupt

Bit 19 of the current PSW controls the arithmetic fault interrupt. This interrupt, if enabled, can occur for any of the following reasons:

Fixed point division by zero
Fixed point quotient overflow
Floating point division by zero
Floating point overflow or underflow

When this interrupt occurs, the current PSW is saved in Registers 14 and 15 of register set 0. The new PSW, from memory location X'000048', becomes the current PSW. All Condition Code bits in the new PSW as stored in memory must be zero. Before going to the interrupt service routine, the Processor sets the carry flag in the Condition Code if the interrupt is the result of a floating point operation. If the interrupt is the result of a fixed point operation, the carry flag is not set.

Any of the following conditions cause fixed point quotient overflow:

A halfword divide operation produces a result greater than 32,767.

A halfword divide operation produces a result less than -32,768.

A fullword divide operation produces a result greater than 2,147,483,647.

A fullword divide operation produces a result less than -2,147,483,648.

When a fixed point division by zero or a fixed point quotient overflow occurs, the operand registers remain unchanged.

Floating point overflow occurs when, in a floating point operation, the value of the exponent exceeds 127. Floating point underflow occurs when, during the execution of a Floating Point instruction, the value of the exponent becomes negative. Following floating point overflow, the result is forced to plus or minus X'7FFF FFFF'. Following a floating point underflow, the result is forced to true zero, X'0000 0000'. After a floating point division by zero, the operand register remains unchanged.

After any arithmetic fault interrupt, the Location Counter of the old PSW contains the address of the instruction immediately following the one that caused the interrupt.

Relocation/Protection Interrupt

Bit 21 of the current PSW controls the relocation/protection interrupt. If this bit is set, and the currently running program violates any of the relocation and protection conditions available in the relocation and protection module, the Processor saves the current PSW in Registers 14 and 15 of register set 0. The new PSW at memory location X'000090' becomes the current PSW.

System Queue Service Interrupt

Memory location X'000080' contains the address of the system queue. In the course of executing any of the following instructions:

Load Program Status Word
Load Program Status Word Register
Exchange Program Status

the Processor tests Bit 22 of the new status being loaded. If this bit is set, the Processor checks the state of the system queue. If there is an entry in the queue, the just loaded PSW becomes the old PSW. It is saved in Registers 14 and 15 of register set 0. The address of the queue, taken from location X'000080', is placed in Register 13 of register set 0. The new PSW from location X'000088' becomes the current PSW.

Protect Mode Violation Interrupt

Bit 23 of the current PSW controls the execution of Privileged instructions. When this bit is set, the Processor is in the Protect mode. Programs running in the Protect mode are not allowed to execute Privileged instructions. Privileged instructions are:

All I/O instructions
Load Program Status Word
Load Program Status Word Register
Exchange Program Status Register
Simulate Interrupt
Simulate Channel Program

If a program running in the protect mode attempts to execute a Privileged instruction, the instruction is not executed. The Processor saves the current PSW in Registers 14 and 15 of register set 0. The new PSW at location X'000030' becomes the current PSW. The Location Counter of the old PSW contains the address of the Privileged instruction.

Illegal Instruction Interrupt

The illegal instruction interrupt cannot be disabled. The interrupt occurs whenever the Processor fetches an instruction word containing an operation code that is not one of those permitted by the system. The Processor saves the current PSW in Registers 14 and 15 of register set 0. The illegal instruction new PSW from memory location X'000030' becomes the current PSW.

When the Processor encounters an illegal instruction, it makes no attempt to execute it. The Location Counter of the old PSW contains the address of the illegal instruction.

Supervisor Call Interrupt

This interrupt occurs as the result of the execution of a Supervisor Call instruction. This instruction provides a means for user level programs to communicate with system programs. The supervisor call interrupt is always enabled. When the Processor executes a Supervisor Call instruction, it:

Saves the current PSW in Registers 14 and 15 of register set 0.

Places the address of the supervisor call parameter block (address of the second operand) in in Register 13 of register set 0.

Loads the current PSW status with the value contained at memory location X'000098', supervisor call new status.

Loads the current PSW Location Counter from one of the supervisor call new PSW Location Counter locations.

STATUS SWITCHING INSTRUCTION FORMATS

The Status Switching instructions use the Register to Register (RR), and the Register and Indexed Storage (RX) instruction formats. In some cases, Load Program Status Word and Load Program Status Word Register, and 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 efficiently with control software. All Status Switching instructions, except the Supervisor Call instruction are privileged operations. Because of this all interrupt handling routines must run in the Supervisor mode.

The instructions described in this section are:

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

Instruction

Load Program Status Word

Mnemonic		Op-Code	Format
LPSW	D2 (X2)	C2	RX1, RX2
LPSW	A2 (FX2, SX2)	C2	RX3

Operation

The 64 bit second operand becomes the current Program Status Word.

Condition Code

Determined by the new PSW

Programming Note

The quantity to be loaded into the current Program Status Word must be located in memory on a double word boundary.

This instruction is a privileged operation.

Instruction

Load Program Status Word Register

Mnemonic		Op-Code	Format
LPSWR	R2	18	RR

Operation

The contents of the register specified by R2 replace Bits 0:31 of the current Program Status Word. The contents of the register specified by R2+1 replace Bits 32:63 of the current Program Status Word.

Condition Code

Determined by new PSW

Programming Notes

The R1 field of this instruction has no significance and must always be zero.

This instruction may be used to change register sets. The new set becomes active upon execution of the next instruction.

This instruction is a privileged operation.

The R2 field of this instruction may not specify a register greater than 14.

Instruction

Exchange Program Status Register

Mnemonic		Op-Code	Format
EPSR	R1, R2	95	RR

Operation

Bits 0:31 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 0:31 of the current Program Status Word.

Condition Code

Determined by new status

Programming Notes

If R1 = R2, Bits 0:31 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

Mnemonic		Op-Code	Format
SINT	D2 (X2)	E2	RI1
SINT	A2 (FX2, SX2)	E2	

Operation

The least significant 10 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 auto driver channel operation.

Programming Notes

The R1 field of this instruction must contain zero.

In the execution of this instruction, the Processor loads Registers 0:4 of register set 0 as for a real interrupt request.

This instruction is a privileged operation.

During the execution of this instruction, the device is addressed and the status returned in Register 3 of register set 0.

Instruction

Supervisor Call

Mnemonic		Op-Code	Format
SVC	D2 (X2)	E1	RX1, RX2
SVC	A2 (FX2, SX2)	E1	RX3

Operation

The address of the second operand replaces Bits 8:31 of Register 13 of register set 0. Bits 0:7 of this register are forced to zero. The current Program Status Word replaces the contents of Registers 14 and 15 of register set 0. The fullword quantity located at X'000098' in memory replaces Bits 0:31 of the current Program Status Word. The R1 field is doubled and added to X'00009C'. The halfword quantity located at this address becomes the current Location Counter.

Condition Code

Determined by the new PSW

CHAPTER 7
INPUT OUTPUT OPERATIONS

Input output (I/O) operations, as defined for the 32 bit series, 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 between external devices and memory 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 independent 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 1,023 external devices. Each device must have its own unique device number or address. Device numbers may range from X'001' through X'3FF'. (Device number X'000' is not used.) The minimum system has provision for 255 device numbers. Larger systems may have either 511 or 1,023.

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 controller is a bi-directional, 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, 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 17 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.

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. Placement of controllers on the I/O Bus determines 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 farther down in line must wait until the Processor has acknoweldged and acted upon requests from higher priority controllers. Requests for attention remain queued until all have been serviced.

INTERRUPT SERVICE POINTER TABLE

Device requests for service may result in either an immediate interrupt or an auto driver 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'0000D0' and contains a halfword entry for each device number in the system. For a minimum system, 255 device numbers, the table extends through memory location X'0002CF'; for a maximum system, the table extends through memory location X'0008CF'. The software controlling I/O operations must set up the table.

When, having acknowledged a request for service, the Processor receives the device address, it adds two times the device address to X'000D0'. The result is the address, within the table, of the entry reserved for the device requesting attention.

If the entry in the table is even (Bit 15 equals 0), the Processor taken an immediate interrupt and transfers control to the software routine at the address contained in the table. If the entry in the table is odd (Bit 15 equals 1), the Processor transfers control to the auto driver channel, without interrupting the currently running program.

At the time the Processor transfers control to the software routine, the old PSW (current at the time of the device request) has been saved in Registers 0 and 1 of general register set 0. The device number has been placed in Register 2, the device status in Register 3. Other registers in register set 0 are undefined. The status portion of the current PSW has been forced to a value of X'0000-4000', which specifies running state, register set 0, machine malfunction interrupt enabled, and all other interrupts disabled. The entry in the interrupt service pointer table has become the new Location Counter.

In using the device number presented by the controller to vector into the interrupt service pointer table, the Processor masks off the high order bits of the address as received from the I/O Bus. In a system with only 255 device numbers, the address is masked to eight bits. In a system with 1,023 device numbers, the address is masked to 10 bits. This action preserves system integrity in the event that a hardware malfunction results in a device address greater than the maximum allowed in the system.

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 at all, 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 device status byte. If all of these bits are zero, device time-out has occurred.

The instructions described in this section are:

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
SCP	Simulate Channel Program

Instructions

Sense Status
Sense Status Register

Assembler Notation		Op-Code	Format
SS	R1,D2 (X2)	DD	RX1,RX2
SS	R1,A2 (FX2,SX2)	DD	RX3
SSR	R1,R2	9D	RR

Operation

Bits 22:31 of the register specified by R1 contain the 10 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 right most four bits of the device status byte. The first operand is unchanged.

Condition Code

Bits 4:7 of the device status byte are copied into the Condition Code. See the appropriate device manual for a description of this status.

Programming Notes

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

In the RR format, the device status byte replaces Bits 24:31 of the register specified by R2. Bits 0:23 are forced to zero.

These instructions are privileged operations.

Instruction

Output Command
Output Command Register

Assembler Notation		Op-Code	Format
OC	R1,D2 (X2)	DE	RX1,RX2
OC	R1,A2 (FX2,SX2)	DE	RX3
OCR	R1,R2	9E	RR

Operation

Bits 22:31 of the register specified by R1 contain the 10 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 24:31 of the register specified by R2 contain the device command.

These instructions are privileged operations.

Instructions

Read Data
Read Data Register

Assembler Notation		Op-Code	Format
RD	R1,D2 (X2)	DB	RX1,RX2
RD	R1,A2 (FX2,SX2)	DB	RX3
RDR	R1,R2	9B	RR

Operation

Bits 22:31 of the register specified by R1 contain the 10 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 24:31 of the register specified by R2. Bits 0:23 of the register are forced to zero.

These instructions are privileged operations.

Read Halfword
Read Halfword Register

Assembler Notation		Op-Code	Format
RH	R1,D2 (X2)	D9	RX1,RX2
RH	R1,A2 (FX2,SX2)	D9	RX3
RHR	R1,R2	99	RR

Operation

Bits 22:31 of the register specified by R1 contain the 10 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 Bits 16:31 of the register specified by R1. Bits 0:15 are forced to zero. The first byte of data from a byte device replaces Bits 16:23 of the register specified by R1. The second byte replaces Bits 24:31. Bits 0:15 are forced to zero.

If the device is byte-oriented, it must be capable of supplying both bytes without intervening status checks. Unlike the RB and RBR instructions, this instruction does not perform status checking between the two byte transfers.

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,D2 (X2)	D7	RX1,RX2
RB	R1,A2 (FX2,SX2)	D7	RX3

Operation

Bits 22:31 of the register specified by R1 contain the 10 bit device address. Bits 8:31 of the fullword located at the second operand address contain the starting address of the data buffer. Bits 8:31 of the fullword located at the second operand address plus four 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

Bits 4:7 of the device status byte are copied into the Condition Code. See the appropriate device manual for a description of this status.

Programming Notes

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 of data 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 22:31 of the register specified by R1 contain the 10 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

Bits 4:7 of the device status byte are copied into the Condition Code. See the appropriate device manual for a description of this status.

Programming Notes

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 of data is transmitted.

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

This instruction is a privileged operation.

Instructions

Write Data
Write Data Register

Assembler Notation		Op-Code	Format
WD	R1,S2 (X2)	DA	RX1,RX2
WD	R1,A2 (FX2,SX2)	DA	RX3
WDR	R1,R2	9A	RR

Operation

Bits 22:31 of the register specified by R1 contain the 10 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 eight bit data byte is contained in Bits 24:31 of the register specified by R2.

These instructions are privileged operations.

Instructions

Write Halfword
Write Halfword Register

Assembler Notation		Op-Code	Format
WH	R1,D2 (X2)	D8	RX1,RX2
WH	R1,A2 (FX2,SX2)	D8	RX3
WHR	R1,R2	98	RR

Operation

Bits 22:31 of the register specified by R1 contain the 10 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 16:31 of the register specified by R2. The first byte of data transmitted to a byte device comes from Bits 16:23 of the register specified by R2, the second byte, from Bits 24:31.

If the device is byte-oriented, it must be capable of accepting both bytes without intervening status checks. Unlike the WB and WBR instructions, this instruction does not perform status checking between the two byte transfers.

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,D2 (X2)	D6	RX1, RX2
WB	R1,A2 (FX2,SX2)	D6	RX3

Operation

Bits 22:31 of the register specified by R1 contain the 10 bit device address. Bits 8:31 of the fullword located at the second operand address contain the starting address of the data buffer. Bits 8:31 of the fullword located at the second operand address plus four contain the ending address or the data buffer.

The Processor transmits eight bit data bytes from consecutive locations in the specified buffer to the device.

Condition Code

Bits 4:7 of the device status byte are copied into the Condition Code. See the appropriate device manual for a description of this status.

Programming Notes

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 of data 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 22:31 of the register specified by R1 contain the 10 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

Bits 4:7 of the device status byte are copied into the Condition Code. See the appropriate device manual for a description of this status.

Programming Notes

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 of data 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	D2 (X2)	D5	RX1, RX2
AL	A2 (FX2, SX2)	D5	RX3

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'000080'. 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 eight bit input device address is specified by memory location X'000078'. The device command code is specified by memory location X'000079'.

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
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

This instruction may only be used with devices whose addresses are less than, or equal to, X'FF'.

The R1 field of this instruction must be zero.

This instruction is a privileged operation.

Instruction

Simulate Channel Program

Assembler Notation		Op-Code	Format
SCP	R1,D2 (X2)	E3	RX1,RX2
SCP	R1,A2 (FX2,SX2)	E3	RX3

Operation

The second operand address is the address of a Channel Command Block (CCB). The buffer switch bit of the Channel Command Word (CCW) specifies the buffer to be used for the data transfer. If this bit is set, Buffer 1 is used. If it is reset, Buffer 0 is used. If the byte count field of the current buffer is positive, the V flag in the Condition Code is set, and the next sequential instruction is executed. If the byte count field is not positive, the following date transfer operation is performed.

If the Channel Command Word specifies read, a byte of data is moved from Bits 24:31 of the register specified by R1 to the appropriate buffer location. If the Channel Command Word specifies write, a byte of data is moved from the appropriate buffer location to Bits 24:31 of the register specified by R1. Bits 0:23 are forced to zero.

After a byte has been transferred, the count field of the appropriate buffer is incremented by one. If the count field is now positive, and if the fast bit of the CCW is reset, the buffer switch bit of the CCW is complemented.

Condition Code

C	V	G	L	
0	0	0	0	Count field is now zero
0	0	0	1	Count field is now less than zero
0	0	1	0	Count field is now greater than zero
0	1	0	0	Count field was positive

Programming Notes

The second operand must be located on a fullword boundary.

This instruction is a privileged operation.

CONTROL OF I/O OPERATIONS

The design of the 32 bit series 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 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 auto driver 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 23 of the current PSW reset. I/O control programs should disable immediate interrupts, controlled by Bit 17 of the current PSW, and use general register set 0. General register set 0 is also used for other interrupts, and the software must establish conventions to govern the sharing of these registers.

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 transfers 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 run with the immediate interrupt disabled, and use register set 0. On the second level are the interruptable programs. They run with the immediate interrupt enabled, and use register set 15.

Before starting interrupt driven operations, the interrupt service pointer table must be set up. This table starts at memory location X'0000D0'. It must contain a halfword address entry for every possible device. The table is ordered according to device addresses in such a way that X'0000D0' plus two times the device address equals the memory address of the table entry reserved for that device. The value placed in the location reserved for a device is the address of the interrupt service routine for the device.

Although there may be gaps in 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 done best with the immediate interrupt disabled. Once the table pointer has been set up, the and 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 location specified in the interrupt service pointer table. At this time, the current PSW has a status that indicates running state, register set 0, machine malfunction interrupt enabled, and all other interrupts disabled. Registers 0 and 1 of register set 0 contain the old PSW, indicating the status and location of the interrupted program. Register 2 of register set 0 contains the device address. Register 3 contains the device status. The software routine can now:

1. Check the device status in Register 2, and if satisfactory,
2. Make the transfer, and
3. Return to the interrupted program by reloading the old PSW from Registers 0 and 1.

The interrupt service routine must not enable the immediate interrupt while it is working with register set 0. To do so allows other interrupt requests to be acknowledged, and the contents of Registers 0:4 would be lost. If it is necessary to enable the immediate interrupt, the routine should save register set 0, switch to register set 15, saving it if necessary, and then enable the immediate interrupt.

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 program processing may proceed simultaneously with the transfer of data between the external device and memory. This is possible because the Selector Channel accesses memory on a cycle stealing basis, which permits the Processor and the channel to share memory. In some cases, execution times of the program in progress may be affected, while in others, the effect is negligible. This depends upon the rate at which the Selector Channel and Processor compete for memory cycles.

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 immediate interrupt.

Selector Channel Devices

The Selector Channel has a private bus similar to the Processor's I/O Bus. Controllers for the devices associated with the Selector Channel are attached to this bus. When the Selector Channel is idle, its private bus is connected directly to the I/O Bus. If this condition exists, the Processor can address, command, and accept interrupt requests from the devices attached to the Selector Channel. When the Selector Channel is busy, this connection is broken. All communication between the Processor and devices on the Selector Channel are cut off. Any attempt by the Processor to address devices on the channel results in instruction time-out.

Selector Channel Operation

Two registers in the Selector Channel hold the current memory address and the final memory address. Before starting a Selector Channel operation, the control software, using Write instructions, places the address of the first byte of the data buffer in the current register and the address of the last byte 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 integer 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, then 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 commands and information 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, the Processor can be directed to the execution of concurrent programs.

On termination, the channel signals the Processor that it requires service. The Processor subsequently takes an immediate interrupt, transferring control to the Selector Channel interrupt service routine. At this time, Registers 0:3 of register set 0 are set up as for any other immediate interrupt.

AUTO DRIVER CHANNEL

The auto driver channel provides a means for multiplexing block data transfers between memory and low or medium speed I/O devices. The operation of the channel is similar in some respects to interrupt driven I/O. The channel is activated upon a service request from a device on the I/O Bus. Upon receipt of a device request, the Processor uses the device number to index into the interrupt service pointer table. If the value contained in the table is even, the Processor transfers control to the interrupt service routine. If the value is odd, it transfers control to the auto driver channel.

To the auto driver channel, the address in the interrupt service pointer table is the address plus one (making it odd) of a Channel Command Block (CCB). The Channel Command Block is basically a channel program consisting of a description of the operation to be performed, and a list of parameters associated with the operation. In addition to the functions of read and write, the channel can translate characters, test device status, chain buffers, calculate longitudinal and cyclic redundancy check values, and transfer control to software routines to take care of unusual situations.

CHANNEL COMMAND BLOCK

The Channel Command Block (CCB), as shown in Figure 11, consists of a Channel Command Word (16 bits) that describes the function, count fields (16 bits each) for two buffers, final addresses (32 bits each) for two buffers, a check word (16 bits) for the longitudinal or cyclic redundancy check, the address (32 bits) of a translation table, and the address (16 bits) of a software routine.

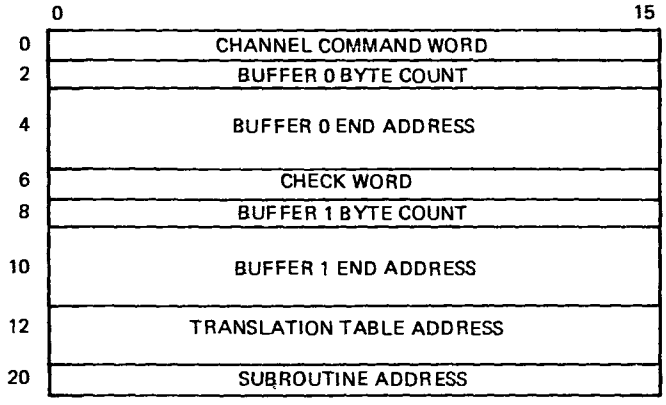


Figure 11. Channel Command Block

Just as there may be many interrupt service routines ready at any time to service device requests, there may be many channel command blocks in the system ready to handle data transfers as required. Each channel command block must start on a fullword boundary. The address plus one of the channel command block is placed in the interrupt service pointer table location for the device involved in the transfer.

Subroutine Address

When the channel transfers control to the software subroutine whose address is contained in the Channel Command Block, Registers 0:4 of register set 0 have already been set up by the Processor to contain the old PSW, the device number, the device status, and the address of the Channel Command Block. The current PSW status specifies run state, register set 0, machine malfunction interrupt enabled, and all other interrupts disabled.

The channel transfers control to the subroutine either unconditionally (controlled by a bit in the Channel Command Word), or because of bad device status, or because it has reached the limit of a buffer. It indicates its reason for transferring control by adjusting the Condition Code as follows.

C	V	G	L	
0	0	0	0	Unconditional transfer
0	0	0	1	Bad status
0	0	1	0	Buffer limit

The subroutine address in the CCB is a 16 bit address. Because of this, the subroutine, or at least the first instruction of the subroutine, must reside in the first 64KB of memory.

Buffers

There is space in the CCB to describe two data buffer areas. The data areas may be located anywhere in memory. The limits of each data area are described by an address field and a count field. The address field contains the address of the last byte in the data area. This is a 24 bit address, right justified in the fullword provided. If the device being controlled is a halfword device, the final address must be odd. If the device is a byte device, the address may be either odd or even.

The count field, in most operations, contains a negative number whose absolute value is equal to one less than the number of bytes to be transferred. The one exception is the case of a single byte transfer, where the count field contains zero.

During data transfers, the channel adds the value contained in the count field to the final address to obtain the current address. It makes the transfer, referencing the current address, then increments the value in the count field by one for a byte device or by two for a halfword device. When the count field becomes positive, i.e., greater than zero, the channel sets the G flag in the Condition Code and transfers control to the specified software subroutine.

Translation

The translation feature is available only for byte devices. If this operation is specified, the fullword provided in the Channel Command Block must contain the 24 bit address, right justified, of a translation table. The table, which must start on a halfword boundary, can contain up to 256 halfword entries. During data transfers, the channel multiplies the data byte by two and adds this value to the translation table address. The result is the address within the translation table of the halfword corresponding to the data byte.

The channel references this location, and, if Bit 0 of the halfword is a one, it substitutes Bits 8:15 of the halfword for the data byte and proceeds with the operation. If Bit 0 of the halfword is a zero, the channel:

Puts the data byte, untranslated, in Bits 24:31 of Register 3, register set 0.

Forces Bits 0:23 of Register 3 to zero.

Multiplies the value contained in the translation table by two, and transfers control to the software routine located at this address.

Upon transfer to the translation subroutine, Registers 0 and 1 of set 0 contain the old PSW. Register 2 contains the device number. Register 3 contains the untranslated character. Register 4 contains the address of the Channel Command Block. The current PSW indicates run state, register set 0, machine malfunction interrupt enabled, and all other interrupts disabled. The Condition Code is zero.

Check Word

If either longitudinal or cyclic redundancy checking is required, the check word in the Channel Command Block contains the accumulated value. The initial value for the check word is usually zero. (There are data dependent exceptions, e.g., where initial characters are not to be included in the check.) The longitudinal check is an Exclusive OR of the character with the check word. The cyclic check uses the formula:

$$X^{16} + X^{15} + X^2 + 1$$

On input, if both redundancy checking and translation are required, the redundancy check is done first, then the character is translated. On output, the character is translated first. Redundancy checking may be used only with byte devices.

Channel Command Word

The Channel Command Word (CCW), as shown in Figure 12, consists of two parts. Bits 0:7 contain a status mask. Bits 8:15 describe the channel operation.

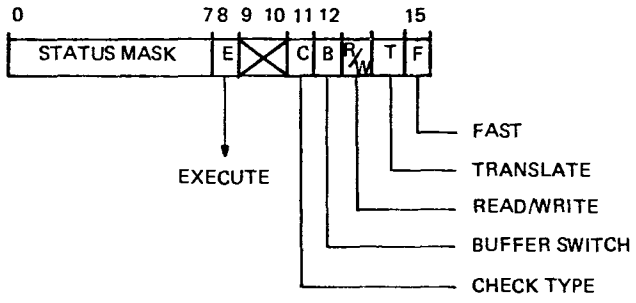


Figure 12. Channel Command Word

Status Mask

On every channel operation involving a data transfer, the status mask is ANDed with the device status. This operation does not change the status mask. If the result is zero, the channel proceeds with the operation. If the result is non-zero, the channel sets the L flag in the Condition Code, and transfers control to the specified software subroutine.

Execute Bit (E)

If this bit is reset, the channel transfers control to the specified subroutine, without taking any other action. The Condition Code is zero. If this bit is set, the channel continues with the operation as specified in the Channel Command Word.

Fast Bit (F)

If this bit is set, the channel performs the I/O transfer in the fast mode. In the fast mode, buffer chaining, redundancy checking, and translation are not allowed. This bit must be set for halfword devices. If this bit is set, Buffer 0 is always used.

Read/Write Bit (R/W)

This bit indicates the type of operation. If this bit is reset, a byte or a halfword is input from the device. If this bit is set, a byte or a halfword is output to the device.

Translate Bit (T)

If this bit is set, and the fast bit reset, the channel translates the data byte.

Check Type Bit (C)

This bit specifies the type of check required. If it is set, the channel performs a cyclic redundancy check. If it is reset, the channel performs a longitudinal redundancy check. This bit is ignored if the fast bit is set.

Buffer Switch Bit (B)

When the fast bit is reset, this bit specifies which of the two buffers is to be used for the transfer. If this bit is reset, Buffer 0 is used. If it is set, Buffer 1 is used. The channel chains buffers when the count field becomes positive. It does this by complementing the buffer switch bit before transferring control to the specified software routine.

Valid Channel Command Codes

The following is a list of valid codes for the Channel Command Word. Note that only the first three may be used with halfword devices.

Channel Command Word 8:15

Hexadecimal	Binary	Meaning
00	00000000	Transfer to subroutine
81	10000001	Read fast mode
85	10000101	Write, fast mode
80	10000000	LRC, Buffer 0, read
82	10000010	LRC, Buffer 0, read, translate
84	10000100	LRC, Buffer 0, write
86	10000110	LRC, Buffer 0, write, translate
88	10001000	LRC, Buffer 1, read
8A	10001010	LRC, Buffer 1, read, translate
8C	10001100	LRC, Buffer 1, write
8E	10001110	LRC, Buffer 1, write, translate
90	10010000	CRC, Buffer 0, read
92	10010010	CRC, Buffer 0, read, translate
94	10010100	CRC, Buffer 0, write
96	10010110	CRC, Buffer 0, write, translate
98	10011000	CRC, Buffer 1, read
9A	10011010	CRC, Buffer 1, read, translate
9C	10011100	CRC, Buffer 1, write
9E	10011110	CRC, Buffer 1, write, translate

CHAPTER 8
MEMORY MANAGEMENT

The memory access and protect controller is an auxiliary module available with the INTERDATA extended series Processors. It provides:

Program segmentation
Automatic relocation of programs
Memory protection

The controller operates on a 20 bit program (or virtual) address produced by the Processor. It converts this address, through an addition process, into a real address, again 20 bits, in memory. At the same time, it checks the program address against a preset limit which it is not allowed to exceed. It verifies that the item referenced by the program address is actually in memory. It also checks the type of memory access, that is, instruction fetch, memory read, or memory write, against the allowable operations. If the program address exceeds the limit, or if the item is not in memory, or if the operation is not allowed, the controller notifies the Processor by generating relocation/protection interrupt.

In an operating system environment, the operation of the memory access and protect controller is completely transparent to most programs. It is very similar to a peripheral device in that only the operating system modules directly responsible for its operation need to be aware of its existence.

The memory access and protect controller contains seventeen hardware registers. Sixteen of the registers, called segmentation registers, contain relocation, protection, and control information. The seventeenth register is the interrupt status register. When the controller interrupts the Processor, it sets this register to indicate the type of violation that caused the interrupt.

The segmentation registers and the status register are assigned absolute memory locations. Memory reference instructions may be used to load the registers and to read out the least significant byte of the status register. The block of memory locations reserved for these registers depends on the particular configuration.

If the configuration has provision for no more than 255 external devices, the segmentation registers are assigned locations X'000300' through X'00033F'. The status register is assigned the fullword location at X'000340'.

If the configuration has provision for more than 255 external devices, the block of memory locations assigned to the controller starts at the nearest multiple of X'000100' above the expanded interrupt service pointer table.

Bit 21 of the current Program Status Word enables the segmentation, relocation, and protection features of the memory access and protect controller. If this bit is set, segmentation, relocation, protection, and the relocation/protection interrupt are enabled. If this bit is reset, all memory references are absolute, and all protection is disabled.

When Bit 21 is reset, the controller is still active in that it traps memory references to the locations assigned to its registers. Although only 68 bytes of address space (e.g., X'000300' through X'000343') are used as register addresses, the controller is set up to trap 256 bytes of address space (e.g., X'000300' through X'0003FF'). If the Processor references any location trapped, but not used by the controller, the results are undefined. When Bit 21 of the current Program Status Word is reset, the Processor can write into the segmentation registers. The Processor can read the least significant eight bits of the interrupt status register. Any Write instruction to the interrupt status register forces it to zero.

Bit 21 of the current Program Status Word must be reset to reference the memory access and protect controller registers. If Bit 21 is set, and a program makes a memory reference that is relocated to one of the locations reserved for the controller, data is read from or written into the corresponding memory locations.

BLOCK ADDRESS CONVENTION

Although the address coming into the controller is a 20 bit address in program space, and the relocated address produced by the controller is a 20 bit address in memory, internal to the controller, addresses are artifically broken down into two parts. These are the block address and the displacement within the block. An incoming address appears as:

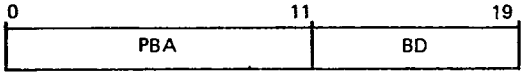


Figure 13. Program Address

in which PBA is the Program Block Address, the address of a 256 byte block in program space. BD is the Byte Displacement within the block. Similarly, the real address appears as:

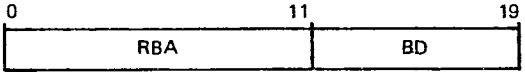


Figure 14. Real Address

in which RBA is the Real Block Address of a 256 byte block of real memory, and BD is the Byte Displacement within that block. The block address convention allows the controller to operate on 12 bit addresses. It also forces program segmentation to begin and to end on 256 byte boundaries.

SEGMENTATION REGISTERS

Each of the 16 segmentation registers is 32 bits wide, and is divided into three fields as shown in Figure 15.

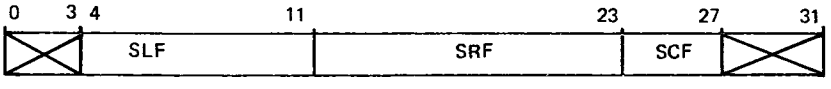


Figure 15. Segmentation Register

SLF is the Segment Limit Field, 8 bits. SRF is the Segment Relocation Field, 12 bits. SCF is the Segment Control Field, 4 bits. Bits 0:3 and 28:31 of the segmentation register are not used and must be zero.

SEGMENTATION REGISTER SELECTION

Bit 11 of the current Program Status Word controls the method of segmentation register selection. If this bit is zero, the memory access and protect controller uses the high order four bits (Bits 0:3) of the 20 bit program address to select the segmentation register (0:15) that is to be used in the relocation and protection process.

If Bit 11 of the current Program Status Word is one, the memory access and protect controller uses Bits 4:7 as the segmentation register select bits. (This state defines a Processor mode in which Bits 0:3 of the program address must be zero.) Thus to the memory access and protect controller, the program address appears as shown in Figure 16.

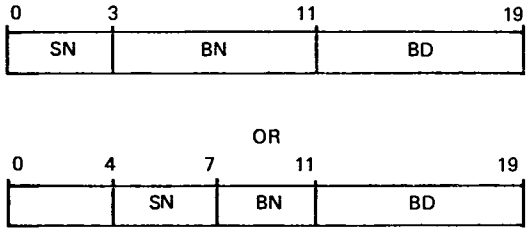


Figure 16. Program Addresses

In this figure, SN is the Segment Number, BN is the Block Number within that segment, and BD is the Byte Displacement within that block.

Each segmentation register contains a 8 bit limit field and a 12 bit relocation field as shown in Figure 17.

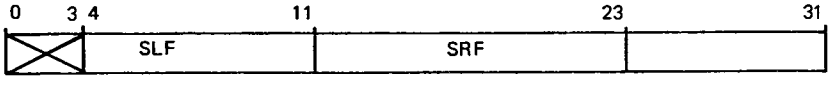


Figure 17. Limit and Relocation Fields

SLF is the Segment Limit field, SRF is the Segment Relocation field.

The relocation field contains the block address of the first 256 byte block of memory controlled by the register. Program addresses are relocated by adding the block number taken from the program address to the value contained in the relocation field. The byte displacement from the program address is then appended to the result to produce the 20 bit address in memory.

In a parallel with this process, the block number is compared with the value contained in the segment limit field. The limit field must contain the maximum program block address allowed for this segment. (Although segments have a fixed maximum length, 64KB or 4KB, the actual segment size may be any multiple of 256 bytes up to the limit.) If the comparison indicates that the block number is less than or equal to the limit, the relocated address is valid. If the block number is greater than the limit, the relocated address is invalid, and the memory access and protect controller interrupts the Processor. This type of protection isolates the program from all others in the system. No program is allowed to generate an address outside its assigned area.

The effect of an invalid address varies, depending on the synchronization between the memory access and protect controller and the Processor. As soon as the controller detects an invalid address, it attempts to notify the Processor. If the Processor cannot accept the interrupt immediately, the controller continues to operate. However, it does not allow memory to be changed. All writes are converted to reads until the Processor clears the interrupt.

FUNCTION OF THE CONTROL FIELD

In addition to the isolation protection provided by the limit field, the memory access and protect controller allows execute protection, two forms of write protection, and non-presence indications. These functions are controlled by Bits 24:27 of the segmentation register, the Segment Control Field is shown in Figure 18.

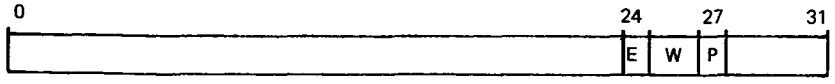


Figure 18. Segment Control Field

If Bit 24, E, is set, the area of memory described by the segmentation register is protected against execution, that is, instruction fetches from the area are not allowed. If this bit is reset, execution of instructions from the area is allowed. Any attempt to execute instructions from an execute protected area causes the memory access and protect controller to interrupt the Processor. The controller attempts to notify the Processor immediately upon the detection of an execute protection violation. If the Processor is not able to accept the interrupt and abort the operation, the controller continues to operate. However, all write operations are converted to read operations until the Processor has cleared the interrupt.

Bits 25 and 26, W, are encoded to provide two types of write protection. The interpretation of these bits is:

00	Unprotected
01	Write Protected
10	Write/Interrupt Protected
11	Write Protected (same as 01)

If the area controlled by the segmentation register is unprotected, all writes are allowed. If the area is write protected, the memory access and protect controller aborts all write operations and attempts to interrupt the Processor. If the Processor cannot accept the interrupt immediately, the controller continues to operate. However, all write operations are converted to read operation. If the area is write/interrupt protected, the memory access and protect controller allows the write operation to proceed, and then interrupts the Processor.

Bit 27, P, is the presence bit. If this bit is set, it means that the area of memory described by the segmentation register has been loaded with the correct program segment. If this bit is reset, it means that the program segment does not exist or has not yet been loaded. If a program address references a segmentation register in which this bit is reset, the controller interrupts the Processor.

INTERRUPTS

The memory access and protect controller can generate interrupts to the Processor only when relocation and protection are enabled (Bit 21 of the current Program Status Word set). It generates interrupts for the following reasons:

Invalid Address
Non-Present Address
Write Protect Violation
Write/Interrupt Condition
Execute Protect Violation

An invalid address interrupt occurs when the block number is greater than the value in the limit field.

A non-present address interrupt occurs when the program accesses a segmentation register in which the presence bit is zero.

A write protect violation interrupt occurs when the program attempts to write in an area controlled by a segmentation register in which the write protect bit is set.

A write/interrupt condition interrupt occurs when the program writes into an area controlled by a segmentation register in which the write/interrupt code appears.

An execute protect violation interrupt occurs when the program attempts to execute an instruction from an area controlled by a segmentation register in which the execute protect bit is set.

When the memory access and protect controller generates an interrupt, it sets a bit in the interrupt status register to indicate the reason for the interrupt. The interrupt status register is 32 bits wide. Bits 0:26 are undefined. The significance of the remaining bits is:

Bit	Meaning
27	Invalid Address
28	Non-Present Address
29	Write Protect Violation
30	Write/Interrupt Condition
31	Execute Protect Violation

The new Program Status Word for the memory access and protect controller interrupt handler should disable relocation and protection. The handler can then determine the cause of the interrupt by testing the memory location assigned to the interrupt status register. The first reference, either read or write, to the interrupt status register clears the interrupt condition within the controller.

APPENDIX 1

INDEX

APPENDIX 2

INSTRUCTION SUMMARY - ALPHABETICAL

INSTRUCTION	OP-CODE	MNEMONIC	PAGE NO.
Add	5A	A	67
Add	6A	AE	86
Add Halfword	4A	AH	68
Add Halfword Immediate	CA	AHI	68
Add Halfword to Memory	61	AHM	70
Add Immediate	FA	AI	67
Add Immdidate Short	26	AIS	67
Add to Bottom of List	65	ABL	54
Add to Memory	51	AM	67
Add to Top of List	64	ATL	54
Add Register	2A	AER	91
Add Register	0A	AR	67
AND	54	N	33
AND Halfword	44	NH	34
AND Halfword Immediate	C4	NHI	34
AND Immediate	F4	NI	33
AND Register	04	NR	33
Autoload	D5	AL	126
Branch and Link	41	BAL	61
Branch and Link Register	01	BALR	61
Branch on False Condition	43	BFC	59
Branch on False Condition Backward Short	22	BFBS	59
Branch on False Condition Forward Short	23	BFFS	59
Branch on False Condition Register	03	BFCR	59
Branch on Index High	C0	BXH	63
Branch on Index Low or Equal	C1	BXLE	61
Branch on True Condition	42	BTC	60
Branch on True Condition Backward Short	20	BTBS	60
Branch on True Condition Forward Short	21	BTFS	60
Branch on True Condition Register	02	BTCR	60
Compare	59	C	73
Compare	69	CE	94
Compare Halfword	49	CH	74
Compare Halfword Immediate	C9	CHI	74
Compare Immediate	F9	CI	73
Compare Register	29	CER	94
Compare Register	09	CR	73
Compare Logical	55	CL	30
Compare Logical Byte	D4	CLB	32
Compare Logical Halfword	45	CLH	31
Compare Logical Halfword Immediate	C5	CLHI	31
Compare Logical Immediate	F5	CLI	30
Compare Logical Register	05	CLR	30
Complement Bit	77	CBT	50
Cyclic Redundancy Check Modulo 12	5E	CRC12	52
Cyclic Redundancy Check Modulo 16	5F	CRC16	52

INSTRUCTION	OP-CODE	MNEMONIC	PAGE NO.
Divide	5D	D	77
Divide	6D	DE	96
Divide Halfword	4D	DH	78
Divide Halfword Register	0D	DHR	78
Divide Register	2D	DER	96
Divide Register	1D	DR	78
Exchange Byte Register	94	EXBR	25
Exchange Halfword Register	34	EXHR	24
Exchange Program Status Register	05	EPSR	109
Exclusive OR	57	X	37
Exlcusive OR Halfword	47	XH	38
Exculsive OR Halfword Immediate	C7	XHI	39
Exclusive OR Immediate	F7	XI	37
Exclusive OR Register	07	XR	37
Fix Register	2E	FXR	97
Float Register	2F	FLR	98
Load	58	L	18
Load	68	LE	88
Load Address	E6	LA	20
Load Byte	D3	LB	23
Load Byte Register	93	LBR	23
Load Complement Short	25	LCS	18
Load Halfword	48	LH	19
Load Halfword Immediate	C8	LHI	19
Load Halfword Logical	73	LHL	21
Load Immediate	F8	LI	18
Load Immediate Short	24	LIS	18
Load Multiple	D1	LM	22
Load Multiple	72	LME	89
Load Program Status Word	C2	LPSW	107
Load Program Status Word Register	18	LPSWR	108
Load Register	28	LER	92
Load Register	08	LR	18
Multiply	5C	M	76
Multiply	6C	ME	95
Multiply Halfword	4C	MH	77
Multiply Halfword Register	0C	MHR	77
Multiply Register	2C	MER	95
Multiply Register	1C	MR	76
OR	56	O	35
OR Halfword	46	OH	36
OR Halfword Immediate	C6	OHI	36
OR Immediate	F6	OI	35
OR Register	06	OR	35
Output Command	DE	OC	117
Output Command Register	9E	OCR	117
Read Block	D7	RB	120
Read Block Register	97	RBR	121
Read Data	DB	RD	118
Read Data Register	9B	RDR	118
Read Halfword	D9	RH	119
Read Halfword Register	99	RHR	119

INSTRUCTION	OP-CODE	MNEMONIC	PAGE NO.
Reset Bit	76	RBT	51
Remove from Bottom of List	67	RBL	55
Remove from Top of List	66	RTL	55
Sense Status	DD	SS	116
Sense Status Register	9D	SSR	116
Set Bit	75	SBT	49
Shift Left Arithmetic	EF	SLA	79
Shift Left Halfword Arithmetic	CF	SLHA	80
Shift Left Halfword Logical	CD	SLHL	43
Shift Left Halfword Logical Short	91	SLHLS	43
Shift Left Logical	ED	SLL	41
Shift Left Logical Short	11	SLLS	41
Shift Right Arithmetic	EE	SRA	81
Shift Right Halfword Arithmetic	CE	SRHA	82
Shift Right Halfword Logical	CC	SRHL	44
Shift Right Halfword Logical Short	90	SRHLS	44
Shift Right Logical	EC	SRL	42
Shift Right Logical Short	10	SRLS	42
Simulate Channel Program	E3	SCP	127
Simulate Interrupt	E2	SINT	110
Store	50	ST	26
Store	60	STE	90
Store Byte	D2	STB	29
Store Byte Register	92	STBR	29
Store Halfword	40	STH	27
Store Multiple	D0	STM	28
Store Multiple	71	STME	91
Subtract	5B	S	71
Subtract	6B	SE	93
Subtract Halfword	4B	SH	72
Subtract Halfword Immediate	CB	SHI	72
Subtract Immediate	FB	SI	71
Subtract Immediate Short	27	SIS	71
Subtract Register	0B	SR	71
Subtract Register	2B	SER	93
Supervisor Call	E1	SVC	111
Test Bit	74	TBT	48
Test Halfword Immediate	C3	THI	40
Test Immediate	F3	TI	39
Translate	E7	TLATE	53
Write Block	D6	WB	124
Write Block Register	96	WBR	125
Write Data	DA	WD	122
Write Data Register	9A	WDR	122
Write Halfword	D8	WH	123
Write Halfword Register	98	WHR	123

APPENDIX 3

INSTRUCTION SUMMARY - NUMERICAL

OP-CODE	MNEMONIC	INSTRUCTION	PAGE NO.
01	BALR	Branch and Link Register	61
02	BTCR	Branch on True Condition Register	60
03	BFCR	Branch on False Condition Register	59
04	NR	AND Register	33
05	CLR	Compare Logical Register	30
06	OR	OR Register	35
07	XR	Exclusive OR Register	37
08	LR	Load Register	18
09	CR	Compare Register	73
0A	AR	Add Register	67
0B	SR	Subtract Register	71
0C	MHR	Multiply Halfword Register	76
0D	DHR	Divide Halfword Register	78
10	SRLS	Shift Right Logical Short	42
11	SLLS	Shift Left Logical Short	41
12	CHVR	Convert to Halfword Value	83
13	BFCR	Branch on False Condition Register	59
18	LPSWR	Load Program Status Word Register	108
1C	MR	Multiply Register	75
1D	DR	Divide Register	78
20	BTBS	Branch on True Condition Backward Short	60
21	BTFS	Branch on True Condition Forward Short	60
22	BFBS	Branch on False Condition Backward Short	59
23	BFFS	Branch on False Condition Forward Short	59
24	LIS	Load Immediate Short	18
25	LCS	Load Complement Short	18
26	AIS	Add Immediate Short	67
27	SIS	Subtract Immediate Short	71
28	LER	Load Register	92
29	CER	Compare Register	94
2A	AER	Add Register	92
2B	SER	Subtract Register	93
2C	MER	Multiply Register	95
2D	DER	Divide Register	96
2E	FXR	Fix Register	97
2F	FLR	Float Register	98
30	BTBS	Branch on True Condition Backward Short	60
31	BTFS	Branch on True Condition Forward Short	60
32	BFBS	Branch on False Condition Backward Short	59
33	BFFS	Branch on False Condition Forward Short	59
34	EXHR	Exchange Halfword Register	24

OP-CODE	MNEMONIC	INSTRUCTION	PAGE NO.
40	STH	Store Halfword	27
41	BAL	Branch and Link	61
42	BTC	Branch on True Condition	60
43	BFC	Branch on False Condition	59
44	NH	AND Halfword	34
45	CLH	Compare Logical Halfword	31
46	OH	OR Halfword	36
47	XH	Exclusive OR Halfword	38
48	LH	Load Halfword	19
49	CH	Compare Halfword	74
4A	AH	Add Halfword	68
4B	SH	Subtract Halfword	72
4C	MH	Multiply Halfword	76
4D	DH	Divide Halfword	78
50	ST	Store	26
51	AM	Add to Memory	67
52	BTC	Branch on True Condition	60
53	BFC	Branch on False Condition	59
54	N	AND	33
55	CL	Compare Logical	30
56	O	OR	35
57	X	Exclusive OR	37
58	L	Load	18
59	C	Compare	73
5A	A	Add	67
5B	S	Subtract	71
5C	M	Multiply	75
5D	D	Divide	77
5E	CRC12	Cyclic Redundancy Check Modulo 12	52
5F	CRC16	Cyclic Redundancy Check Modulo 16	52
60	STE	Store	90
61	AHM	Add Halfword to Memory	72
64	ATL	Add to Top of List	54
65	ABL	Add to Bottom of List	54
66	RTL	Remove from Top of List	55
67	RBL	Remove from Bottom of List	55
68	LE	Load	88
69	CE	Compare	94
6A	AE	Add	87
6B	SE	Subtract	93
6C	ME	Multiply	95
6D	DE	Divide	76

OP-CODE	MNENONIC	INSTRUCTION	PAGE NO.
71	STME	Store Multiple	91
72	LME	Load Multiply	89
73	LHL	Load Halfword Logical	21
74	TBT	Test Bit	48
75	SBT	Set Bit	49
76	RBT	Reset Bit	51
77	CBT	Complement Bit	50
90	SRHLS	Shift Right Halfword Logical Short	44
91	SLHLS	Shift Left Halfword Logical Short	43
92	STBR	Store Byte Register	29
93	LBR	Load Byte Register	23
94	EXBR	Exchange Byte Register	25
95	EPSR	Exchange Program Status Word	109
96	WBR	Write Block Register	125
97	RBR	Read Block Register	121
98	WHR	Write Halfword Register	123
99	RHR	Read Halfword Register	119
9A	WDR	Write Data Register	122
9B	RDR	Read Data Register	118
9D	SSR	Sense Status Register	116
9E	OCR	Output Command Register	117
C0	BXH	Branch on Index High	63
C1	BXLE	Branch on Index Low or Equal	63
C2	LPSW	Load Program Status Word	107
C3	THI	Test Halfword Immediate	40
C4	NHI	AND Halfword Immediate	34
C5	CLHI	Compare Logical Halfword Immediate	31
C6	OHI	OR Halfword Immediate	36
C7	XHI	Exclusive OR Halfword Immediate	38
C8	LHI	Load Halfword Immediate	19
C9	CHI	Compare Halfword Immediate	74
CA	AHI	Add Halfword Immediate	68
CB	SHI	Subtract Halfword Immediate	72
CC	SRHL	Shift Right Halfword Logical	44
CD	SLHL	Shift Left Halfword Logical	43
CE	SRHA	Shift Right Halfword Arithmetic	82
CF	SLHA	Shift Left Halfword Arithmetic	80
D0	STM	Store Multiple	28
D1	LM	Load Multiple	22

OP-CODE	MNEMONIC	INSTRUCTION	PAGE NO.
D2	STB	Store Byte	29
D3	LB	Load Byte	32
D4	CLB	Compare Logical Byte	52
D5	AL	Autoload	126
D6	WB	Write Block	124
D7	RB	Read Block	120
D8	WH	Write Halfword	123
D9	RH	Read Halfword	119
DA	WD	Write Data	122
DB	RD	Read Data	118
DD	SS	Sense Status	116
DE	OC	Output Command	117
E0	TS	Test and Set	47
E1	SVC	Supervisor Call	111
E2	SINT	Simulate Interrupt	110
E3	SCP	Simulate Channel Program	127
E6	LA	Load Address	20
E7	TLATE	Translate	53
EA	RRL	Rotate Right Logical	46
EB	RLL	Rotate Left Logical	45
EC	SRL	Shift Right Logical	42
ED	SLL	Shift Left Logical	41
EE	SRA	Shift Right Arithmetic	81
EF	SLA	Shift Left Arithmetic	79
F3	TI	Test Immediate	39
F4	NI	AND Immediate	33
F5	CLI	Compare Logical Immediate	30
F6	OI	OR Immediate	35
F7	XI	Exclusive OR Immediate	37
F8	LI	Load Immediate	18
F9	CI	Compare Immediate	73
FA	AI	Add Immediate	67
FB	SI	Subtract Immediate	71

APPENDIX 4
EXTENDED BRANCH MNEMONICS

INSTRUCTION	OP CODE (M1)	MNEMONIC	OPERANDS
Branch on Carry	428	BC	A(X2)
Branch on Carry Register	028	BCR	R2
Branch on No Carry	438	BNC	A(X2)
Branch on No Carry Register	038	BNCR	R2
Branch on Equal	433	BE	A(X2)
Branch on Equal Register	033	BER	R2
Branch on Not Equal	423	BNE	A(X2)
Branch on Not Equal Register	023	BNER	R2
Branch on Low	428	BL	A(X2)
Branch on Low Register	028	BLR	R2
Branch on Not Low	438	BNL	A(X2)
Branch on Not Low Register	038	BNLR	R2
Branch on Minus	421	BM	A(X2)
Branch on Minus Register	021	BMR	R2
Branch on Not Minus	431	BNM	A(X2)
Branch on Not Minus Register	031	BNMR	R2
Branch on Plus	422	BP	A(X2)
Branch on Plus Register	022	BPR	R2
Branch on Not Plus	432	BNP	A(X2)
Branch on Not Plus Register	032	BNPR	R2
Branch on Overflow	424	BO	A(X2)
Branch on Overflow Register	024	BOR	R2
Branch Unconditional	430	B	A(X2)
Branch Unconditional Register	030	BR	R2
Branch on Zero	433	BZ	A(X2)
Branch on Zero Register	033	BZR	R2
Branch on Not Zero	423	BNZ	A(X2)
Branch on Not Zero Register	023	BNZR	R2
No Operation	420	NOP	
No Operation Register	020	NOPR	
Branch on Carry Short	208	BCS	A (Backward Reference)
	218	BCS	A (Forward Reference)
Branch on No Carry Short	228	BNCS	A (Backward Reference)
	238	BNCS	A (Forward Reference)
Branch on Equal Short	223	BES	A (Backward Reference)
	233	BES	A (Forward Reference)
Branch on Not Equal Short	203	BNES	A (Backward Reference)
	213	BNES	A (Forward Reference)
Branch on Low Short	208	BLS	A (Backward Reference)
	218	BLS	A (Forward Reference)
Branch on Not Low Short	228	BNLS	A (Backward Reference)
	238	BNLS	A (Forward Reference)

INSTRUCTION	OP CODE (M1)	MNEMONIC	OPERANDS
Branch on Minus Short	201	BMS	A (Backward Reference)
	211	BMS	A (Forward Reference)
Branch on Not Minus Short	221	BNMS	A (Backward Reference)
	231	BNMS	A (Forward Reference)
Branch on Plus Short	202	BPS	A (Backward Reference)
	212	BPS	A (Forward Reference)
Branch on Not Plus Short	222	BNPS	A (Backward Reference)
	232	BNPS	A (Forward Reference)
Branch on Overflow Short	204	BOS	A (Backward Reference)
	214	BOS	A (Forward Reference)
Branch Unconditional Short	220	BS	A (Backward Reference)
	230	BS	A (Forward Reference)
Branch on Zero Short	223	BNA	A (Backward Reference)
	233	BZS	A (Forward Reference)
Branch on Not Zero Short	203	BNZS	A (Backward Reference)
	213	BNZS	A (Forward Reference)

APPENDIX 5
ARITHMETIC REFERENCES

TABLE OF POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5

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