Table 1-9. Interrupts

Interrupt	PSW Control Bit
External	1
Machine Malfunction	2
Fixed Point Divide Fault	3
Automatic I/O Service	4
Floating-Point Arithmetic Fault	5
Channel Termination	6
Protect Mode	7
Illegal Instruction	Cannot be disabled
Channel Termination Queue Overflow	Cannot be disabled
Supervisor Call	Cannot be disabled

Table 1-10. Interrupt Response Times (Microseconds)

Interrupt Occurrence / Machine Activity	External Interrupt	Machine Malf. Interrupt	Fixed Point Div. Fault Interrupt	Floating Poin Divide Fault Interrupt
* Any instruction except Load Multiple, Store Multiple, Read Block, Write Block				
• Channel termination disabled	15.0	15.8	**12.2	**44.1
• Channel termination enabled	16.8	17.6	**14.0	**45.9
* Load Multiple, Store Multiple	21.2	22.0	----	----
* Read Block, Write Block	***	***		

* No interleaved data channel activity. No automatic I/O service activity.

** Includes execution of the instruction itself.

*** Maximum response time is 170,400 μsec at maximum data rate.

Table 1-11. Automatic I/O Service Times (Microseconds)

	NOP	NULL	DMT	Output Command	Read	Write	Bad Status	Immediate Interrupt
Base	6.8	8.8	8.4	10.4	12.2+1.8n	12.4+1.6n	19.8	6.2
Initiate	---	1.6	1.6	---	5.0	5.0	---	---
Term Char (no match)	---	---	---	---	1.6	1.6	---	---
Term Char (match)	---	---	---	---	4.2	4.2	---	---
Queue (top)	---	7.2	7.2	---	7.2	7.2	---	---
Queue (bottom)	---	7.0	7.0	---	7.0	7.0	---	---
Chain	---	1.4	1.4	---	1.4	1.4	---	---
Continue	---	-5.0+	-5.0+	---	-5.0+	5.0+	---	---
Queue Service Interrupt	2.6	2.6	2.6	2.6	2.6	2.6	2.6	---

NOTES: 1. On read and write, n = number of bytes per interrupt.

2. On continue, subtract 5.0 µsec and add the service time of the continute action.

3. The Initiate row represents the useless case where Initiate=1, and Output Command = 0 (zero).

4. The --- means does not apply.

Table 1-12. New and Old PSW Locations

Interrupt	Hexadecimal WORAM Locations	
	Old PSW Location	New PSW Location
External	40-43	44-47
Machine Malfunction	38-3B	3C-3F
Fixed Point Divide Fault	48-4B	4C-4F
Automatic I/O Service	Defined by Interrupt Pointer Table	Defined by Interrupt Pointer Table
Floating-Point Arithmetic Fault	28-2B	2C-2F
Channel Termination	82-85	86-89
Protect Mode	30-33	34-37
Illegal Instruction	30-33	34-37
Channel Termination Queue Overflow	8C-8F	90-93
Supervisor Call	96-99	9A-BB

During automatic I/O operations, interrupt response times are increased by the total automatic I/O service time (including continue) minus 0.8 microseconds. Automatic I/O service times are shown in Table 1-11.

During interleaved data channel operations, interrupt response times are increased by 0.4 + 3.8N microseconds, where N is the number of words transferred.

In addition, all interrupt response times are subject to degradation by memory refresh and DMA activity.

2. Control of Interrupts. Each type of interrupt is enabled or disabled by their associated Enable/Disable PSW bit as shown in Table 1-9. Interrupts without a controlling PSW bit are always enabled. Disabled interrupts are not queued. An Enable/Disable PSW bit is in the enable state when set to one. The disable state corresponds to Enable/Disable PSW bit being reset to zero. Immediately after setting an Enable/Disable bit in the Program Status Register, a pending interrupt of the corresponding type causes the interrupt procedure to occur. The Enable/Disable bits of the Program Status Register may be set or reset by any of the following methods.

a. Execution of a Load Program Status Word instruction, or Exchange Program Status instruction.

b. Exchange of the current PSW in Program Status Register with a new PSW stored in a fixed location in memory. This may occur as the result of executing a Simulate Interrupt instruction or Supervisor Call instruction, receiving an external interrupt, or generation of an internal interrupt.

3. Source and Occurrence of Interrupts. An External interrupt condition is created by a controller on the I/O Mux Bus setting the Attention signal. This condition may occur any time at random except when all controllers have been commanded to disable. If Bit 1 of the Program Status Register is set, the Processor responds to an external interrupt upon completion of the current instruction being executed. If, however, the instruction being executed has an execution time in excess of 16 microseconds (except possibly WB or RB), the execution of the instruction is aborted in such a way that the instruction is reinitiated when processing of the interrupted routine is resumed.

The machine malfunction interrupt occurs following execution of the current instruction. The Fixed Point Divide Fault interrupt occurs during execution of the current instruction. The Automatic I/O service interrupt occurs following execution of the current instruction. The Floating-Point Arithmetic Fault interrupt occurs during execution of the current instruction. The Channel Termination interrupt occurs either following the execution of the current instruction or during a Load Program Status Word or Exchange Program Status instruction. The Illegal Instruction and Protect Mode interrupt occurs prior to instruction execution. The Channel Termination Queue Overflow interrupt occurs following execution of the current instruction. The Supervisor Call Interrupt occurs as part of the execution of the Supervisor Call instruction.

4. PSW Exchange. The interrupt procedure is based on the concepts of Old, Current, and New Program Status Words. The Current PSW is contained in the Program Status Register which defines the operating status of the machine. When this status must be interrupted, the current PSW becomes an Old PSW by storing the contents of the Program Status Register in a memory location dedicated to the type of interrupt that has occurred. The New PSW becomes the Current PSW by being loaded from a dedicated location in the Program Status Register. The status portion of the Current PSW contains the operating status for the interrupt service routine. The Program Status Word exchange procedure does not change the contents of the New PSW location.

5. New and Old PSW Locations. Each interrupt type has at least one New PSW location and one Old PSW location associated with it, with the exception that Protect Mode and Illegal Instruction interrupts share the same pair of locations. The New/Old PSW locations associated with each interrupt type are shown in Table 1-12.

(b) Internal Interrupts. The Processor is capable of generating the following six internal interrupts:

1. Fixed-Point Divide Fault Interrupt. The Fixed-Point Divide Fault Interrupt, enabled by Bit 3 of the Program Status Word, is indicative of division by zero or quotient overflow. Quotient overflow is defined as quotient magnitude greater than $2^{15}-1$, for a halfword quotient. The interrupt takes place before modification of the operand registers. After a Fixed-Point Divide Fault Interrupt, the Old PSW Location Counter points to the next instruction following the Divide instruction.

2. Floating-Point Arithmetic Fault Interrupt. The Floating Point Arithmetic Fault Interrupt enabled by Bit 5 of the Current PSW, occurs on exponent overflow or underflow as well as on division by zero. In the case of division by zero, the interrupt takes place prior to alteration of the operand register. An exponent overflow sets the results to ±X'7FFF FFFF'. An exponent underflow sets the results of X'0000 0000'. The Location Counter of the Old PSW points to the next instruction.

3. Machine Malfuntion Interrupt. Bit 2 of the Current Program Status Word controls the Machine Malfunction Interrupt. This error occurs on a primary power fail, a memory parity error, and during the restart process following a power down.

a. Parity Error. If a Parity Error condition occurs as specified in Paragraph 1-4b(7)(b), and if Bit 2 of the current PSW is set, the Current Program Status Word is stored as the Machine Malfunction Old PSW location, and the Current PSW is loaded from the Machine Malfunction New PSW location. The Condition code field of the Current PSW is then adjusted by setting the G flag (PSW 14) if the parity error occurred on instruction read, or setting the V flag (PSW 13) if the error occurred on an operand read. The following should be noted: it is not possible to guarantee programmed recovery from a parity error; and the Condition Code field of the Machine Malfunction New PSW location in memory must be zero.

<u>b.</u> Advanced Power Failure. If Bit 2 of the PSW is set, the setting of the Advance Power Failure signal by the Power Fault Detect function causes a Machine Malfunction interrupt to occur. After the PSW exchange, the L flag (PSW 15) of the current PSW is set. The Processor remains operational for 1.6 milliseconds and then initiates a hardware power down sequence as specified in Paragraph 1-4b(9)(f).

<u>c.</u> Power Restore. The resetting of the Advance Power Failure signal when Processor power is restored causes the general registers to be reloaded and the Program Status Register to be restored as specified in Paragraph 2-3f(5)(b). If Bit 2 of PSW loaded in the Program Status Register is set, the Processor exchanges the PSW from the Machine Malfuntion location.

<u>4.</u> Illegal Instruction Interrupt. The Illegal Instruction interrupt occurs when the Processor attempts to execute an instruction which is not within its valid instruction repertoire. Execution of the invalid instruction is immediately terminated and the Program Counter is not altered. The old PSW stored as a result of an Illegal Instruction interrupt points to the address of the Illegal Instruction.

<u>5.</u> Protect Mode Violation Interrupt. The Protect Mode Violation interrupt is enabled when Bit 7 of the Current PSW is set, which puts the Processor in the Protect Mode. The interrupt occurs, in this mode, when an attempt is made to execute a Privileged instruction. Privileged instructions are all I/O instructions and System Control instructions: Load Program Status Word, Exchange Program Status, and Simulate Interrupt. When such an instruction is attempted in this mode, the instruction is not executed, and the Illegal Instruction Interrupt procedure takes place, as described above. The Location Counter does not increment, so that the Old PSW points to the Privileged instruction that caused the interrupt.

<u>6.</u> Supervisor Call (SVC) Interrupt. This interrupt always occurs as the result of executing an SVC instruction, which is used to communicate between running programs and operating systems. When an SVC instruction is executed, the following action shall take place:

<u>a.</u> The current PSW is stored at the Supervisor Call Old PSW location, Location X'0098'.

<u>b.</u> The effective address from the SVC instruction is stored at the Supervisor Call argument pointer, Location X'0094'.

<u>c.</u> The status portion of the Current PSW is loaded from the Supervisor Call New PSW Status location, Location X'009A'.

<u>d.</u> The Current Location Counter is loaded from one of the Supervisor Call New PSW Location Counter locations.

(c) Input/Output Control Interrupts. The Processor has two classes of interrupts directly related to peripheral device handling. These are the External Interrupt and the Immediate Interrupt. Two other classes, the Channel Termination Interrupt and the Channel Queue Overflow Interrupt can occur upon termination of an Automatic I/O channel sequence. PSW Bits 1 and 4, in combination, shall control the External and Immediate Interrupts. If individually enabled by the program, a periphal device is allowed to request Processor service when the device itself is ready to transfer data via an interrupt. The Processor responds to this signal in the following ways depending on the setting of bits 1 and 4 in the Program Status Word.

<u>1.</u> I/O Interrupt Lock-Out. If Bit 1 of the Program Status Register is reset, I/O Device Interrupt signals are ignored, and the requests are not queued. It is up to the interrupting device to keep its request up until PSW Bit 1 is set and the signal is acknowledged.

<u>2.</u> External Interrupt. When Bit 1 of the Program Status Register is set, and Bit 4 is reset, the Input/Output Multiplexer Bus "Attention" signal causes the Processor to store the current contents of the Program Status Register at the External interrupt Old PSW location into the Program Status Register. The resulting software service routine uses the Acknowledge Interrupt instruction to identify the interrupting input/output device and then take appropriate action.

<u>3.</u> Immediate Interrupt. When both Bit 1 and Bit 4 of the Program Status Register are set, the Input/Output Multiplexer Bus "Attention" signal causes the Processor to automatically perform an Acknowledge Interrupt operation causing the interrupting input/output device to transmit its device address byte to the Processor. The Processor uses the device address byte to index into an Interrupt Pointer Table in memory locations DO to 2CF (hexadecimal). The Service Pointer word thus obtained is the address of the Old PSW location for the interrupting device, or a Channel Command Word for a channel I/O operation. If Bit 14 of the Service Pointer is reset, the Processor stores the current contents of the Program Status Register into the Old PSW location. The contents of the New PSW location, addressed by the Service Pointer word plus four, are then loaded into Bits 0-15 of the Program Status Register. Finally, the Service Pointer word plus six is loaded into Bits 16-31 of the Program Status Register (Program Counter portion). In this way, the Processor automatically enters the specific software service routine associated with the interrupting device. If Bit 15 of the Service Pointer is set, the address contained is that of a Channel Command Word implying that Automatic I/O Channel service is required.

4. Automatic I/O Channel Termination Interrupt. The termination of an Automatic I/O Channel operation may result in the storing of a termination pointer in the circular list located at the address specified by the Queue Pointer location. If, at this time, Bit 6 of the Current PSW is set, the Current PSW is stored at the Channel Termination Old PSW location, and the Program Status Register loaded from the Channel Termination New PSW location. In this way, the control software is notified of the completion of a channel I/O operation. Whenever the Processor executes a Load Program Status Word instruction or an Exchange Program Status instruction, Bit 6 of the newly loaded PSW is examined. If Bit 6 of the loaded PSW is set, and there is an entry in the queue, this interrupt occurs.

5. Channel Termination Queue Overflow Interrupt. If the Processor attempts to enter a Channel I/O Termination Pointer in the Termination Queue and the queue is already full, it stores the termination pointer at Location X'008A', the Overflow Termination Pointer location; stores the Current PSW in Location X'008C', the Queue Overflow Old PSW location; and loads the Program Status Register from Location X'0090', the Queue Overflow New PSW location. This action allows the software to clear out the queue before any channel I/O terminations are lost. This interrupt is always enabled.

(d) I/O Device Interrupt Priorities. The I/O device priorities are established in the following manner. Following an interrupt, the processor acknowledges the interrupt by issuing the signal TACK. This signal is sent to the highest priority device controller. If this device requested the interrupt, this signal serves to acknowledge receipt of the interrupt and is not passed to the other controllers. If, however, the interrupt did not originate with the device, the acknowledge is passed on to the next highest priority I/O device, which shall treat it in like manner. The acknowledge is passed down the line until it gets to the device that initially requested the interrupt. The device then identifies itself by placing its address on the I/O Mux Bus.

(9) Operational Modes and Control. A diagram and explanation of the 1116 front panel is shown in Figure 1-10. The control of the item and resolution of contention for resources are in accordance with the following subordinate paragraphs. The item is in one of the following modes at all times:

a. Power-off
b. Start-up
c. Program load from I/O mux bus
d. Supervisor mode (operational)
e. Wait state
f. Protect mode (operational)
g. Power-down
h. Stand-by
i. Restart

The transition from one mode to another is as shown in Figure 1-8.

(a) Power-Off. A power-off condition only occurs when one or more of the voltage levels of the memory keep-alive power is below the specified limits. In the power-off condition the contents of memory are no longer refreshed and the execution of instructions is halted. Note the contents of read/write memory modules will not be retained in the power-off condition. The processor is held in a power-off condition by the Power Fault Detect function from the instant power is applied until the voltage levels reach tolerance. When all voltages have reached tolerance, the Processor transitions into the Start-up mode; the Memory Interface Function commences refreshing the dynamic Memory Modules; and the Memory Operational Indicator signal is reset.

(b) Start-Up and Program Load. When all power supply voltages are in tolerance and the Memory Operational Indicator signal is reset, the Processor performs the Start-up sequence pictured in Figure 1-9. If the Processor receives a program load signal, the Processor performs program load from the I/O Mux Bus. After completing the loading sequence the Processor shall begin program execution.

(c) Supervisor Mode. If Bit 7 of the current PSW is reset, the Processor is in the Supervisor Mode. The execution of any legal instruction is enabled in the Supervisor Mode including privileged instructions.

(d) Protect Mode. If Bit 7 of the current PSW is set, the Processor is in the Protect Mode. In this mode the execution of Priviledged instructions are disabled. The Privileged instructions include all I/O instructions, and most of the System Control instructions. Attempts to execute a Privileged instruction in the Protect Mode causes a Protect Mode Violation interrupt as specified in Paragraph 1-4b(8)(b)5.

(e) Wait State. Replacing the current PSW with one in which Bit 0 is set puts the Processor in the wait state. When the Processor is in the wait state, program execution is halted. However, the Processor still responds to machine malfunction, external, and immediate interrupts, if they are enabled. Automatic I/O channel operations are also capable of temporarily forcing the Processor out of the wait state. With all interrupts disabled, only operator intervention from the Processor Maintenance Panel or a Power-Off condition can force the Processor out of the wait state.

(f) Power Down. Whenever any of the power supply voltages becomes out of tolerance, the Processor begins a power down sequence. 1.6 milliseconds are allocated for continued software execution, and then the Processor begins a systematic power down sequence in which the contents of the 16 general purpose registers, PSW, and Location Counter shall be stored in memory. The Processor then waits for the power to go down.

(g) Stand-By. Shutting the main processor power supply down while leaving the memory supplies operational places the Processor in the Stand-By mode. In this mode, normal processing is suspended but memory refresh is operational and the contents of memory are preserved.

(h) Restart. During the power up sequence the Processor determines whether or not the memory has remained operational (previous memory contents not lost). If the memory is still valid and the restored PSW has bit 2 set, the current contents of the Program Status Register are placed in the Machine Malfunction Old PSW location X'0038'-003B', the contents of the Machine Malfunction New PSW location X'003C-003F' are placed in the Program Status Register, and processing resumes at the location indicated by the new contents of the Program Status Register. If PSW Bit 2 is not set, and the Processor is in the Run mode, the Processor resumes program execution where it was interrupted by the power down sequence.

(10) Clock Oscillator. The clock oscillator is a crystal controlled oscillator with an operating frequency of 40.000 MHz, an accuracy of 0.01%, and the capability of being within accuracy limits within 0.1 sec of power application.

(a) REAL-TIME Clock. The Real-Time clock increments every millisecond. It resets every 200 days.

(b) ELAPSED TIME Clock. The Elapsed Time clock decrements every millisecond. When it reaches -1 millisecond an ETC interrupt is generated.

(1) Instruction Lengths. The Processor handles both halfword and fullword instruction formats. A 16-bit halfword format is used for Register-to-Register and Short Format instructions. The Short Format instructions are used to manipulate small quantities or execute short branches relative to the present Location Counter. A 32-bit fullword format is used for the Register and Indexed Memory and the Register Immediate formats. The specific formats are organized as shown in Figure 2-34.

```
       | 0        7 | 8      11 | 12      15 |
       |____________|___________|____________|
[RR]   |     OP     |    R1*    |     R2     |
       |____________|___________|____________|
```

REGISTER TO REGISTER

```
       | 0        7 | 8      11 | 12      15 |
       |____________|___________|____________|
[SF]   |     OP     |    R1*    |     N      |
       |____________|___________|____________|
```

SHORT FORMAT

```
       | 0        7 | 8      11 | 12      15 | 16                31 |
       |____________|___________|____________|_______________________|
[RX]   |     OP     |    R1*    |     X2     |          A2           |
       |____________|___________|____________|_______________________|
```

REGISTER AND INDEXED MEMORY

```
       | 0        7 | 8      11 | 12      15 | 16                31 |
       |____________|___________|____________|_______________________|
[R1]   |     OP     |    R1     |     X2     |          I2           |
       |____________|___________|____________|_______________________|
```

REGISTER IMMEDIATE

*REPLACED BY M1 FOR BRANCH INSTRUCTIONS

Figure 2-34. Instruction Word Formats

(2) Instruction Fields. The instruction fields of the four instruction formats shown in Figure 2-34 are defined and function as follows:

(a) OP Field. The 8-bit OP field in all formats specifies the machine operation to be performed. Operation codes are represented as two hexadecimal characters.

(b) R1 Field. The 4-bit R1 field in the instruction formats specifies the general register address of the first operand.

(c) R2 Field. The 4-bit R2 field in the instruction formats specifies the general register address of the second operand.

(d) N Field. The 4-bit Data field of the SF instructions supplies data in the case of Fixed-Point Arithmetic instructions, or a displacement from the current Location Counter in the case of Branch instructions.

(e) X2 Field. A nonzero X2 field in the RX and RI formats specifies a General Register whose contents are used as an index value. The index value (X2) may be positive or negative. If X2 is zero, no address modification takes place. General Registers 1 thru 15 are optionally used for indexing, but General Register 0 is never used for indexing.

(f) A2 Field. The 16-bit Address field specifies a memory address in the RX format.

(g) I2 Field. The 16-bit Immediate field contains a value (data) to be used as an immediate operand in the RI format.

(h) M1 Field (MASK). The 4-bit M1 field in the Branch instructions is used to test the Condition Code in the Program Status Word.

(3) Format Usage. The first and second operand designations for each instruction format conforms with Table 2-2, and the following rules for using each instruction format:

(a) RR Format. The RR instructions are used for operations between registers. The first operand is the contents of the register specified by the R1 field of the instruction word. The second operand is the contents of the register specified by the R2 field.

(b) SF Format. The SF instructions are used for: short immediates, in which the N field specifies a 4-bit data value; short shifts, in which the N field specifies the shift count; and short branches, in which the N field specifies displacement (in halfwords) from the current instruction address. The register specified by R1 contains the first operand.

(c) RX Format. The RX instructions are used for operations between register and memory with the option of indexing. The first operand is the contents of the register specified by the R1 field of the instruction word. The second operand is the contents of the memory location specified by the A2 field of the instruction word, or the sum of the A2 field and the contents of the General Register specified by the X2 field if indexing is specified. The A2 field may contain a maximum value of 65,535.

(d) RI Format. In the RI instructions, the first operand is the contents of the General Register specified by the R1 field of the instruction word. The second operand is the number contained in the I2 field of the instruction word, or the sum of the I2 field and the contents of the General Register specified by the X2 field if indexing is specified. The second operand of an RI instruction specifies the number of bit positions in Shift instructions, or forms the second operand in Immediate instructions.

(e) Exceptions. The explanation of each instruction in Appendix I has precedence over these rules and will result in some exceptions to the first operand/second operand nomenclature used above. For example, with Branch On Condition instructions, the R1 field of the instruction is a 4-bit mask (M1) which is ANDed with the Condition Code in the Current PSW. For all Input/Output instructions, the contents of the register specified by R1 specifies the device number for the I/O operation. For the Supervisor Call instruction, the R1 field specifies 1 out of 16 possible types of supervisor calls. With the Load Program Status Word (LPSW), Simulate Interrupt (SINT) and Auto Load (AL) instructions, the R1 field must be zero.

(f) General Register Restrictions. Each general register functions as an accumulator or index register in all arithmetic and logical operations with the following restrictions:

1. General register 0 is not used as an index register. A zero entry in the X2 field of the RX and RI instruction formats shall indicate that no indexing is to take place.

2. The R1 and/or R2 field must specify an even numbered general register for all fullword fixed-point instructions and halfword fixed-point multiply and divide instructions.

3. For Branch or Index instructions, the R1 field specifies the first of three consecutive general registers, and the value of the R1, therefore, should be equal to or less than 13.

4. For Floating-Point instructions the R1 field and R2 field for register to register operations must be an even value, and specify one of the Floating-Point Registers rather than one of the General Registers.

5. With any RR type instruction, the R1 field and the R2 field may specify the same register, but special attention should be given to note what the instruction will do. For example, with the EPSR instruction, if the R1 field equals the R2 field, the program status is stored in a General Register, but the program status is unchanged.

6. In the Conditional Branch instructions, the R1 field does not specify a register. Instead, it contains a mask value which is tested with the condition code.

e. Instruction Set and Execution Times. The basic HMP-1116 contains and is capable of executing an instruction set of 123 instructions as specified in Table 2-3. All operation codes are represented in hexadecimal notation. The execution time for each instruction is equal to the time specified in the corresponding table less degradation due to memory refresh for a volatile memory configuration. Overall timing degradation due to memory refresh does not exceed 4.05 percent.

Table 2-4 contains the extended branch mnemonics which are recognized by CAL (Common Assembler Language), the assembler used for the HMP-1116. CAL recognizes the extended branch mnemonic, selects the appropriate branch instruction from table 2-3 and inserts the M1 mask value in the R1 field.

Table 2-2. Designations for First and Second Operands

First Operand:	The contents of the register specified by the R1 field (R1)	RR, RX, RI, and SF
	The mask value M1 in the R1 field	RR, RX and SF branch on condition
	The actual value of the R1 field	SVC
Second Operand:	The contents of the register specified by the R2 field (R2)	RR
	The contents of the address derived by adding the A2 field and the contents of the General Register specified by the X2 field [(A2) + (X2)]	RX
	The I2 field plus the contents of the General Register specified by the X2 field. I2 + (X2)	RI
	The actual value of the N field	SF

Table 2-3. HMP-1116 Instruction Repertoire

Instruction	Data (Bits)	Mnemonic RR/SF	Mnemonic RI	Mnemonic RX	OP Code (Hex) RR/SF	OP Code (Hex) RI	OP Code (Hex) RX	Execution Times (sec) RR/SF	RI	RI Indexed	RX	Comments
Fixed-Point Load/Store												
Load Immediate Short	4	LIS			24			0.8				
Load Complement Short	4	LCS			25			0.8				
Load Halfword	16	LHR	LHI	LH	08	C8	48	0.8	1.2	1.4	1.8	
Load from Bank 0	16			LH0			74				2.0	
Load from Bank 1	16			LH1			75				2.0	
Load from Bank 2	16			LH2			76				2.0	
Load from Bank 3	16			LH3			77				2.0	
Load Multiple	16			LM			D1				2.6+0.8n	n=No. of Regs
Store Halfword	16			STH			40				2.0	
Store in Bank 0	16			STH0			F8				2.2	
Store in Bank 1	16			STH1			F9				2.2	
Store in Bank 2	16			STH2			FA				2.2	
Store in Bank 3	16			STH3			FB				2.2	
Store Multiple	16			STM			D0				2.4+0.8n	n=No. of Regs
Fixed-Point Arithmetic												
Add Immediate Short	4	AIS			26			0.8				
Add Halfword	16	AHR	AHI	AH	0A	CA	4A	0.8	1.2	1.4	1.8	
Add Halfway to Memory	16			AHM			61				2.4	
Add with Carry Halfword	16	ACHR		ACH	0E		4E	1.0			2.0	
Subtract Immediate Short	4	SIS			27			0.8				
Subtract Halfword	16	SHR	SHI	SH	0B	CB	4B	0.8	1.2	1.4	1.8	
Subtract with Carry Halfword	16	SCHR		SCH	0F		4F	1.0			2.0	
Multiply Halfword	16	MHR		MH	0C		4C	5.0			5.8	
Multiply Halfword Unsigned	16	MHUR		MHU	9C		DC	4.8			5.6	
Divide Halfword	16	DHR		DH	0D		4D	7.4/7.6/7.8/8.0			8.2/8.4/8.6/8.8	++/+-/-+/--
Logical and Compare												
AND Halfword	16	NHR	NHI	NH	04	C4	44	0.8	1.2	1.4	1.8	
OR Halfword	16	OHR	OHI	OH	06	C6	46	0.8	1.2	1.4	1.8	
Exclusive or Halfword	16	XHR	XHI	XH	07	C7	47	0.8	1.2	1.4	1.8	
Test Halfword Immediate	16		THI			C3			1.2	1.4		
Compare Halfword	16	CHR	CHI	CH	09	C9	49	1.4	1.6	1.8	2.2	
Compare Logical Halfword	16	CLHR	CLHI	CLH	05	C5	45	0.8	1.2	1.4	1.8	

Instruction	Mnemonic				OP Code (Hex)			Execution Times (sec)				
	Data (Bits)	RR/SF	RI	RX	RR/SF	RI	RX	RR/SF	RI	RI Indexed	RX	Comments
Byte Handling												
Load Byte	8	LBR		LB	93		D3	0.8			2.0	
Store Byte	8	STBR		STB	92		D2	1.2			2.4	
Exchange Byte	8	EXBR			94			0.8				
Compare Logical Byte	8			CLB			D4				2.0	
Shift and Rotate												
Shift Left Logical Short	16	SLLS			91			1.4 + 0.2 (n-1)				n=No. of Shifts
Shift Left Halfword Logical	16		SLHL			CD			1.8 + 0.2 (n-1)	2.0 + 0.2 (n-1)		n=No. of Shifts
Shift Left Fullword Logical	32		SLL			ED			2.0 + 0.2 (n-1)	2.2 + 0.2 (n-1)		n=No. of Shifts
Shift Right Logical Short	16	SRLS			90			1.4 + 0.2 (n-1)				n=No. of Shifts
Shift Right Halfword Logical	16		SRHL				CC		1.8 + 0.2 (n-1)	2.0 + 0.2 (n-1)		n=No. of Shifts
Shift Right Fullword Logical	32		SRL			EC			2.0 + 0.2 (n-1)	2.2 + 0.2 (n-1)		n=No. of Shifts
Rotate Left Fullword Logical	32		RLL			EB			2.0 + 0.2 (n-1)	2.2 + 0.2 (n-1)		n=No. of Shifts
Rotate Right Fullword Logical	32		RRL			EA			2.0 + 0.2 (n-1)	2.2 + 0.2 (n-1)		n=No. of Shifts
Shift Left Halfword Arithmetic	16		SLHA			CF			2.6 + 0.2 (n-1)	2.8 + 0.2 (n-1)		n=No. of Shifts
Shift Left Fullword Arithmetic	32		SLA			EF			2.6 + 0.2 (n-1)	2.8 + 0.2 (n-1)		
Shift Right Halfword Arithmetic	16		SRHA			CE			2.2 + 0.2 (n-1)	2.4 + 0.2 (n-1)		n=No. of Shifts
Shift Right Fullword Arithmetic	32		SRA			EE			2.4 + 0.2 (n-1)	2.6 + 0.2 (n-1)		n=No. of Shifts

Table 2-3. HMP-1116 Instruction Repertoire (Cont'd)

Instruction	Mnemonic				OP Code (Hex)			Execution Times (sec)				
	Data (Bits)	RR/SF	RI	RX	RR/SF	RI	RX	RR/SF	RI	RI Indexed	RX	Comments
Branch*												
Branch on True Backward Short	-	BTBS			20			1.2/1.6				No BR/BR
Branch on True Forward Short	-	BTFS			21			1.2/1.6				No BR/BR
Branch on True Condition	-	BTCR		BTC	02		42	1.2/1.4			1.6/1.8;	No Index(No BR/BR);
											1.8/2.0	Index (No BR/BR)
Branch on False Backward Short	-	BFBS			22			1.2/1.6				No BR/BR
Branch on False Forward Short	-	BFFS			23			1.2/1.6				No BR/BR
Branch on False Condition	-	BFCR		BFC	03		43	1.2/1.4			1.6/1.8;	No Index(No BR/BR);
											1.8/2.0	Index (No BR/BR)
Branch on Index High	-		BXH			C0			3.0/3.2	3.2/3.4		No BR/BR
Branch on Index Low or Equal	-		BXLE			C1			3.0/3.2	3.2/3.4		No BR/BR
Branch and Link	-	BALR		BAL	01		41	1.2			1.4/1.6	No Index/Index
Floating-Point												
Floating-Point Load	32	LER		LE	28		68	11.5			11.5	Average Time
Floating-Point Store	32			STE			60				4.2	Average Time
Floating-Point Add	32	AER		AE	3A		6A	16.4			16.4	Average Time
Floating-Point Subtract	32	SER		SE	2B		6B	16.8			16.8	Average Time
Floating-Point Compare	32	CER		CE	29		69	5.7			5.7	Average Time
Floating-Point Multiply	32	MER		ME	2C		6C	23.6			23.6	Average Time
Floating-Point Divide	32	DER		DE	2D		6D	40.7			40.7	Average Time
System Control												
Load Program Status Word	32		LPSW				C2		3.4/5.2	3.4/5.2		New PSW6 = Ø
												New PSW6 = 1
Exchange Program Status	16	EPSR			95			2.4/4.2				New PSW6 = Ø
												New PSW6 = 1
Simulate Interrupt	8		SINT			E2			SVC-0.2	SVC		SVC = Interrupt
												Service Times
Supervisor Call	32		SVC			E1			4.6	4.8		
Exchange Operand Bank Address	2/15	EPOR			2E			1.6				
Exchange Program Address	20/32	EPPR			2F			2.0				

*See :-4. Extended Branch Mnemonics

Table 2-3. HMP-1116 Instruction Repertoire (Cont'd)

Instruction	Mnemonic				OP Code (Hex)			Execution Times (sec)				
	Data (Bits)	RR/SF	RI	RX	RR/SF	RI	RX	RR/SF	RI	RI Indexed	RX	Comments
Input/Output												
Acknowledge Interrupt	8	AIR		AI	9F		DF	3.2			4.6	
Sense Status	8	SSR		SS	9D		DD	2.4			3.8	
Output Command	8	OCR		OC	9E		DE	2.0			3.2	
Read Data (Byte)	8	RDR		RD	9B		DB	2.2			3.6	
Write Data (Byte)	8	WDR		WD	9A		DA	2.2			3.2	
Read Halfword	16	RHR		RH	99		D9	3.4/2.8			5.0/4.0	Byte/HW
Write Halfword	16	WHR		WH	98		D8	3.8/3.0			4.6/3.8	Byte/HW
Autoload	8n			AL			D5				4.4+3.6n	n=No. of Bytes
Read Block	8n	RBR		RB	97		D7	4.0+2.8n			4.2+2.8n	n=No. of Bytes
Write Block	8n	WBR		WB	96		D6	4.2+2.6n			4.4+2.6n	n=No. of Bytes
List Processing												
Add to Top of List	16			ATL			64				3.2/6.4/ 6.6	OVF/Norm/Wrap
Add to Bottom of List	16			ABL			65				3.2/6.4/ 6.4	OVF/Norm/Wrap
Remove from Top of List	16			RTL			66				2.6/5.6/ 5.8	Empty/Norm/Wrap
Remove from Bottom of List	16			RBL			67				2.6/5.8/ 5.8	Empty/Norm/Wrap

Table 2-4. Extended Branch Mnemonics

Instruction	Mnemonic	OP-Code and M1 Mask
Branch on Carry	BC	428
Branch on Carry RR	BCR	028
Branch on Carry Short	BCS	208/218
Branch on No Carry	BNC	438
Branch on No Carry RR	BNCR	038
Branch on No Carry Short	BNCS	228/238
Branch on Equal	BE	433
Branch on Equal RR	BER	033
Branch on Equal Short	BES	223/233
Branch on Not Equal	BNE	423
Branch on Not Equal RR	BNER	023
Branch on Not Equal Short	BNES	203/213
Branch on Low	BL	428
Branch on Low RR	BLR	028
Branch on Low Short	BLS	208/218
Branch on Not Low	BNL	438
Branch on Not Low RR	BNLR	038
Branch on Not Low Short	BNLS	228/236
Branch on Minus	BM	421
Branch on Minus RR	BMR	021
Branch on Minus Short	BMX	201/211
Branch on Not Minus	BNM	431
Branch on Not Minus RR	BNMR	031
Branch on Not Minus Short	BNMS	221/231
Branch on Plus	BP	422
Branch on Plus RR	BPR	022
Branch on Plus Short	BPS	202/212
Branch on Not Plus	BNP	432
Branch on Not Plus RR	BNPR	032
Branch on Not Plus Short	BNPS	222/232
Branch on Overflow	BO	424
Branch on Overflow RR	BOR	424
Branch on Overflow Short	BOS	204/214
Branch on No Overflow	BNO	434
Branch on No Overflow RR	BNOR	034
Branch on No Overflow Short	BNOS	224/234

Table 2-4. Extended Branch Mnemonics (Cont'd)

Instruction	Mnemonic	OP-Code and M1 Mask
Branch on Zero	BZ	433
Branch on Zero RR	BZR	033
Branch on Zero Short	BZS	223/233
Branch on Not Zero	BNZ	423
Branch on Not Zero RR	BNZR	023
Branch on Not Zero Short	BNZS	203/213
Branch Unconditional	B	430
Branch Unconditional RR	BR	030
Branch Unconditional Short	BS	220/230
No Operation	NOP	420
No Operation RR	NOPR	020

Table 2-5. Memory Addressing Example

Address	0050	0051	0052	0053	0054	0055	0056	0057
Contents	01	23	45	67	89	AB	CD	EF
Operand Length and Position	Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
	Halfword		Halfword		Halfword		Halfword	
	Fullword				Fullword			
	Floating-Point				Floating-Point			

For example, if the address referenced in Table 2-5 is 0050_{16}, then;

A Byte-Oriented instruction would extract the value 01_{16}, as an operand.

A Halfword-Oriented instruction would extract the value 0123_{16} as an operand.

A Fullword instruction would extract the value 01234567_{16} as an operand.
A Floating-Point instruction would extract the value 01234567_{16} as an operand.

f. Memory Addressing. Memory locations are numbered consecutively, beginning at 0000, for each 8-bit byte. Operands in memory shall be addressed by the RX type instructions. Since the address portion (A) of an RX instruction is 16-bits wide, it is possible to directly address 65,536 bytes within a memory bank.

The Processor transfers binary information between the Memory and the Processor as 16-bit halfwords. The instruction being performed determines if the address specified is that of a byte, a halfword, a fullword or floating-point word. If a byte of information is desired, either the left or right byte of the halfword read from memory is manipulated as determined by the specific address. If a halfword of information is desired, the entire 16 bits read from memory are used. If a fullword or floating-point word is desired, a second 16 bits are read from memory and combined with the original halfword. Table 2-5 gives an example of addressing.

(1) Bank and Memory Addressing. Memory banks are numbered consecutively from zero to three. Memory locations within a bank are numbered consecutively, beginning at 0000, for each 8-bit byte. Bits 8 and 9 of the PSW select the memory bank for instruction accessing and bits 10 and 11 of the PSW select the memory bank for operand accessing, allowing the operand to be in a different bank from the instruction. Operands within a memory bank are addressed by the RX type instructions. Since the address portion (A) of an RX instruction is 16 bits wide, it is possible to directly address 65,536 bytes within a memory bank.

(a) Accessing. The Processor transfers binary information between the Memory and the Processor as 16-bit halfwords. The instruction being performed determines if the address specified is that of a byte, a halfword, a fullword or floating-point word. If a byte of information is desired, either the left or right byte of the halfword read from memory is manipulated as determined by the specific address. If a halfword of information is desired, the entire 16 bits read from memory are used. If a fullword or floating-point word is desired, a second 16 bits are read from memory and combined with the original halfword. Table 2-5 gives an example of addressing.

(b) Addressing. Bytes of information are addressed by their specific hexadecimal address. A group of bytes combined to form a halfword, a fullword, or floating-point word are addressed by the leftmost byte in the group. Halfword, fullword, or floating-point word operands must be positioned at an address which is a multiple of two. Any memory reference for either a halfword, a fullword or floating-point word of information must reference that halfword, fullword, or floating-point word with an address which is a multiple of two. The use of an address which is odd may yield an undefined result.

(2) Addressing Modes. The Processor is capable of direct addressing, indexed addressing. and relative (Branch instructions only) addressing modes.

(3) Effective Address Generation. The Processor generates the effective address as follows:

(a) Direct Addressing. The second halfword of a fullword instruction is used as the effective address within the selected bank.

(b) Indexed Addressing. The effective memory address within the selected bank is computed as the sum of the contents of a general purpose register and the second halfword of a fullword instruction. If the sum exceeds 65,535, only the 16 less significant bits of sum are used (modulo 65,536).

(c) Relative Addressing. The effective address is a 4-bit displacement (±15 halfwords) relative to the present location counter. Bank boundaries cannot be crossed.

(4) Maximum Addressing Range. The Processor, Memory and DMA bus are all capable of directly addressing up to 262,144 bytes of memory.

(5) Memory Allocation. Locations in memory bank zero are allocated for Floating-Point Registers, register save areas, and interrupt processing as specified in Table 2-6 and described in the following paragraphs.

(a) Floating-Point Registers. Eight 32-bit registers are reserved for use by the Floating-Point instructions. The floating-point registers occupy memory locations 00 thru 1F and shall be addressable by even numbers 0, 2, 4, 6, 8, A, C, E. The floating-point registers are normally addressable by the Floating-Point instructions.

(b) Power Fail Locations. The Register Save Pointer at Location X'22', points to the first of 16 consecutive halfword locations in memory where the General Registers are saved in the event of power failure. When power is restored, the General Registers are restored automatically from these locations. The Current PSW is saved and restored in similar fashion from Location X'24' - X'27'.

(c) Interrupt PSWs. These locations are reserved for the Old and New PSWs for the various internal and external interrupts.

(d) Bootstrap Loader. Locations 50 to 7F are reserved for a bootstrap loader capable of reading a block of data from a byte-oriented device.

(e) Channel/I/O Termination Parameters. These locations are used in conjunction with Termination interrupts from automatic I/O channel operation.

(f) Supervisor Call Parameters. These locations are used for the PSW exchange associated with the Supervisor Call (SVC) instruction.

I-2. CATEGORICAL LIST OF HMP1116 OPERATION CODES

Data Instruction	(Bits)	Mnemonic RR/SF	Mnemonic RI	Mnemonic RX	Op Code (Hex) RR/SF	Op Code (Hex) RI	Op Code (Hex) RX
Fixed-Point Load/Store							
Load Immediate Short	4	LIS			24		
Load Complement Short	4	LCS			25		
Load Halfword	16	LHR	LHI	LH	08	C8	48
Load Fullword	32	LDPR		LDP	18		58
Load from Bank 0	16			LH0			74
Load from Bank 1	16			LH1			75
Load from Bank 2	16			LH2			76
Load from Bank 3	16			LH3			77
Load Multiple	16			LM			D1
Store Halfword	16			STH			40
Store Fullword	32			STDP			50
Store in Bank 0	16			STH0			F8
Store in Bank 1	16			STH1			F9
Store in Bank 2	16			STH2			FA
Store in Bank 3	16			STM			D0
Fixed-Point Arithmetic							
Add Immediate Short	4	AIS			26		
Add Halfword	16	AHR	AHI	AH	0A	CA	4A
Add Halfword to Memory	16			AHM			61
Add with Carry Halfword	16	ACHR		ACH	0E		4E
Add Fullword	32	ADPR		ADP	1A		5A
Subtract Immediate Short	4	SIS			17		
Subtract Halfword	16	SHR	SHI	SH	0B	CB	4B
Subtract with Carry Halfword	16	SCHR		SCH	0F		4F
Subtract Fullword	32	SDPR		SDP	1B		5B
Multiply Halfword	16	MHR		MH	0C		4C
Multiply Halfword Unsigned	16	MHUR		MHU	9C		DC
Multiply Fullword	32	MDPR		MDP	1C		5C
Divide Halfword	16	DHR		DH	0D		4D
Divide Fullword	32	DDPR		DDP	1D		5D

Data		Mnemonic			Op Code (Hex)		
Instruction	(Bits)	RR/SF	RI	RX	RR/SF	RI	RX
		Logical and Compare					
AND Halfword	16	NHR	NHI	NH	04	C4	44
OR Halfword	16	OHR	OHI	OH	06	C6	46
Exclusive OR Halfword	16	XHR	XHI	XH	07	C7	47
Test Halfword Immediate	16		THI			C3	
Compare Halfword	16	CHR	CHI	CH	09	C9	49
Compare Fullword	32	CDPR		CDP	19		59
Compare Logical Halfword	16	CLHR	CLHI	CLH	05	C5	45
Compare Logical Fullword	32	CLDPR		CLDP	15		55
		Byte Handling					
Load Byte	8	LBR		LB	93		D3
Store Byte	8	STBR		STB	92		D2
Exchange Byte	8	EXBR			94		
Compare Logical Byte	8			CLB			D4
		Shift and Rotate					
Shift Left Logical Short	16	SLLS			91		
Shift Left Halfword Logical	16		SLHL			CD	
Shift Left Fullword Logical	32		SLL			ED	
Shift Left Logical Doubleword	32		SLGL			E7	
Shift Right Logical Short	16	SRLS			90		
Shift Right Halfword Logical	16		SRHL			CC	
Shift Right Fullword Logical	32		SRL			EC	
Rotate Left Fullword Logical	32		RLL			EB	
Rotate Right Fullword Logical	32		RRL			EA	
Shift Left Halfword Arithmetic	16		SLHA			CF	
Shift Left Fullword Arithmetic	32		SLA			EF	
Shift Left Doubleword Arithmetic	32		SLQA			E9	

Data Instruction	(Bits)	Mnemonic RR/SF	Mnemonic RI	Mnemonic RX	Op Code (Hex) RR/SF	Op Code (Hex) RI	Op Code (Hex) RX
Shift and Rotate (Cont'd)							
Shift Right Halfword Arithmetic	16		SRHA			CE	
Shift Right Fullword Arithmetic	32		SRA			EE	
Shift Right Doubleword Arithmetic	32		SRQA			58	
Branch*							
Branch on True Backward Short	--	BTBS			20		
Branch on True Forward Short	--	BTFS			21		
Branch on True Condition	--	BTCR		BTC	02		42
Branch on False Backward Short	--	BFBS			22		
Branch on False Forward Short	--	BFFS			23		
Branch on False Condition	--	BFCR		BFC	03		43
Branch on Index High	--		BXH			C0	
Branch on Index Low or Equal	--		BXLE			C1	
Branch and Link	--	BALR		BAL	01		41
Floating-Point							
Floating-Point Load	32	LER		LE	28		68
Floating-Point Store	32			STE			60
Floating-Point Add	32	AER		AE	3A		6A
Floating-Point Subtract	32	SER		SE	2B		6B
Floating-Point Compare	32	CER		CE	29		09
Floating-Point Multiply	32	MER		ME	2C		6C
Floating-Point Divide	32	DER		DE	2D		6D
System Control							
Load Program Status Word	32		LPSW				C2
Exchange Program Status	16	EPSR			96		
Simulate Interrupt	8		SINT			E2	

Data Instruction	(Bits)	Mnemonic RR/SF	Mnemonic RI	Mnemonic RX	Op Code (Hex) RR/SF	Op Code (Hex) RI	Op Code (Hex) RX
System Control (Cont'd)							
Supervisor Call	32		SVC			E1	
Exchange Operand Bank Address	2/15	EPOR			2E		
Exchange Program Address	20/32	EPPR			2F		
Input/Output							
Acknowledge Interrupt	8	AIR		AI	9F		DF
Sense Status	8	SSR		SS	9D		DD
Output Command	8	OCR		OC	9E		DE
Read Data (Byte)	8	RDR		RD	9B		DB
Write Data (Byte)	8	WDR		WD	9A		DA
Read Halfword	16	RHR		RH	99		D9
Write Halfword	16	WHR		WH	98		D8
Autoload	8n			AL			D5
Read Block	8n	RBR		RB	97		D7
Write Block	8n	WBR		WB	96		D6
List Processing							
Add to Top of List	16			ATL			64
Add to Bottom of List	16			ABL			65
Remove from Top of List	16			RTL			66
Remove from Bottom of List	16			RBL			67
Added Instructions							
Set Status Bits	8		SESB			E4	
Reset Status Bits	8		RESB			E3	
Load Complement Halfword	16	LCHR		LCH	12		53
Load and Change Number Base Halfword	16	LCNHR		LCNH	10		51
Load Absolute Value Halfword	16	LAVR		LAV	11		52

NOTE: For a list of 48-bit floating-point instructions, see Table VI-5.

Appendix VI

INSTRUCTIONS

VI-1. CONTROLLER COMPUTER INSTRUCTIONS DESCRIPTION. The execution of each instruction shall effect the Condition Code in the Program Status Register as specified in the CVGL (see definitions below) chart accompanying the description of each instruction. Each CVGL chart illustrates the possible variations of the Condition Code where one indicates set, zero indicates reset, and X indicates undefined after the execution of the instruction. The operation code and the locations of all fields shall be as specified in the corresponding instruction diagrams. All operation codes are represented in hexadecimal notation. The execution time for each instruction is specified in Appendix IV. For the purpose of this TM, the instructions are grouped in the following categories.

Category of Instruction	Number of Types	Paragraph
Fixed-Point Load/Store	19	VI-1b.
Fixed-Point Arithmetic	27	VI-1c.
Logical and Compare	20	VI-1d.
Byte Handling	6	VI-1e.
Shift/Rotate	16	VI-1f.
Branch	12	VI-1g.
Floating-Point	13	VI-1h.
System Control	6	VI-1i.
Input/Output	19	VI-1j.
List Processing	4	VI-1k.
Trigonometric	9	VI-1l.
Instruction Augment Set	5	VI-1m.
TOTAL	156	

NOTE: For a list of 48-bit floating-point instructions, see Table VI-5.

The symbols and abbreviations used in the following subparagraphs are defined as follows:

( ) [ ]	Parentheses or Brackets. Read as "the content of ....."
<--	Arrow. Read as "is replaced by ..." or "replaces ...".

A2	The 16-bit halfword address which is a part of the RX instructions.
R1	The address of a General Register, the content of which is the first operand.
M1	Mask of four bits specifying Branch on Condition testing.
R2	The address of a General Register, the content of which is the second operand of an RR instruction.
I2	The immediate value which is used as the second operand.
X2	The address of a General Register, the content of which is used as an index value.
N	The four bit second operand used with Short Format Immediate instructions, and the four bit displacement value used with Short Format Branck instructions.
(0:7) (8:15) (16:31)	A bit grouping within a byte, a halfword, or a fullword. Read as "0 thru 7 inclusive", "bits 8 through 15 inclusive", etc.
PSW	Program Status Word of 32 bits containing the Operational Control, Condition Code, and current instruction address.
CC	Condition Code of four bits contained in the PSW.
C	Carry Bit contained in the Condition Code (Bit 12 of PSW).
V	Overflow Bit contained in the Condition Code (Bit 13 of PSW).
G	Greater Than Bit contained in the Condition Code (Bit 14 of PSW).
L	Less Than Bit contained in the Condition Code (Bit 15 of PSW).
+	Add
-	Subtract
*	Multiply
/	Divide
:	Logical comparison, when used (e.g., R1:R2)
(R1, R1+1)	Fullword contained in a pair of registers.
[A2+(X2), A2+(X2)+2]	Fullword located in memory.

a. Effective Address Generation. The effective address for accessing memory will be generated by combining two MSBs and sixteen LSBs for all RX format instructions, except if specifically stated otherwise in the instruction description. The two MSBs shall be the operand bank bits 10 and 11 of the Current PSW. The sixteen LSBs shall be derived by adding the A field and the content of the general register specified by the X2 field (i.e., A2+(X2)). This shall give the item a memory addressing capacity of 256 K-bytes or 128 K-halfwords.

b. Fixed-Point Load/Store Instructions. The Fixed-Point Load/Store instructions may be used to preset a register with an index value, load a register with the first operand for a subsequent arithmetic operation (e.g., add, multiply), or set the Condition Code for supplemental testing by a Branch on Condition instruction. These instructions shall use the Register to Register (RR), the Short Format (SF), the Register to Indexed Storage (RX), and the Register and Immediate Storage (RI) formats. The exact format, op-code, assembler notation, and diagrammatic representation of each instruction shall be as shown in Figures VI-1, -2, and -3. The operation and resulting Condition Code shall be as follows:

(1) Load Immediate Short. The Load Immediate Short (LIS) instruction shall cause the four bit second operand to be expanded to a 16-bit halfword with high order bits set to zero. This halfword shall be loaded into the General Register specified by R1. The resulting Condition Code shall be:

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

(2) Load Complement Short. The Load Complement Short (LCS) instruction shall cause the four bit second operand to be expanded to a 16-bit halfword with high order bits set to zero. the two's complement of this halfword shall be loaded into the General Register specified by R1. The resulting Condition Code shall be:

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

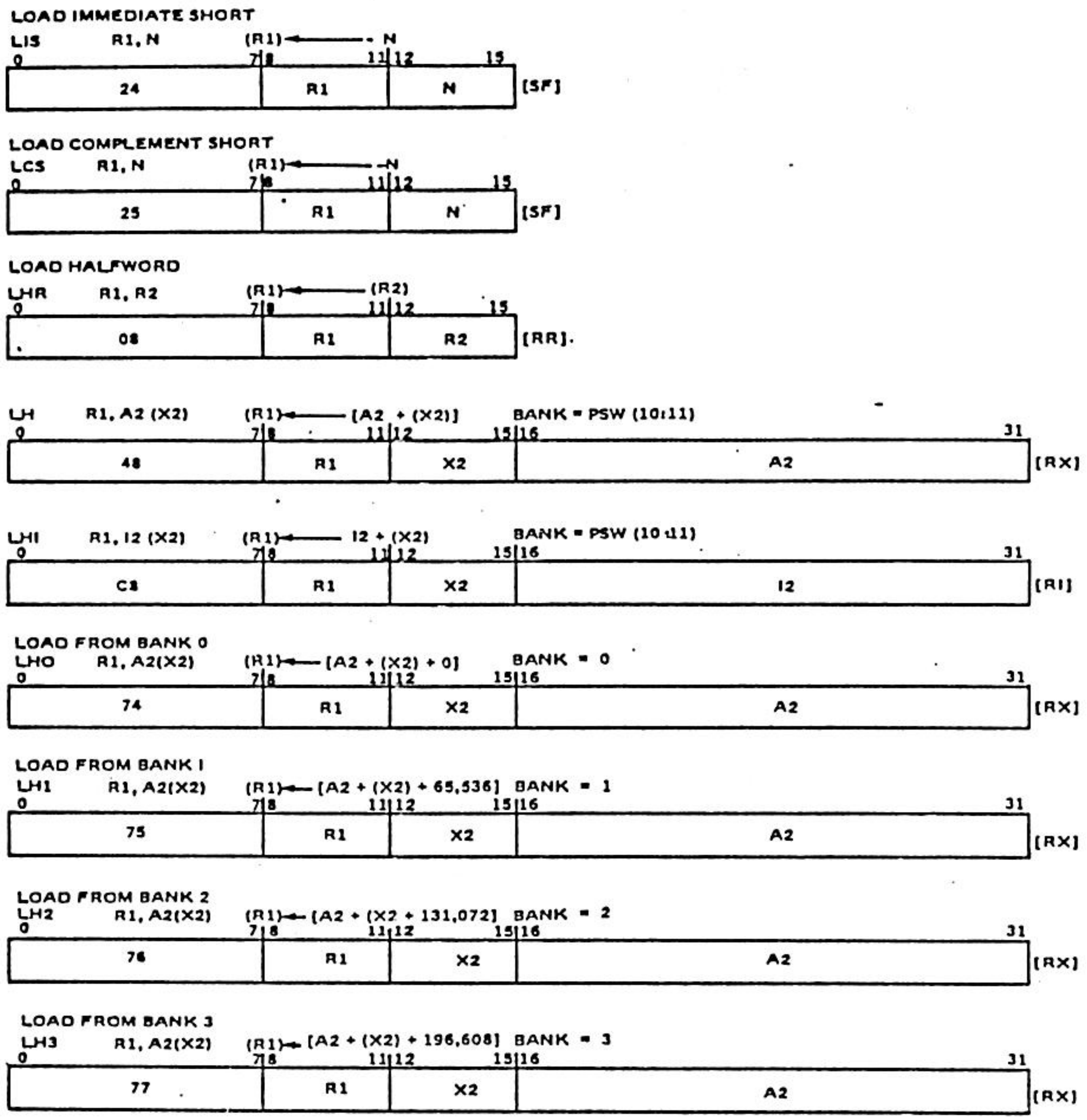
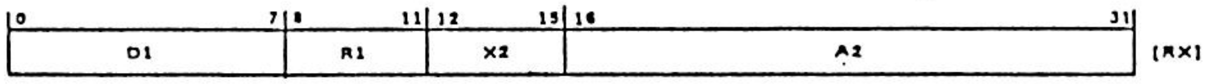


Figure VI-1. Fixed-Point Load Instructions

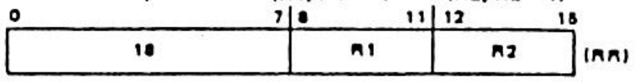
LOAD MULTIPLE
LM R1, A2 (X2)

1. (R1)⟵ (A2 + (X2))
2. R1: X'F'
   IF R1 = X'F', THE INSTRUCTION IS FINISHED
   IF R1 ≠ X'F', THEN:
3. R1⟵R1 + 1
4. A2⟵A2 + 2, RETURN TO STEP 1

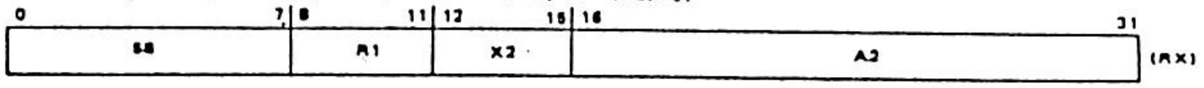


LOAD FULLWORD

LDFR R1, R1 (R1, R1 + 1) ⟵ (R2, R2 + 1)

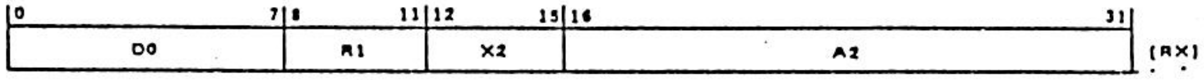


LDF R1, A2 (X2) (R1, R1 + 1) ⟵ (A2 + (X2), A2 + (X2) + 2)



STORE MULTIPLE
STM R1, A2 (X2)

1. (R1)⟶(A2 + (X2))
2. R1: X'F'
   IF R1 = X'F', THEN INSTRUCTION IS FINISHED
   IF R1 ≠ X'F', THEN:
3. R1⟵R1 + 1
4. A2⟵A2 + 2, RETURN TO STEP 1



STORE FULL WORD

STDF R1, A2 (X2) (R1, R1 + 1)⟶(A2 + (X2), A2 + (X2) + 2)

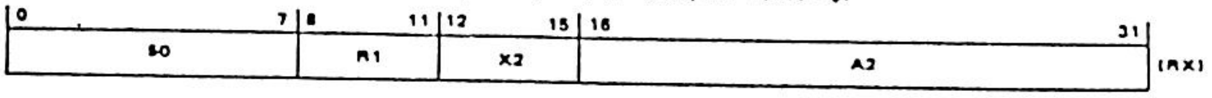


Figure VI-2. Multiple Load/Store Instructions

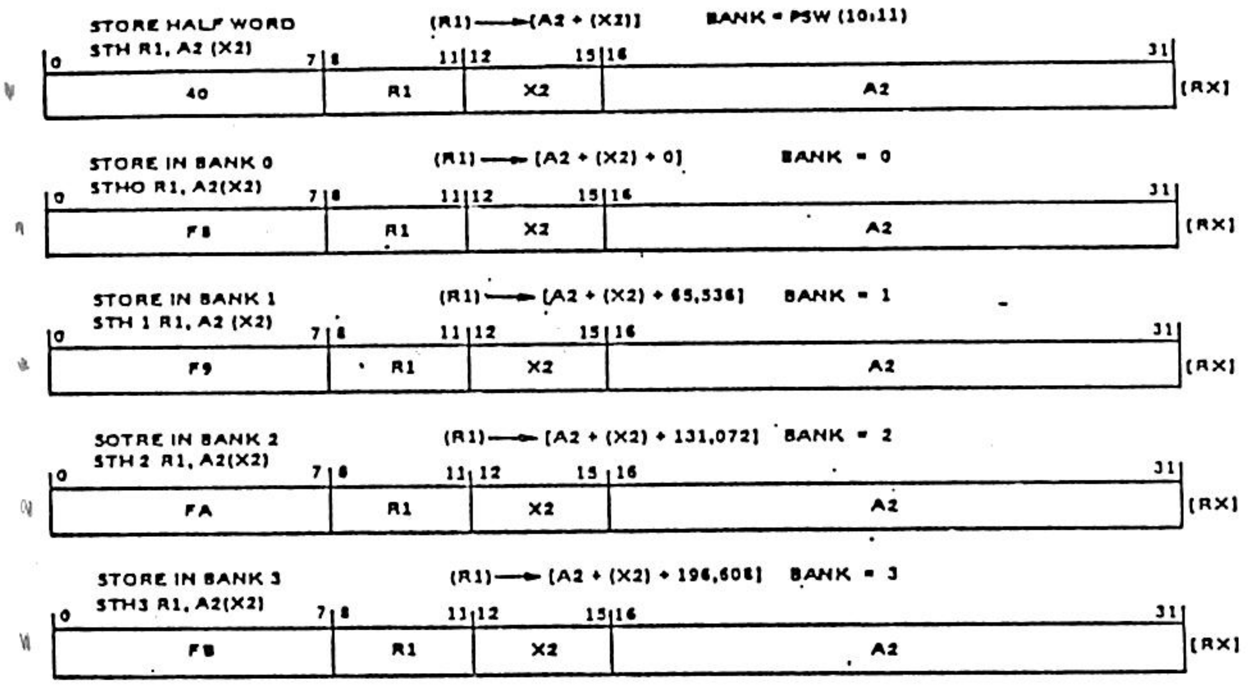


Figure VI-3. Fixed-Point Store Instructions

(3) Load Halfword. The Load Halfword (LHR. LH, LHI) instructions shall cause the second operand to be loaded into the General Register specified by R1. In the RR format, if R1 equals R2, the load instruction shall function as a test on the content of the register. In the RX format, the second operand shall be located on a halfword boundary. The resulting Condition Code shall be:

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

(4) Load Fullword Instructions. The load fullword (LDPR and LDP) instructions cause the second fullword operand to be loaded into the general registers specified by R1 and R1 + 1. R1 and R2 specify even numbered registers. In the RR format, if R1 equals R2, the load instruction functions as a test on the contents of the register pair. In the RX format, the second operand is located on a halfword boundary. The resulting Condition Code shall be:

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

(5) Load From Bank 0. The Load From Bank 0 (LHO) instruction shall be a halfword load instruction whose effective address shall be computed relative to bank 0, without regard to the operand bank bits (10:11) in the Current PSW. The instruction shall cause the second operand to be loaded into the General Register specified by R1. The second operand shall be located on a halfword boundary. The resulting Condition Code shall be:

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

(6) Load From Bank 1. The Load From Bank 1 (LH1) instruction shall be a halfword load instruction whose effective address shall be computed relative to bank 1, without regard to the operand bank bits (10:11) in the Current PSW. The instruction shall cause the second operand to be loaded into the General Register specified by R1. The second operand shall be located on a halfword boundary. The resulting Condition Code shall be:

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

(7) Load From Bank 2. The Load From Bank 2 (LH2) instruction shall be a halfword load instruction whose effective address shall be computed relative to bank 2, without regard to the operand bank bits (10:11) in the Current PSW. The instruction shall cause the second operand to be loaded into the General Register specified by R1. The second operand shall be located on a halfword boundary. The resulting Condition Code shall be:

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

(8) Load From Bank 3. The Load From Bank 3 (LH3) instruction shall be a halfword load instruction whose effective address shall be computed relative to bank 3, without regard to the operand bank bits (10:11) in the Current PSW. The instruction shall cause the second operand to be loaded into the General Register specified by R1. The second operand shall be located on a halfword boundary. The resulting Condition Code shall be:

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

(9) Load Multiple. The Load Multiple (LM) instruction shall cause sequential halfwords from memory to be loaded into successive General Registers, beginning with the General Register specified by the R1 field. The first halfword shall be defined by A2+(X2) and shall be located on a halfword boundary. The operation shall be terminated when R15 is loaded from memory. Note that any number of sequential General Registers can be loaded in this manner. The Condition Code shall be unchanged by the execution of this instruction.

(10) Store Multiple. The Store Multiple (STM) instruction shall cause successive General Registers to be stored sequentially into memory, beginning with the General Register specified by the R1 field. The first storage address shall be determined by A2+(X2) and shall be located on a halfword boundary. The operation shall be terminated when R15 is stored in memory. Note that any number of sequential General Registers can be transferred in this manner. The Store Multiple (STM) instruction, in conjunction with the Load Multiple (LM) instruction is an aid to subroutine execution. They permit the easy saving and restoring of the registers required by the subroutine. The Store Multiple instruction can be used upon entering the subroutine, and the Load Multiple would be the last instruction executed before returning from the subroutine. The Condition Code shall be unchanged by the execution of this instruction.

(11) Store Halfword. The Store Halfword (STH) instruction shall store the 16-bit first operand in the memory location specified by the second operand. The second operand shall be located on a halfword boundary. The first operand shall remain unchanged. The Condition Code shall remain unchanged.

(12) Store In Bank 0. The Store in Bank 0 (STH0) instruction shall be a halfword store instruction whose second operand effective address shall be computed relative to bank 0, without regard to the operand bank bits (10:11) in the Current PSW. The instruction shall store the 16-bit first operand in the memory location specified by the second operand. The second operand shall be located on a halfword boundary. The first operand shall remain unchanged. The Condition Code shall remain unchanged.

(13) Store In Bank 1. The Store in Bank 1 (STH1) instruction shall be a halfword store instruction whose second operand effective address shall be computed relative to bank 1, without regard to the operand bank bits (10:11) in the Current PSW. The instruction shall store the 16-bit first operand in the memory location specified by the second operand. The second operand shall be located on a halfword boundary. The first operand shall remain unchanged. The Condition Code shall remain unchanged.

(14) Store In Bank 2. The Store in Bank 2 (STH2) instruction shall be a halfword store instruction whose second operand effective address shall be computed relative to bank 2, without regard to the operand bank bits (10:11) in the Current PSW. The instruction shall store the 16-bit first operand in the memory location specified by the second operand. The second operand shall be located on a halfword boundary. The first operand shall remain unchanged. The Condition Code shall remain unchanged.

(15) Store In Bank 3. The Store in Bank 3 (STH3) instruction shall be a halfword store instruction whose second operand effective address shall be computed relative to bank 3, without regard to the operand bank bits (10:11) in the Current PSW. The instruction shall store the 16-bit first operand in the memory location specified by the second operand. The second operand shall be located on a halfword boundary. The first operand shall remain unchanged. The Condition Code shall remain unchanged.

(16) Store Fullword Instructions. The store fullword (STDP) instruction causes two successive general registers to be stored sequentially into memory, beginning with the general register specified in the R1 field. R1 specifies an even numbered register. The first storage address is determined by A2 + (X2) + 2. The contents of the general register pair remains unchanged. The Condition Code remains unchanged. This instruction is subject to memory protect (when available).

c. Fixed-Point Arithmetic Instructions. The item shall execute the Fixed-Point Arithmetic instructions to provide for addition, subtraction, multiplication, and division of a fixed-point data contained in the general register and/or memory. The Fixed-Point Arithmetic instructions provide for calculating addresses and indexes for counting, and for general purpose fixed-point arithmetic. The Fixed-Point Arithmetic instructions shall use the RR, SF, RX, and RI formats. The exact format, op-code, assembler notation, and diagrammatic representation of each instruction shall be as shown in Figures VI-4, -5, -6, and -7. The operation and resulting Condition Code shall be as follows:

(1) Add Immediate Short. The Add Immediate Short (AIS) instruction shall cause the four bit second operand N to be added to the contents of the General Register specified by R1. The second operand shall be expanded to a 16-bit halfword by forcing the high order bits to zero. The resulting Condition Code shall be:

C	V	G	L	
X	X	0	0	Sum is zero
X	X	0	1	Sum is less than zero
X	X	1	0	Sum is greater than zero
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

(2) Add Halfword. The Add Halfword (AHR, AH, and AHI) instructions shall cause the second operand to be added algebraically to the contents of the General Register specified by R1. The result of this 16-bit addition replaces the contents of the register specified by R1. In the RX format, the second operand must be located on a halfword boundary. The Add Halfword Immediate (AHI) instruction shall produce a value which is the algebraic sum of the address field itself, the content of a General Register index (X2), and the first operand General Register (R1).The resulting Condition Code shall be:

C	V	G	L	
X	X	0	0	Sum is zero
X	X	0	1	Sum is less than zero
X	X	1	0	Sum is greater than zero
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

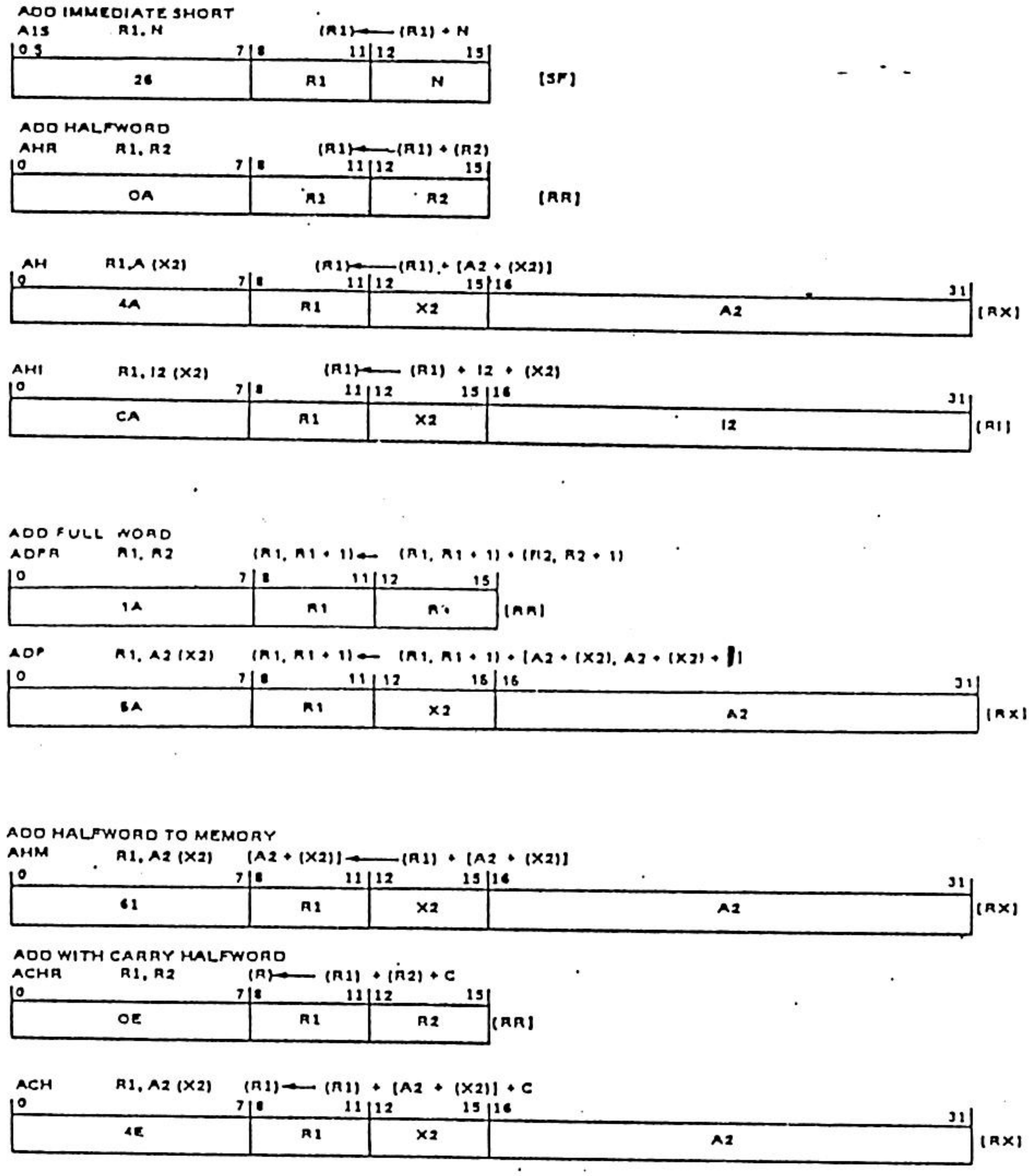

ADD IMMEDIATE SHORT
AIS R1, N (R1) ← (R1) + N
0 7 8 11 12 15
26 R1 N (SF)
ADD HALFWORD
AHR R1, R2 (R1) ← (R1) + (R2)
0 7 8 11 12 15
0A R1 R2 (RR)
AH R1,A (X2) (R1) ← (R1) + [A2 + (X2)]
0 7 8 11 12 15 16 31
4A R1 X2 A2 (RX)
AHI R1, I2 (X2) (R1) ← (R1) + I2 + (X2)
0 7 8 11 12 15 16 31
CA R1 X2 I2 (RI)
ADD FULL WORD
ADFR R1, R2 (R1, R1 + 1) ← (R1, R1 + 1) + (R2, R2 + 1)
0 7 8 11 12 15
1A R1 R2 (RR)
ADF R1, A2 (X2) (R1, R1 + 1) ← (R1, R1 + 1) + [A2 + (X2), A2 + (X2) + 1]
0 7 8 11 12 15 16 31
5A R1 X2 A2 (RX)
ADD HALFWORD TO MEMORY
AHM R1, A2 (X2) [A2 + (X2)] ← (R1) + [A2 + (X2)]
0 7 8 11 12 15 16 31
61 R1 X2 A2 (RX)
ADD WITH CARRY HALFWORD
ACHR R1, R2 (R) ← (R1) + (R2) + C
0 7 8 11 12 15
0E R1 R2 (RR)
ACH R1, A2 (X2) (R1) ← (R1) + [A2 + (X2)] + C
0 7 8 11 12 15 16 31
4E R1 X2 A2 (RX)

Figure VI-4. Fixed-Point Add Instruction

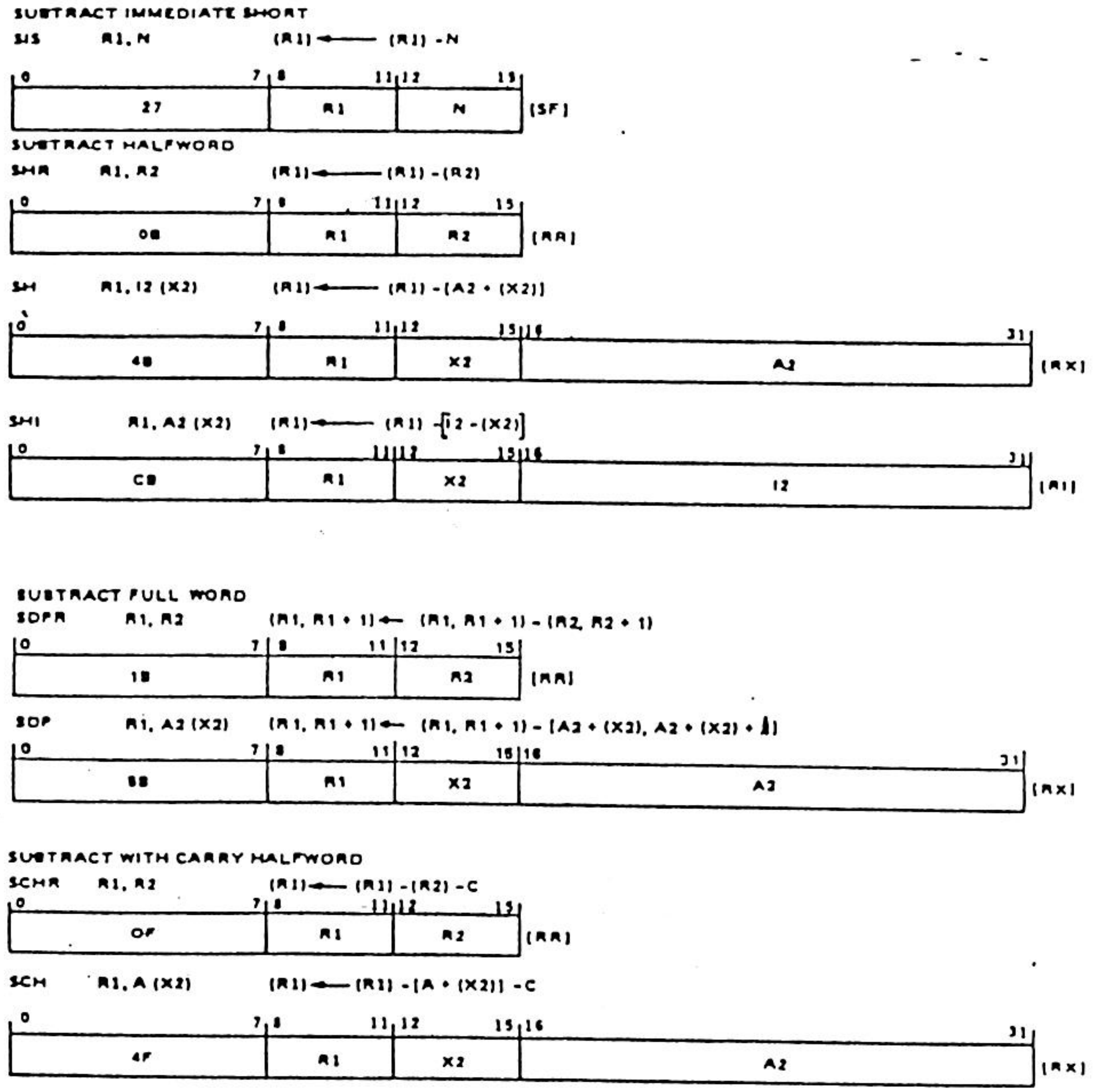

SUBTRACT IMMEDIATE SHORT
SIS R1, N (R1) ← (R1) - N
0 7 8 11 12 15
27 R1 N (SF)
SUBTRACT HALFWORD
SHR R1, R2 (R1) ← (R1) - (R2)
0 7 8 11 12 15
0B R1 R2 (RR)
SH R1, I2 (X2) (R1) ← (R1) - (A2 + (X2))
0 7 8 11 12 15 16 31
4B R1 X2 A2 (RX)
SHI R1, A2 (X2) (R1) ← (R1) - [I2 - (X2)]
0 7 8 11 12 15 16 31
CB R1 X2 I2 (RI)
SUBTRACT FULL WORD
SDFR R1, R2 (R1, R1 + 1) ← (R1, R1 + 1) - (R2, R2 + 1)
0 7 8 11 12 15
1B R1 R2 (RR)
SDF R1, A2 (X2) (R1, R1 + 1) ← (R1, R1 + 1) - [A2 + (X2), A2 + (X2) + 4]
0 7 8 11 12 15 16 31
5B R1 X2 A2 (RX)
SUBTRACT WITH CARRY HALFWORD
SCHR R1, R2 (R1) ← (R1) - (R2) - C
0 7 8 11 12 15
0F R1 R2 (RR)
SCH R1, A (X2) (R1) ← (R1) - [A + (X2)] - C
0 7 8 11 12 15 16 31
4F R1 X2 A2 (RX)

Figure VI-5. Fixed-Point Subtract Instructions

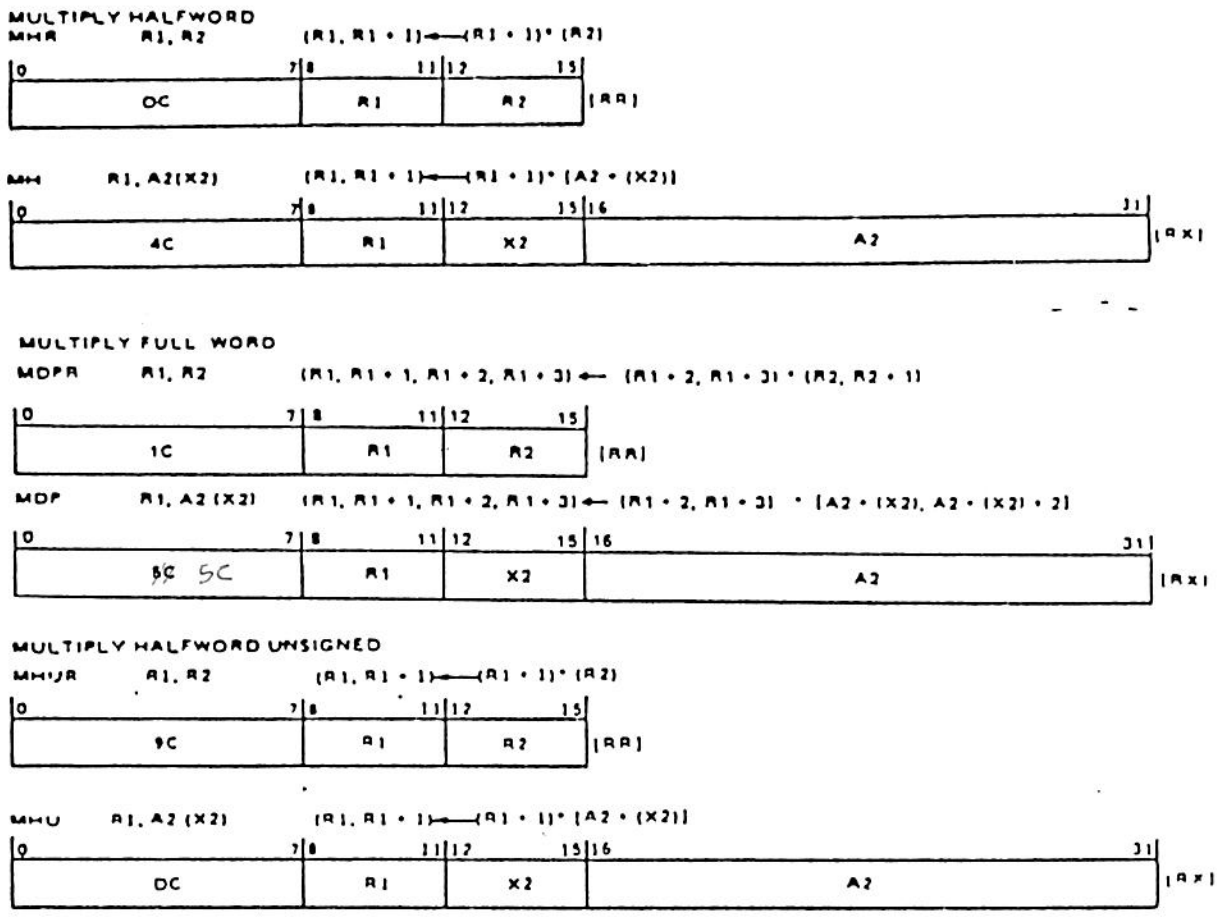

MULTIPLY HALFWORD
MHR R1, R2 (R1, R1 + 1) ← (R1 + 1) * (R2)
0 7 8 11 12 15
OC R1 R2 (RR)
MH R1, A2(X2) (R1, R1 + 1) ← (R1 + 1) * (A2 + (X2))
0 7 8 11 12 15 16 31
4C R1 X2 A2 (RX)
MULTIPLY FULL WORD
MDPR R1, R2 (R1, R1 + 1, R1 + 2, R1 + 3) ← (R1 + 2, R1 + 3) * (R2, R2 + 1)
0 7 8 11 12 15
1C R1 R2 (RR)
MDP R1, A2 (X2) (R1, R1 + 1, R1 + 2, R1 + 3) ← (R1 + 2, R1 + 3) * (A2 + (X2), A2 + (X2) + 2)
0 7 8 11 12 15 16 31
~~5C~~ 5C R1 X2 A2 (RX)
MULTIPLY HALFWORD UNSIGNED
MHUR R1, R2 (R1, R1 + 1) ← (R1 + 1) * (R2)
0 7 8 11 12 15
9C R1 R2 (RR)
MHU R1, A2 (X2) (R1, R1 + 1) ← (R1 + 1) * (A2 + (X2))
0 7 8 11 12 15 16 31
DC R1 X2 A2 (RX)

Figure VI-6. Fixed Point Multiply Instructions

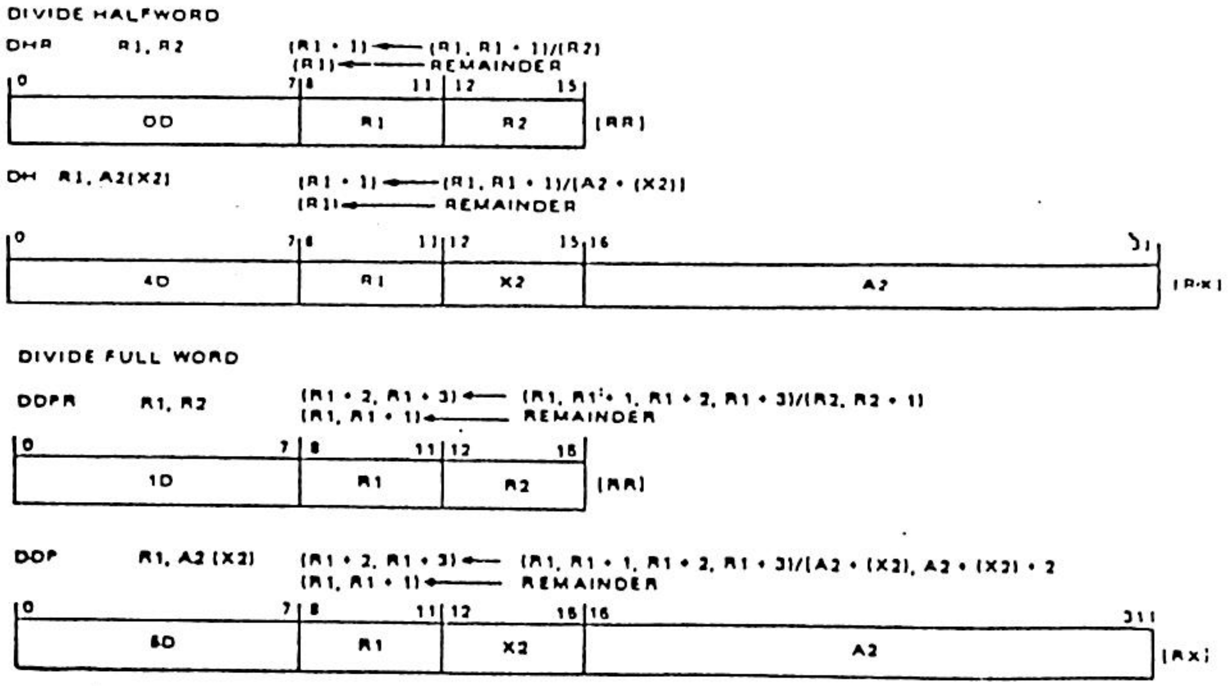

DIVIDE HALFWORD
DHR R1, R2 (R1 + 1) ← (R1, R1 + 1)/(R2)
(R1) ← REMAINDER
0 7 8 11 12 15
OD R1 R2 (RR)
DH R1, A2(X2) (R1 + 1) ← (R1, R1 + 1)/(A2 + (X2))
(R1) ← REMAINDER
0 7 8 11 12 15 16 31
4D R1 X2 A2 (RX)
DIVIDE FULL WORD
DDPR R1, R2 (R1 + 2, R1 + 3) ← (R1, R1 + 1, R1 + 2, R1 + 3)/(R2, R2 + 1)
(R1, R1 + 1) ← REMAINDER
0 7 8 11 12 15
1D R1 R2 (RR)
DDP R1, A2 (X2) (R1 + 2, R1 + 3) ← (R1, R1 + 1, R1 + 2, R1 + 3)/(A2 + (X2), A2 + (X2) + 2
(R1, R1 + 1) ← REMAINDER
0 7 8 11 12 15 16 31
5D R1 X2 A2 (RX)

Figure VI-7. Fixed-Point Divide Instructions

(3) Add Halfword to Memory. The Add Halfword to Memory (AHM) instruction shall cause the second operand [A2+(X2)] to be added to the contents of the General Register specified by R1. The result of the addition shall not replace the contents of R1; but, instead, shall be stored in memory at the address specified by A2+(X2). The first operand (R2) shall remain unchanged. This instruction effectively permits every location in core memory to be used as a counter. The second operand shall be located on a halfword boundary. The resulting Condition Code shall be:

C	V	G	L	
X	X	0	0	Sum is zero
X	X	0	1	Sum is less than zero
X	X	1	0	Sum is greater than zero
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

(4) Add With Carry Halfword. The Add With Carry Halfword (ACHR and ACH) instructions shall cause the 16-bit second operand and the Carry Bit of the Condition Code (PSW 12) to be added algebraically to the General Register specified by R1. The resulting sum shall be contained in R1. The second operand shall be unchanged. Multiple precision addition operations require a carry forward from the least significant operands to the most significant. To accomplish this, the locations containing the least significant portions of the two operands are summed using the Add Halfword (AH) instruction. A carry forward, if it occurs, is retained in the carry bit position of the Condition Code (PSW 12). The locations containing the next least significant portions of the two operands are then summed, using the Add With Carry Halfword (ACH) instruction. The carry bit contained in the Condition Code (set from the previous addition) participates in this sum; the carry bit position is then set to reflect the new result. The Add With Carry Halfword (ACH) instruction is used on succeeding pairs of operands until the most significant operand of the multiple precision words have been summed. The resulting Condition code is valid for testing the multiple precision word. The Condition Code shall be:

C	V	G	L	
X	X	0	0	Sum is zero
X	X	0	1	Sum is less than zero
X	X	1	0	Sum is greater than zero
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

(5) Add Fullword Instructions. The add fullword (ADPR and ADP) instructions cause the fullword second operand to be added algebraically to the contents of the general register pair specified by R1, R1 + 1. R1 and R2 specify even numbered registers. In the RX format, the second operand is located on a halfword boundary. The resulting Condition Code shall be:

C	V	G	L	
X	X	0	0	Sum is zero
X	X	0	1	Sum is less than zero
X	X	1	0	Sum is greater than zero
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

(6) Subtract Immediate Short. The Subtract Immediate Short (SIS) instruction shall cause the four bit second operand N to be subtracted from the contents of the general register specified by R1. The second operand is obtained by expanding the four bit data field of N to a 16-bit halfword by forcing the high order bits to zero. This instruction is useful for decrementing a register by a small value (e.g., X'2'). The resulting Condition Code shall be:

C	V	G	L	
X	X	0	0	Sum is zero
X	X	0	1	Sum is less than zero
X	X	1	0	Sum is greater than zero
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

(7) Subtract Halfword. The Subtract Halfword (SHR, SH, and SHI) instructions shall cause the second operand to be subtracted from the general register specified by R1. The difference shall be contained in R1. The second operand is unchanged. In the RX format, the second operand shall be located on a halfword boundary. The Subtract Halfword Immediate (SHI) instruction shall produce a value which is the difference between the first operand general register (R1), less the sum of the address field itself and the content of a General Register Index (X2). The resulting Condition Code shall be:

C	V	G	L	
X	X	0	0	Sum is zero
X	X	0	1	Sum is less than zero
X	X	1	0	Sum is greater than zero
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

(8) Subtract Fullword Instruction. The subtract fullword (SDPR and SDP) instructions cause the fullword second operand to be subtracted algebraically from the contents of the general register pair specified by R1, R1 + 1. The result replaces the contents of the register specified by R1, R1 + 1. R1 and R2 specify even numbered registers. In the RX format, the second operand is located on a halfword boundary. The resulting Condition Code shall be:

C	V	G	L	
X	X	0	0	Sum is zero
X	X	0	1	Sum is less than zero
X	X	1	0	Sum is greater than zero
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

(9) Subtract With Carry Halfword. The Subtract With Carry Halfword (SCHR and SCH) instructions shall cause the 16-bit second operand with the carry (borrow) bit to be subtracted from the general register specified by R1. The difference shall be contained in R1. The second operand shall be located on a halfword boundary. The resulting Condition Code shall be:

C	V	G	L	
X	X	0	0	Sum is zero
X	X	0	1	Sum is less than zero
X	X	1	0	Sum is greater than zero
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

(10) Multiply Halfword. The Multiply Halfword (MHR and MH) instructions shall cause the 16-bit second operand to be multiplied by the contents of the general register specified by R11. The R1 field of the instruction shall specify an even numbered register. The resulting 32-bit product shall be contained in R1 and R1+1, an even-odd pair; the second operand shall be unchanged. The sign of the product shall be determined by the rules of algebra. In the RX format, the second operand shall be located on a halfword boundary. After multiplication, the most significant fifteen bits with a sign shall be contained in R1. The least significant sixteen bits shall be contained in R1+1. The Condition Code shall remain unchanged.

(11) Multiply Fullword Instructions. The multiply fullword (MDPR and MDP) instructions cause the fullword second operand to multiply by the contents of the general register pair specified by R1 + 2, R1 + 3. R1 and R2 specify even numbered registers. In the RX format, the second operand is located on a halfword boundary. The resulting 64-bit product is contained in R1, R1 + 1, R1 + 2, R1 + 3. The sign of the product is determined by the rules of algebra. After multiplication, the most significant 31 bits with sign are contained in R1, R1 + 1. The least significant 32 bits are contained in R1 + 2 and R1 + 3. The Condition Code shann remain unchanged.

(12) Multiply Halfword Unsigned. The Multiply Halfword Unsigned (MHUR and MHU) instructions shall cause the 16-bit second operand to be multiplied by the contents of the general register specified by R1+1. All sixteen bits of both operands shall be considered to be magnitude. The resulting 32-bit product shall be contained in R1 and R1+1, and the second operand shall be unchanged. The R1 field of the instruction shall specify an even numbered register. This instruction is most useful in applications requiring multiple precision multiply capability. The Condition Code shall remain unchanged.

(13) Divide Fullword Instructions. The divide fullword (DDP and DDPR) instructions cause the fullword second operand to be divided into the double word dividend contained in the general register specified by R1, R1 + 1, R1 + 2, R1 + 3. R1 and R2 must specify even numbered registers. The resulting 31-bit quotient with sign is contained in R1 + 2, R1 + 3; a 31-bit remainder is contained in R1, R1 + 1. The sign of the result is determined by the rules of algebra; the sign of the remainder is the same as the sign of the dividend. In the RX format, the second operand is located on a halfword boundary. Attempted division by zero, or a quotient which would be greater than X'8000', causes a fixed-point divide fault interrupt, if enabled by bit 3 of the program status word. The operands remain unchanged when a fixed-point divide fault interrupt occurs. The Condition Code shall remain unchanged in all cases.

(10) Divide Halfword. The Divide Halfword (DH and DHR) instructions shall cause the 16-bit second operand to be divided into the 32-bit dividend contained in the general register specified by R1 and R1+1. The first operand (R1) shall specify an even numbered register. The resulting fifteen bit quotient, with sign, shall be contained by R1+1; a fifteen bit remainder, with sign, shall be contained in R1. The second operand shall be unchanged. The sign of the result shall be terermined by the rules of algebra; the sign of the remainder shall be the same as the sign of the dividend. In the RX format, the second operand shall be located on a halfword boundary. Attempted division by zero, or a quotient which would be greater than X'8000", shall cause termination of the instruction execution and a Fixed-Point Divide Fault Interrupt (if enabled by Bit 3 of the Program Status Word). The operands shall remain unchanged when a Fixed-Point Divide Fault Interrupt occurs. The Condition Code shall be unchanged in all cases.

d. Logical and Compare Instructions. The item shall execute Logical and Compare instructions such that each bit of the first operand is logically combined, or compared, with the corresponding bit in the second operand. The Logical and Compare instructions shall use the RR, RX, and RI formats. The exact format, op-code, assembler notation, and diagrammatic representation of each instruction shall be as shown in Figures VI-8 and -9. The operation and resulting Condition Code shall be as follows:

(1) AND Halfword. The AND Halfword (NH, NHR, NHI) instructions shall cause the logical product of the 16-bit second operand, and the content of the general register specified by R1, to replace the content of R1. The 16-bit product shall be formed on a bit-by-bit basis. In the RX format, the second operand shall be located on a halfword boundary. The AND Halfword Immediate (NHI) instruction shall produce a value which is the logical product of the address field itself plus the content of a General Register index (X2) with the first operand general register (R1). The truth table for the AND function is:

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

The resulting Condition Code shall be:

C	V	G	L	
X	X	0	0	Logical product is zero
X	X	0	1	Logical product is not zero
X	X	1	0	Logical product is not zero

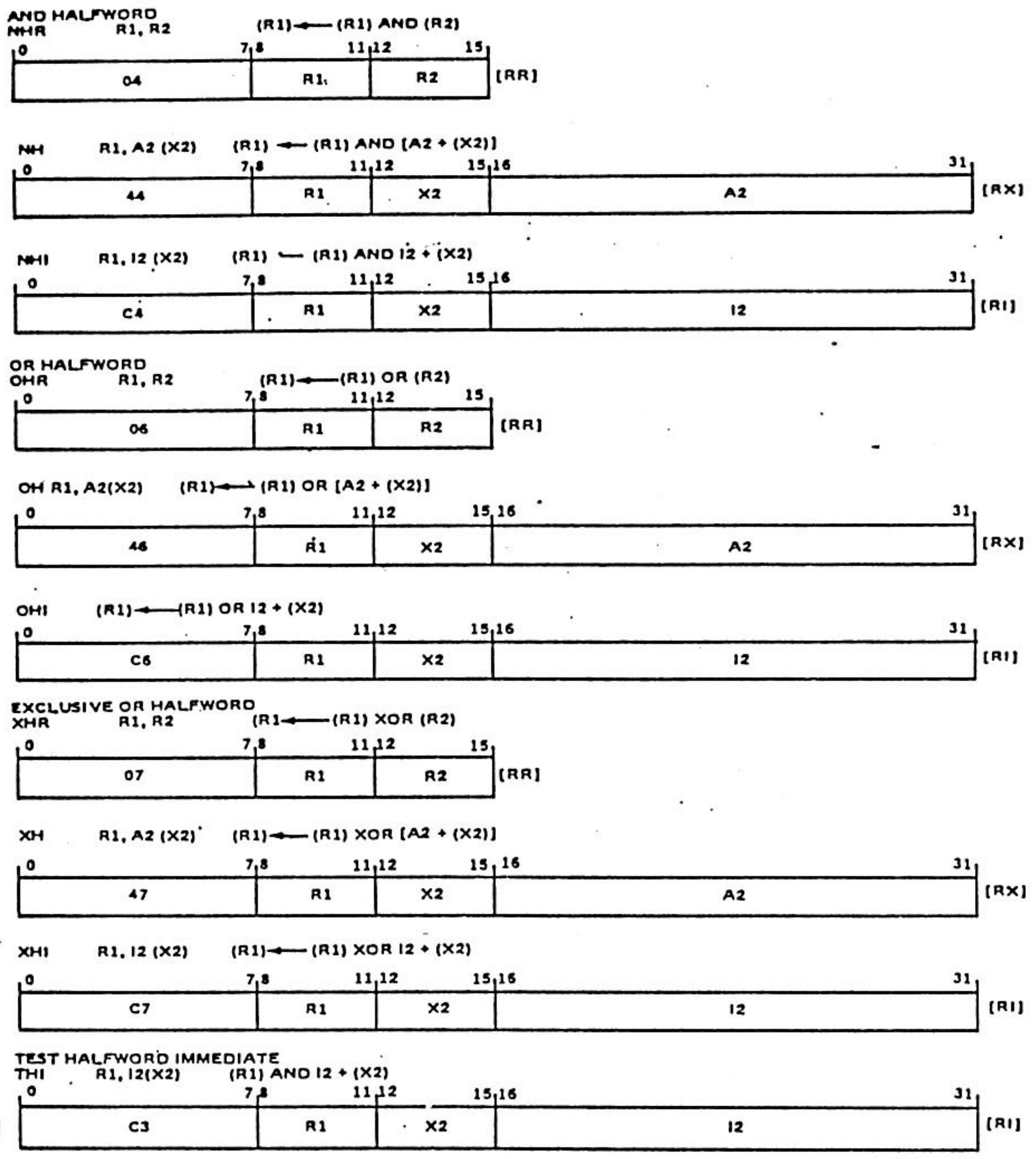


Figure VI-8. Logical and Bit Manipulating Instructions

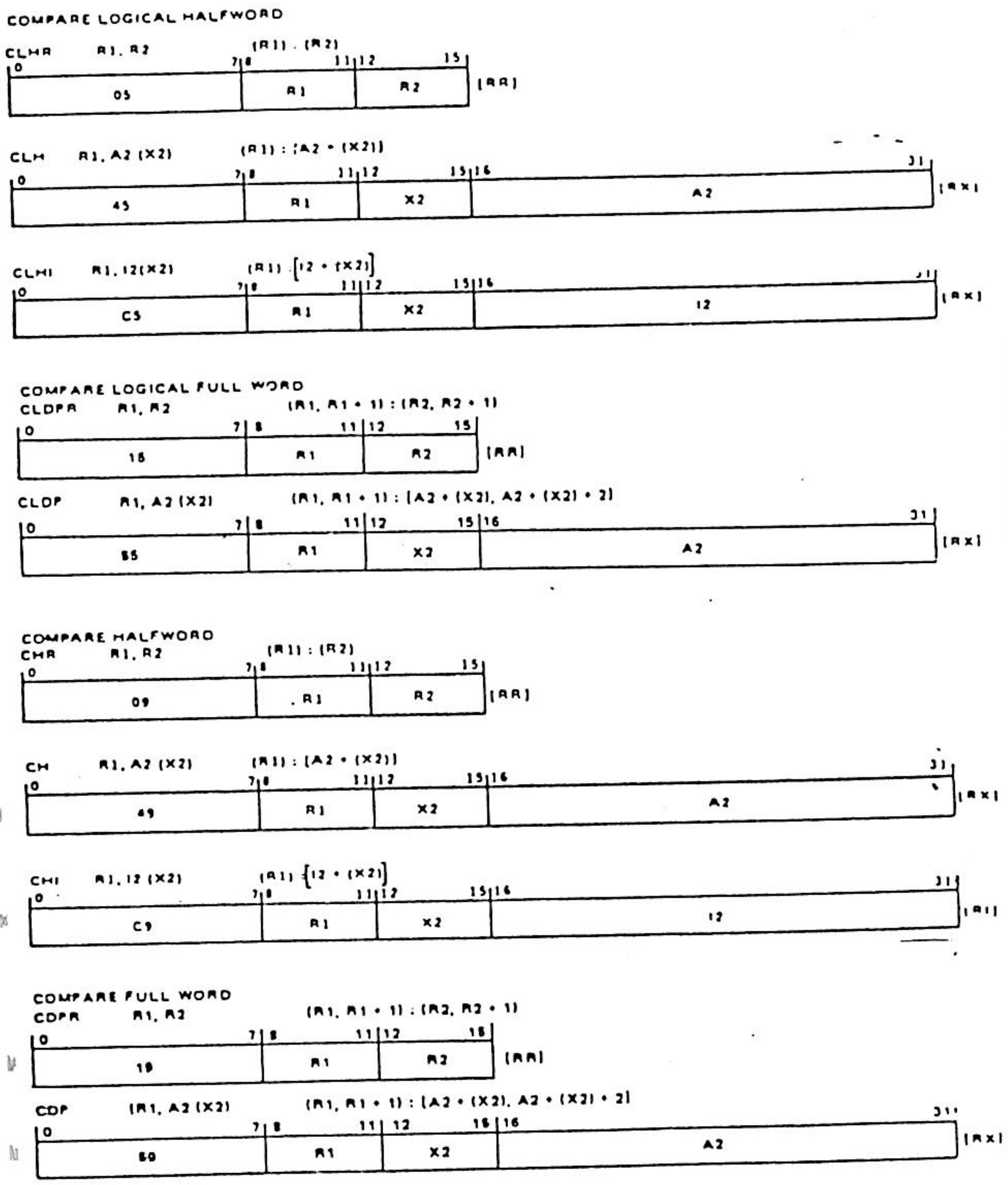


Figure VI-9. Compare Instructions

(2) OR Halfword. The OR Halfword (OH, OHR, OHI) instructions shall cause the logical sum of the 16-bit second operand and the content of the general register specified by R1 to replace the content of R1. The 16-bit sum shall be formed on a bit-by-bit basis. In the RX format, the second operand shall be located on a halfword boundary. The OR Halfword Immediate (OHI) instruction shall produce a value which is the logical sum of the address field itself plus the content of a General Register index (X2) with the first operand general register (R1). The truth table for the OR function is:

```
0 OR 0 = 0
0 OR 1 = 0
1 OR 0 = 0
1 OR 1 = 1
```

The resulting Condition Code shall be:

C	V	G	L	
X	X	0	0	Logical sum is zero
X	X	0	1	Logical sum is not zero
X	X	1	0	

(3) Exclusive OR Halfword. The Exclusive OR Halfword (XH, XHR, XHI) shall cause the logical difference of the 16-bit second operand and the general register specified by R1 to replace the content of R1. The 16-bit difference shall be formed on a bit-by-bit basis. In the RX format, the second operand shall be located on a halfword boundary. The Exclusive OR Halfword Immediate (XHI) instruction shall produce a value which is the logical difference of the address field itself plus the content of a General Register index (X2) with the first operand general register (R1). The truth table for the Exclusive OR function is:

```
0 XOR 0 = 0
0 XOR 1 = 0
1 XOR 0 = 0
1 XOR 1 = 1
```

The resulting Condition Code shall be:

C	V	G	L	
X	X	0	0	Logical sum is zero
X	X	0	1	Logical sum is not zero
X	X	1	0	

(4) Test Halfword Immediate. The Test Halfword Immediate (THI) instruction shall cause each bit in the 16-bit second operand to be logically ANDed with the corresponding bit in the general register specified by R1. The contents of R1 and the second operand shall remain unchanged. The Test Halfword Immediate instruction can be used to test the state of individual bits, or combinations of bits, in a general register. For example, to test the state of Bit 6 in Register 3, use THI 3,X'0200'. The resulting Condition Code shall be:

C	V	G	L	
X	X	0	0	None of the bits of the result set
X	X	0	1	Bit 0 of the result set
X	X	1	0	One or more of bits 1-15 of the result set

(5) Compare Logical Halfword. The Compare Logical Halfword (CLHR, CLH, and CLHI) instructions shall cause the first operand specified by R1 to be compared logically to the 16-bit second operand. The result shall be indicated by the setting of the Condition Code PSW (12:15). Both operands shall remain unchanged. The logical comparison shall be performed by subtracting the second operand from the first operand. The result shall be in the Condition Code setting, and the operands shall not be modified. The Compare Halfword Immediate (CLHI) instruction shall produce a value which is the logical comparison of the address field itself plus the contents of a General Register index (X2) with the first operand general register (R1). In the RX format, the second operand shall be located on a halfword boundary. The resulting Condition Code shall be:

C	V	G	L	
X	X	0	0	First operand equal to second operand
X	X	0	1	First operand not equal to second operand
X	X	1	0	
1	X	X	X	First operand less than second operand
0	X	X	X	First operand equal to or greater than second operand

(6) Compare Logical Fullword Instructions. The compare logical fullword (CLDPR and CLDP) instructions cause the fullword first operand specified by R1, R1 + 1 to be compared logically to the fullword second operand. The result is indicated by the setting of the Condition Code PSW 12:15. Both operands remain unchanged. R1 and R2 specify even numbered registers. In the RX format, the second operand is located on a halfword boundary. The logical comparison is performed by subtracting the second operand from the first operand. The result is indicated in the Condition Code setting; the operands are not modified. The resulting Condition Code shall be:

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

(7) Compare Halfword. The Compare Halfword (CHR, CH, and CHI) instructions shall cause the first operand specified by R1 to be compared to the 16-bit second operand. The comparison shall be algebraic, taking into account the sign and magnitude of each number. The result shall be indicated by the setting of the Condition Code PSW (12:15). Both operands shall remain unchanged. In the RX format, the second operand shall be located on a halfword boundary. The Compare Halfword (CH) instructions, permit arithmetic comparison of signed two's complement 16-bit integers. The resulting Condition Code shall be:

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

(8) Compare Fullword Instructions. The compare fullword (CDPR and CPD) instructions cause the fullword first operand specified by R1, R1 + 1 to be compared to the fullword second operand. The comparison is algebraic, taking into account the sign and magnitude of each number. The result is indicated by the setting of the Condition Code PSW (12:15). Both operands remain unchanged. R1 and R2 specify even numbered registers. In the RX format, the second operand is located on a halfword boundary. The resulting Condition Code shall be:

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

e. Byte Handling Instructions. The item shall execute the Byte Handling instructions to provide for transferring bytes between memory and the general registers. All operands shall be 8-bit bytes. The Byte Handling instructions shall use the RR and RX formats. The exact format, op-code, assembler notation, and diagrammatic representation of each instruction shall be as shown in Figure VI-10. The operation and resulting Condition Code shall be as follows:

(1) Load Byte. The Load Byte (LB, LBR) instructions shall cause the 8-bit second operand to be loaded into the right-most (least significant) eight bits of the general register specified by R1. The left-most (most significant) eight bits of R1 shall be set to zero. The second operand shall remain unchanged. In the Load Byte Register (LBR) instruction, the second operand shall be taken from the least significant eight bits (Bits 8:15) of the register specified by R2. The Condition Code shall remain unchanged.

(2) Store Byte. The Store Byte (STB, STBR) instructions shall cause the right-most (least significant) 8-bit byte of the first operand to be stored in the general register or core memory location specified by the second operand. The first operand shall be unchanged. In the RR form of this instruction, the left-most byte of R2 (0:7) shall be unchanged, and the eight bit quantity shall be stored in bits 8:15 of the register specified by R2. The Condition Code shall remain unchanged.

(3) Exchange Byte. The Exchange Byte Register (EXBR) instruction shall cause the two 8-bit bytes of the second operand to be exchanged and loaded into the general register specified by R1. The contents of R2 shall remain unchanged. R1 and R2 may specify the same register. The Condition Code shall remain unchanged.

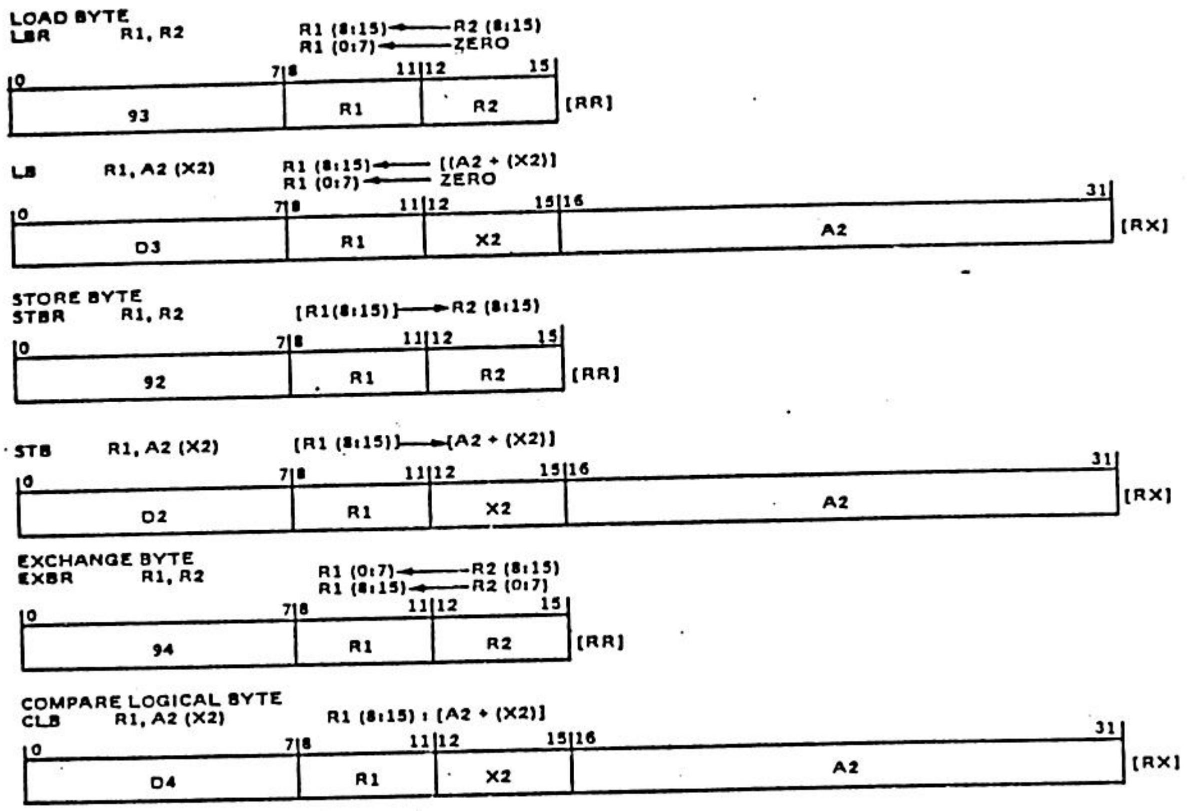


Figure VI-10. Byte Handling Instructions

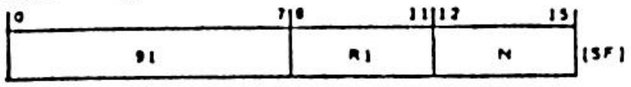
(4) Compare Logical Byte. The Compare Logical Byte (CLB) instruction shall cause the least significant 8-bit byte of the first operand to be logically compared to the 8-bit second operand. The result shall be indicated by the setting of the Condition Code PSW (12:15). Neither operand shall be changed. The resulting Condition Code shall be:

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

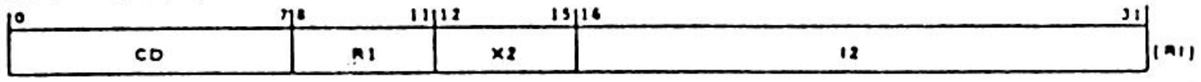
f. Shift/Rotate Instructions. The item shall execute the Shift/Rotate instructions to provide for arithmetic and logical manipulation of information contained in the general registers. Instructions for halfword and fullword operands shall be provided. Bits shifted out of the high or low order end of a general register shall be passed through the carry bit position of the Condition Code. After execution of a Shift instruction, the last bit which was shifted out shall be contained in the Carry position. The fullword Shift and Rotate instructions shall manipulate a pair of general registers. The R1 field of these instructions must specify an even-numbered register. The register specified shall contain the most significant sixteen bits of the fullword operand. The next sequential general register shall contain the least significant sixteen bits. A shift of zero positions shall cause the Condition Code to be set properly with no alteration to the information contained in the general register. The Shift/Rotate instructions shall use the SF and RI formats. The exact format, op-code, assembler notation, and diagrammatic representation of each instruction shall be as shown in Figures VI-11, -12, -13, -14, -15, and -16. The operation and resulting Condition Code shall be as follows:

(1) Shift Left Logical. The Shift Left Logical (SLL, SLLS, SLHL) instructions shall cause the content of the first operand to be shifted left the number of positions specified by the second operand. High order bits shifted out of Position 0 shall be shifted through the carry bit of the PSW and then lost. Zeros shall be shifted into the low order bit position. The last bit shifted shall remain in the carry bit. For the Shift Left Logical Short (SLLS) instruction, the N field (Bits 12-15) of the instruction shall specify the number of positions the content of R1 is to be shifted. For the Shift Left Halfword Logical (SLHL) instruction, only the low order four bits (12-15) of I2+(X2) shall be used for the shift count. The Shift Left Logical (SLL) instruction shall shift Registers R1 and R1+1, an even-odd pair. The R1 field of the instruction shall specify an even register. The shift count shall be specified by the low order five bits (11-15) of the value I2+(X2). The Carry shall be formed by the output of R1. The resulting Condition Code shall be:

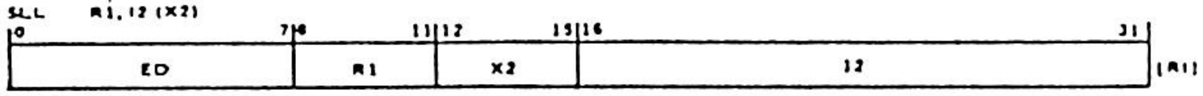
SHIFT LEFT LOGICAL
SLLS R1, N



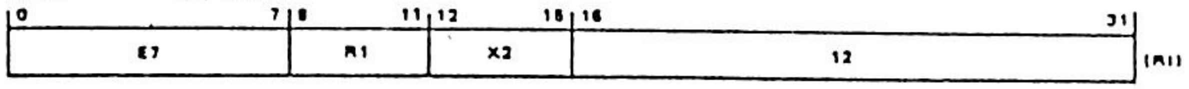
SLHL R1, I2 (X2)



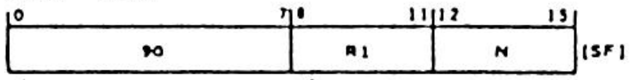
SLL R1, I2 (X2)



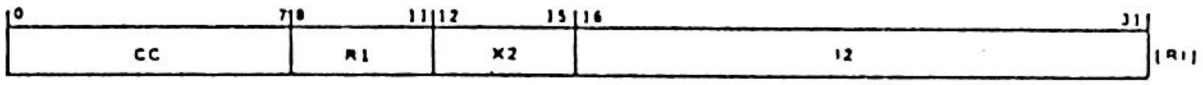
SHIFT LEFT DOUBLEWORD LOGICAL
SLDL R1, I2 (X2)



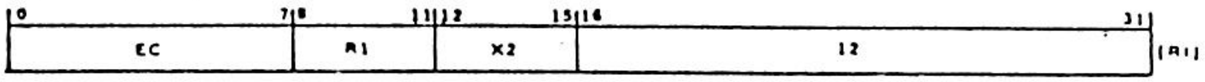
SHIFT RIGHT LOGICAL
SRLS R1, N



SRHL R1, I2 (X2)



SRL R1, I2(X2)



SHIFT RIGHT DOUBLE WORD LOGICAL
SRDL R1, I2 (X2)

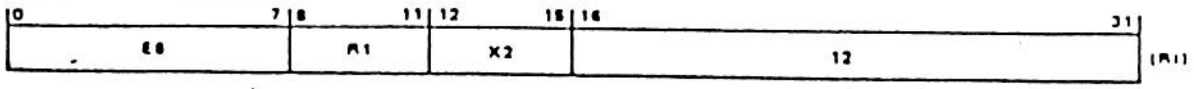


Figure VI-11. Logical Shift Instruction

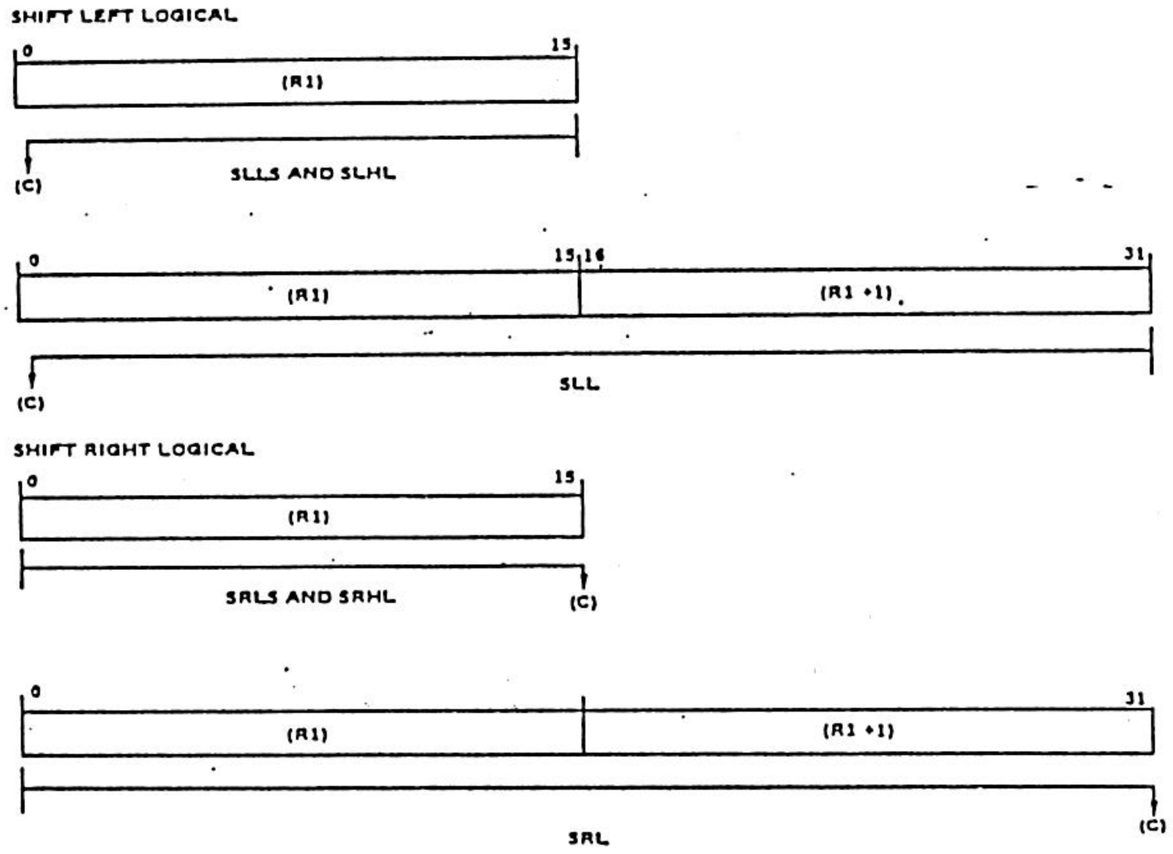


Figure VI-12. Logical Shifts Illustration

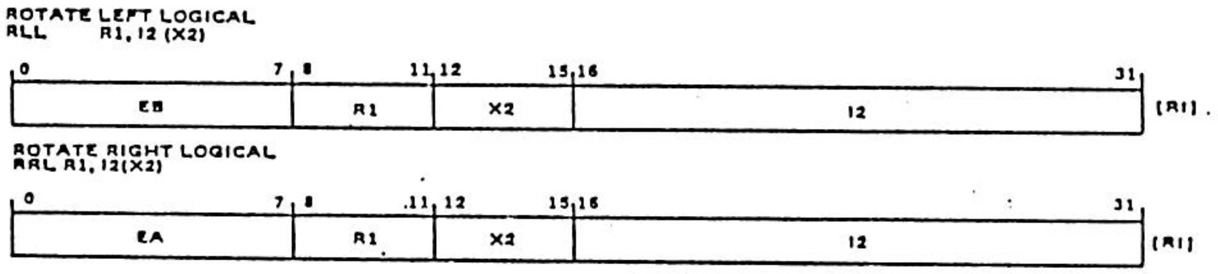


Figure VI-13. Logical Rotate Instructions

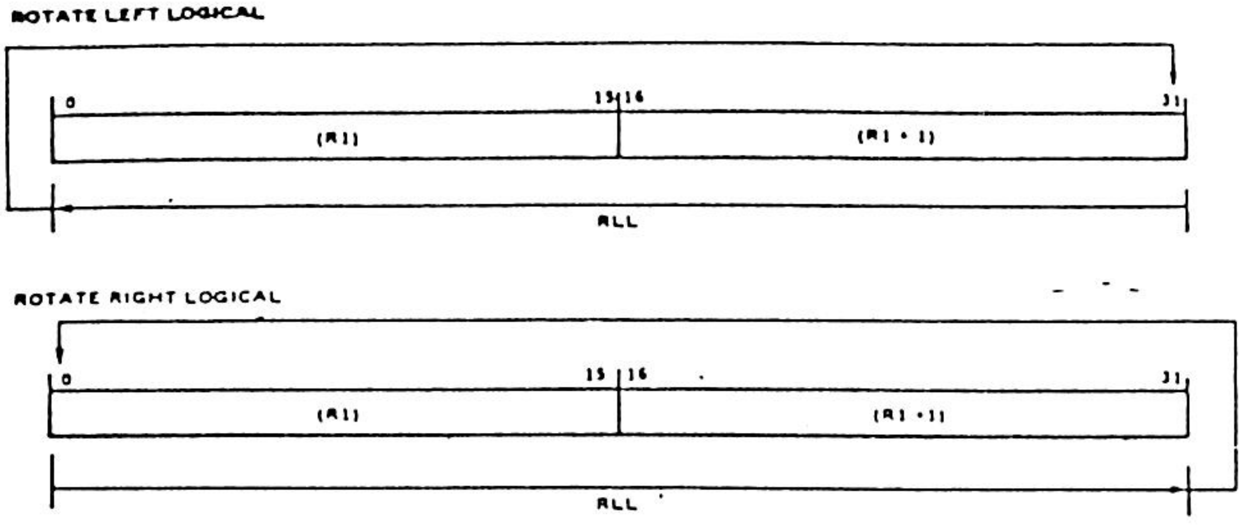


Figure VI-14. Logical Rotate Illustration

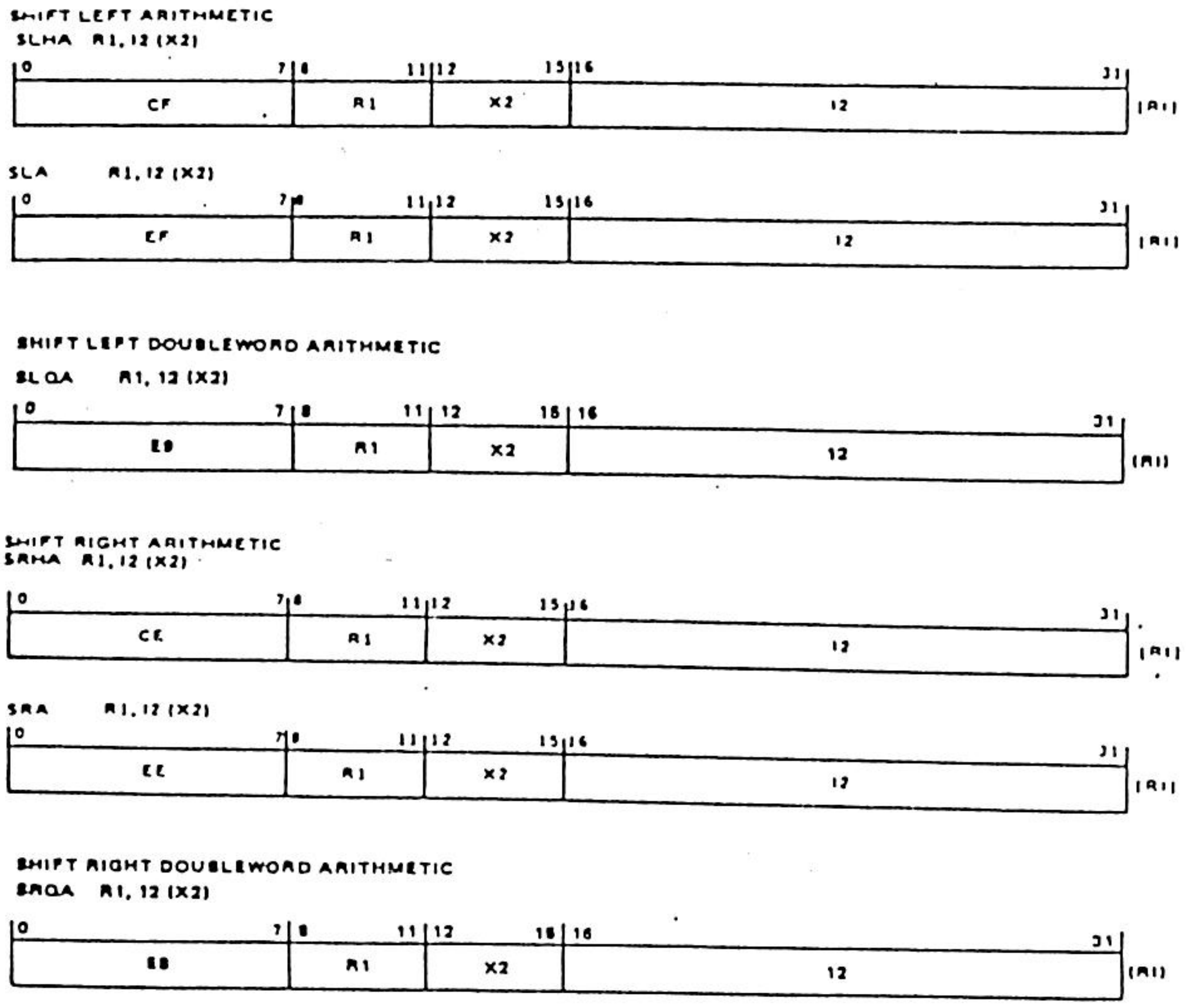


Figure VI-15. Arithmetic Shift Instructions

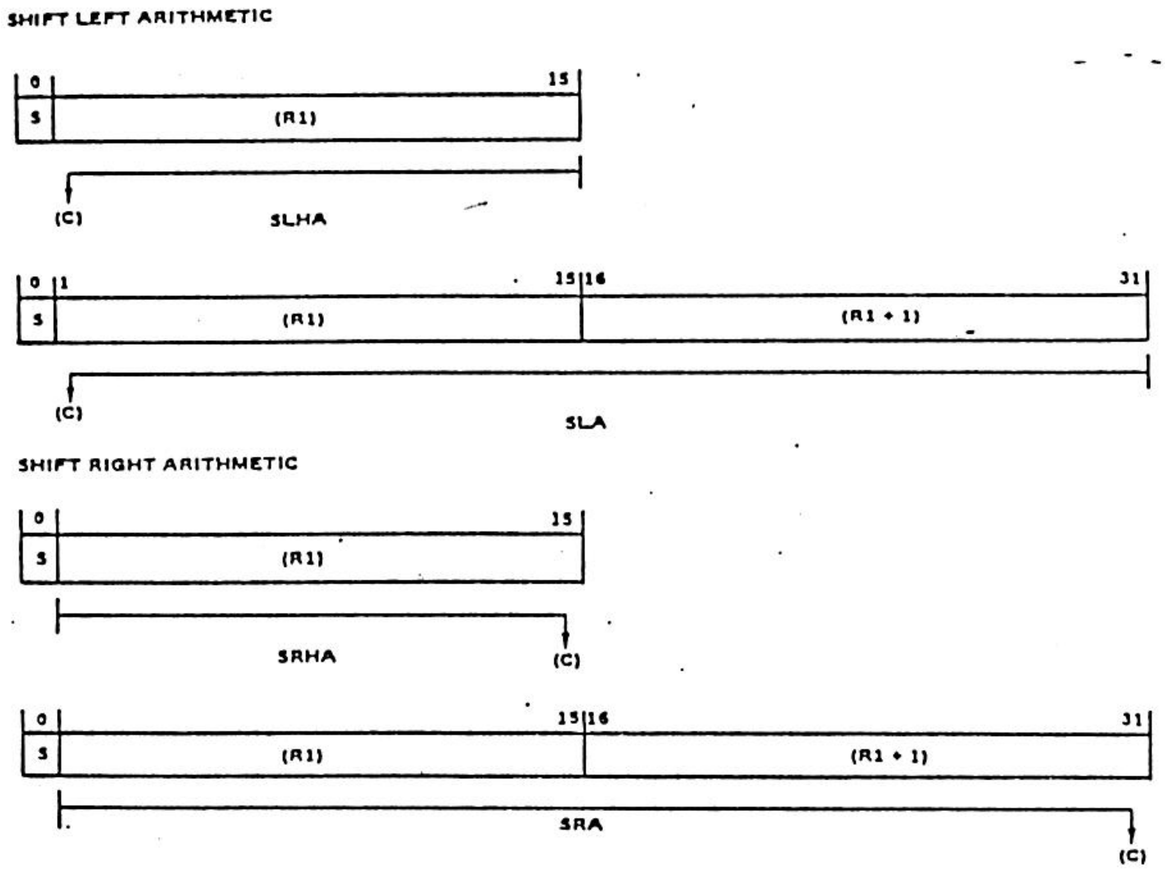


Figure VI-16. Arithmetic Shift Illustration

C	V	G	L	
	0	0	0	Result is zero
	0	0	1	Result is not zero
	0	1	0	Result is not zero
0				Last bit that was shifted out was a zero
1				Last bit that was shifted out was a one.

When the first operand is Fixed-Point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

(2) Shift Right Logical. The Shift Right Logical (SRL, SRLS, SRHL) instructions shall cause the content of the first operand to be shifted right the number of bit positions specified by the second operand. Low order bits shifted out of Position 15 (for the halfword instruction) or Position 31 (for the fullword instruction) shall be shifted through the carry bit of the PSW and then lost. Zeros shall be shifted into Position 0. The last bit shifted shall remain in the carry bit. For the Shift Right Logical Short (SRLS) instruction, the N field (Bits 12-15) of the instruction shall specify the number of positions the content of R1 is to be shifted. For the Shift Right Halfword Logical instruction, only the low order four bits (12-15) of I2+(X2) shall be used for the shift count. The Shift Right Logical (SRL) instruction shall shift Registers R1 and R1+1, an even-odd pair. The R1 field of the instruction shall specify an even register. The shift count shall be specified by the low order five bits (11-15) of the value I2+(X2). The Carry shall be formed by the output of R1. The resulting Condition Code shall be:

C	V	G	L	
	0	0	0	Result is zero
	0	0	1	Result is not zero
	0	1	0	Result is not zero
0				Last bit that was shifted out was a zero
1				Last bit that was shifted out was a one.

(3) Rotate Left Logical. The Rotate Left Logical (RLL) instruction shall cause the 32-bit first operand specified by R1 to be shifted left, end around, the number of positions specified by the low order five bits of the value I2+(X2). All thirty-two bits of the fullword shall be shifted. Bits shifted out of Position 0 shall be shifted into Position 31. A shift specification of sixteen bits shall interchange the two halves (R1, R1+1) of the first operand. The Rotate Left Logical

instruction shall rotate Registers R1 and R1+1, an even-odd pair. The R1 field of the instruction shall specify an even register. The resulting Condition Code shall be:

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

(4) Rotate Right Logical. The Rotate Right Logical (RRL) instruction shall cause the 32-bit first operand specified by R1 to be shifted right, end around, the number of positions specified by the low order five bits of the value I2+(X2). All thirty-two bits of the fullword shall be shifted. Bits shifted out of Position 31 shall be shifted into Position 0. A shift specification of sixteen bits shall interchange the two halves (R1, R1+1) of the first operand. The Rotate Right Logical instruction rotates Registers R1 and R1+1, an even-odd pair. The R1 field of the instructions shall specify an even register. The resulting Condition Code shall be:

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

(5) Shift Left Arithmetic. The Shift Left Arithmetic (SLA, SLHA) instruction shall cause the content of the first operand to be shifted left the number of bit positions specified by the second operand. The Sign Bit shall be unchanged. High order bits shifted out of Position 1 shall be shifted through the carry bit of the PSW and then lost. Zeros shall be shifted into the low order bit position. For the Shift Left Halfword Arithmetic (SLHA) instruction, the shift count shall be specified by the low order four bits (12-15) of the value of I2+(X2). The Shift Left Arithmetic (SLA) instruction shall shift Registers R1 and R1+1, an even-odd pair. R1 shall specify an even register. The shift count shall be specified by the low order five bits (11-15) of the value of I2+(X2). The resulting Condition Code shall be:

C	V	G	L	
	0	0	0	Result is zero
	0	0	1	Result is less than zero
	0	1	0	Result is greater than zero
0				Last bit that was shifted out was a zero
1				Last bit that was shifted out was a one.

(6) Shift Right Arithmetic. The Shift Right Arithmetic (SRA, SRHA) instruction shall cause the content of the first operand to be shifted right the number of bit positions specified by the second operand. The Sign Bit (Bit 0 of R1) shall be unchanged and shall be shifted right into Bit 1; therefore, Bit 0 shall be propagated right as many positions as specified by the second operand. Low order bits of the first operand shall be shifted through the carry bit of the PSW and then lost. For the Shift Right Halfword Arithmetic (SRHA) instruction, the shift count shall be specified by the low order four bits (12-15) of the value of I2+(X2). The Shift Right Arithmetic (SRA) instruction, shall shift Registers R1 and R1+1, an even-odd pair. R1 shall specify an even register. The shift count shall be specified by the low order five bits (11-15) of the value of I2+(X2). The Carry shall be formed by the output of R1+1 instead of R1. The resulting Condition Code shall be:

C	V	G	L	
	0	0	0	Result is zero
	0	0	1	Result is less than zero
	0	1	0	Result is greater than zero
0				Last bit that was shifted out was a zero
1				Last bit that was shifted out was a one.

(7) Shift Left Double Word Logical Instruction. The shift left double word logical (SLQL) instruction causes the content of the first operand to be shifted left the number of positions specified by the second operand. High order bits shifted out of position 0 are shifted through the carry bit of the PSW and then lost. Zeros are shifted into the low order bit position. The last bit shifted remains in the carry bit. The R1 field of the instruction specifies an even register. The shift count is specified by the low order six bits (10 thru 15) of the value I2 + (X2). The resulting Condition Code shall be:

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
0	X	X	X	Last bit that was shifted out was a zero
1	X	X	X	Last bit that was shifted out was a one.

(8) Shift Right Double Word Logical Instruction. The shift right double word logical (SRQL) instruction causes the content of the first operand to be shifted right the number of positions specified by the second operand. Low order bits shifted out of position 63 are shifted through the carry bit of the PSW and then lost. Zeros are shifted into position 0. The last bit shifted remains in the carry bit. The R1 field of the instruction specifies an even register. The shift count is specified by the low order six bits (10 thru 15) of the value I2 + (X2). The resulting Condition Code shall be:

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
0	X	X	X	Last bit that was shifted out was a zero
1	X	X	X	Last bit that was shifted out was a one.

(9) Shift Left Double Word Arithmetic Instruction. The shift left double word arithmetic (SLQA) instruction causes the content of the first operand to be shifted left the number of bit positions specified by the second operand. High order bits shifted out of position 1 are shifted through the carry bit of the PSW and then lost. Zeros are shifted into the low order bit position. The R1 field of the instruction specifies an even register. The shift count is specified by the low order six bits (10 thru 15) of the value I2 + (X2). The resulting Condition Code shall be:

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
0	X	X	X	Last bit that was shifted out was a zero
1	X	X	X	Last bit that was shifted out was a one.

(10) Shift Right Double Word Arithmetic Instruction. The shift right double word arithmetic (SRQA) instruction causes the content of the first operand to be shifted right the number of bit positions specified by the second operand. The sign bit (bit 0) of R1 remains unchanged and is shifted right into bit 1; therefore, bit 0 is propagated right as many positions as specified by the second operand. Low order bits of the first operand are shifted through the carry bit of the PSW and then lost. The R1 field of the instruction specifies an even register. The shift count is specified by the low order six bits (10 thru 15) of the value I2 + (X2). The carry is formed by the output of R1 + 3. The resulting Condition Code shall be:

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
0	X	X	X	Last bit that was shifted out was a zero
1	X	X	X	Last bit that was shifted out was a one.

g. Branch Instructions. The item shall execute the Branch instructions to provide programmed decisions for entry to subprograms, as well as testing the result of arithmetic, logical, or indexing operations. Many processor operations result in setting of the Condition Code in the PSW. The Branch on Condition instructions shall implement the testing of the Condition Code through the use of a mask field contained in the instruction itself (M1 field). The Branch instructions shall use the RX, RR, and SF formats. The exact format, op-code, assembler notation, and diagrammatic representations of each instruction shall be as shown in Figures VI-17, -18, and -19. The operation and resulting Condition Code shall be as follows:

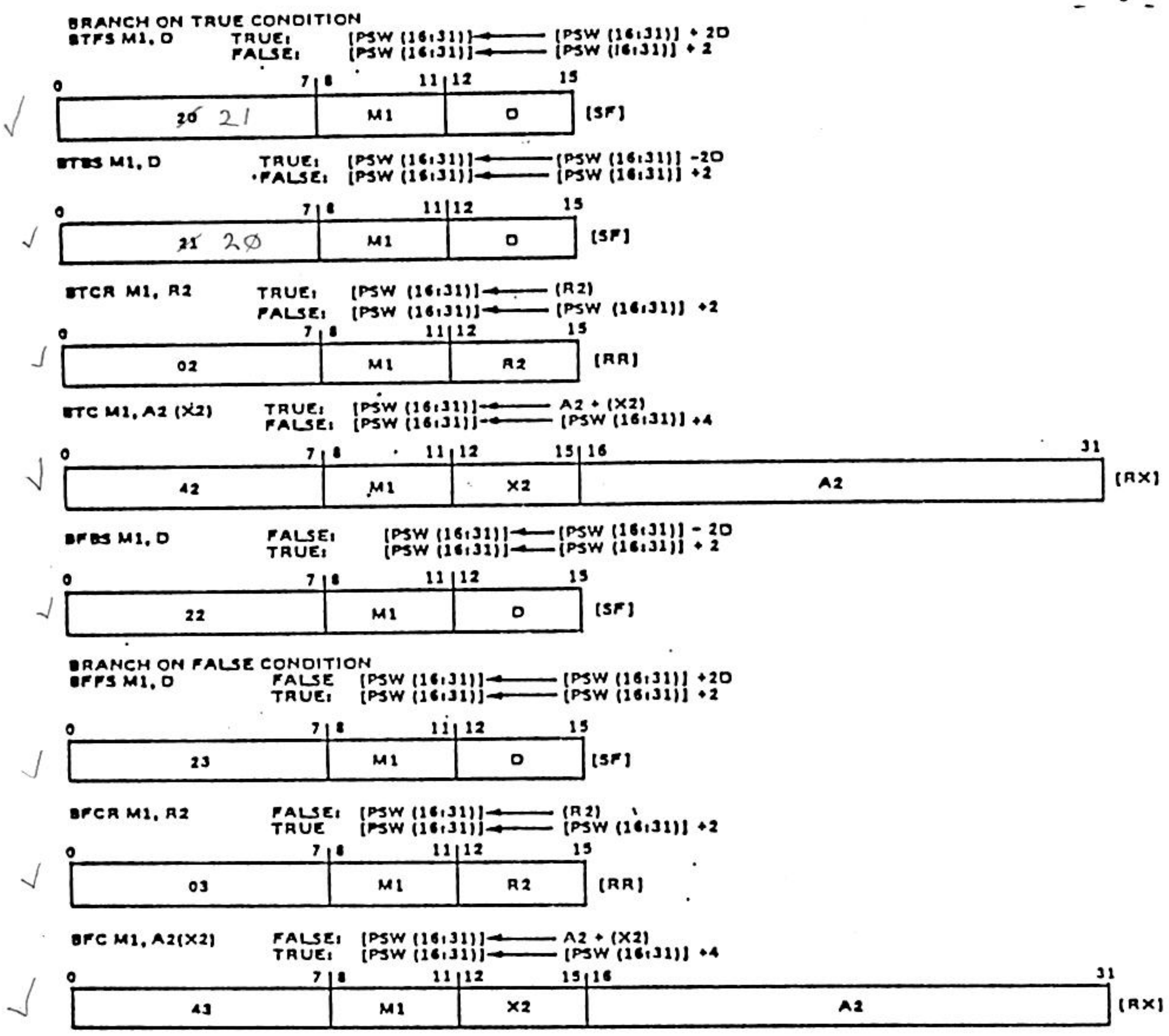

BRANCH ON TRUE CONDITION
BTFS M1, D TRUE: [PSW (16:31)] ← [PSW (16:31)] + 2D
FALSE: [PSW (16:31)] ← [PSW (16:31)] + 2
0 7 8 11 12 15
~~20~~ 21 M1 D (SF)
BTBS M1, D TRUE: [PSW (16:31)] ← [PSW (16:31)] -2D
FALSE: [PSW (16:31)] ← [PSW (16:31)] +2
0 7 8 11 12 15
~~21~~ 2∅ M1 D (SF)
BTCR M1, R2 TRUE: [PSW (16:31)] ← (R2)
FALSE: [PSW (16:31)] ← [PSW (16:31)] +2
0 7 8 11 12 15
02 M1 R2 (RR)
BTC M1, A2 (X2) TRUE: [PSW (16:31)] ← A2 + (X2)
FALSE: [PSW (16:31)] ← [PSW (16:31)] +4
0 7 8 11 12 15 16 31
42 M1 X2 A2 (RX)
BFBS M1, D FALSE: [PSW (16:31)] ← [PSW (16:31)] - 2D
TRUE: [PSW (16:31)] ← [PSW (16:31)] + 2
0 7 8 11 12 15
22 M1 D (SF)
BRANCH ON FALSE CONDITION
BFFS M1, D FALSE [PSW (16:31)] ← [PSW (16:31)] +2D
TRUE: [PSW (16:31)] ← [PSW (16:31)] +2
0 7 8 11 12 15
23 M1 D (SF)
BFCR M1, R2 FALSE: [PSW (16:31)] ← (R2)
TRUE [PSW (16:31)] ← [PSW (16:31)] +2
0 7 8 11 12 15
03 M1 R2 (RR)
BFC M1, A2(X2) FALSE: [PSW (16:31)] ← A2 + (X2)
TRUE: [PSW (16:31)] ← [PSW (16:31)] +4
0 7 8 11 12 15 16 31
43 M1 X2 A2 (RX)
√
√
√
√
√
√
√
√

Figure VI-17. Branch on True/False Instructions

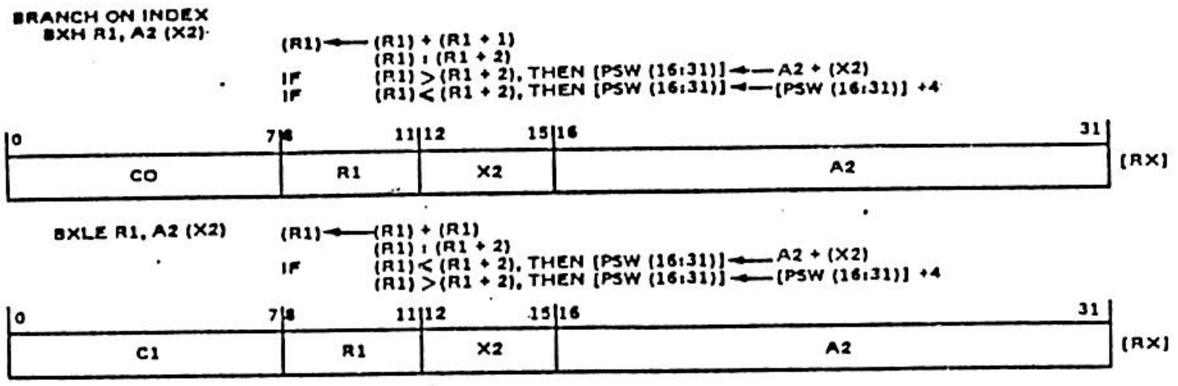

BRANCH ON INDEX
BXH R1, A2 (X2)
(R1) ← (R1) + (R1 + 1)
(R1) : (R1 + 2)
IF (R1) > (R1 + 2), THEN [PSW (16:31)] ← A2 + (X2)
IF (R1) < (R1 + 2), THEN [PSW (16:31)] ← [PSW (16:31)] +4
0
7
8
11
12
15
16
31
CO
R1
X2
A2
[RX]
BXLE R1, A2 (X2)
(R1) ← (R1) + (R1)
(R1) : (R1 + 2)
IF (R1) < (R1 + 2), THEN [PSW (16:31)] ← A2 + (X2)
(R1) > (R1 + 2), THEN [PSW (16:31)] ← [PSW (16:31)] +4
C1
[RX]

Figure VI-18. Branch on Index Instructions

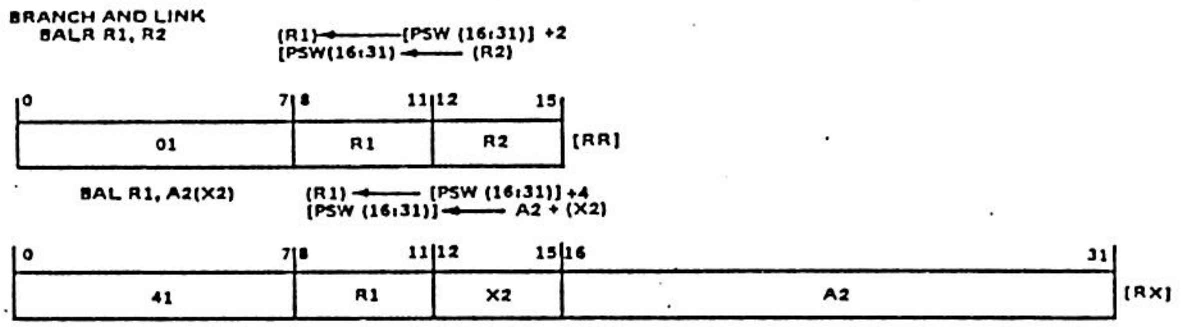


Figure VI-19. Branch and Link Instructions

(1) Branch on True. The Branch on True (BTBS, BTFS, BTCR, BTC) instructions shall cause the Condition Code field of the PSW (12:15) to be tested for the condition specified by the Mask Field (M1). If any of the conditions tested are found to be true, a Branch shall be executed to the 16-bit address specified by the second operand. If none of the conditions tested are found to be true, the next sequential instruction shall be executed. A logical AND shall be performed between each bit in the Condition Code and ist corresponding bit in the M1 field. If any resultant bit is a one, the Branch shall occur. The Condition Code [PSW (12:15)] shall not be changed. For example, if the Condition Code is 1010 and the M1 field is 1000, the Branch occurs with Branch on True instructions. The Branch on True Backward Short (BTBS) instruction shall cause a Branch to an address relative to the present Location Counter when the tested condition is true. The displacement shall be specified by the N field (Bits 12-15) of the instruction. The N field (times two) shall be subtracted from the present Location Counter to generate the address of the next instruction. The Branch on True Forward Short (BTFS) instruction shall cause a Branch to an address relative to the present Location Counter when the tested condition is true. The displacement shall be specified by the N field (Bits 12_15) of the instruction. The N field (times two) shall be added to the present Location Counter to generate the address of the next instruction. The Short Branch instructions (e.g., BTBS) are appropriate for Branches which specify small displacements from the present Location Counter (i.e., in sense status loops used for program controlled I/O).

(2) Branch on False. The Branch on False (BFBS, BFFS, BFCF, BFC) instructions shall cause the Condition Code field of the PSW (12:15) to be tested for the condition specified by the mask field (M1). If all conditions tested are found to be false, a Branch shall be executed to the 16-bit address specified by the second operand. If any of the conditions tested are found to be true, the next sequential instruction shall be executed. A logical AND shall be performed between each bit in the Condition Code and its corresponding bit in the M1 field. If any resultant bit is a one, the Branch shall not occur. The Condition Code [PSW (12:15)] shall not be change. For example, if the Condition Code is 1010 and the M1 field is 1100, the Branch does not occur with the Branch on False instruction. The Branch on False Backward Short (BFBS) instruction shall cause a Branch to an address relative to the present Location Counter shen the tested condition is false. The displacement shall be specified by the Nfield (Bits 12-15) of the instruction. The N field (times two) shall be subtracted from the present Location Counter to generate the address of the next instruction. The Branch on False Forward Short (BFFS) instruction shall cause a Branch to an address relative to the present Location Counter when the tested condition is false. The displacement shall be specified by the N field (Bits 12-15) of the instruction. 1The N field (times two) shall be added to the present Location Counter to generate the address of the next instruction. Branck on False Condition with a mask of 0 shall be an Unconditional Branch.

(3) Branch on Index. The Branch on Index High (BLH) instruction and the Branch on Index Low or Equal (BXLE) instruction shall cause the index (R1) to be incremented by R1+1, and logically compared to the index limit (R1+2). Prior to execution of this instruction, the general register specified by the first operand (R1) shall contain a 16-bit starting value, R1+1 shall contain a 16-bit increment value, and R1+2 shall contain a 16-bit comparand (limit or final value). All values shall be signed. For the Branch on Index High instruction, the contents of R1+1 should be negative. As long as the index (R1) is greater than the limit (R1+2), the 16-bit address specified by the second operand shall be transferred to the instruction address field of the PSW (16:31). The next instruction executed shall be accessed from the location specified by the new instruction address. When the count is not greater than the index limit, the instruction following Branch on Index High shall be executed. For the Branch on Index Low or Equal instruction, the contents of R1+1 should be positive. As long as the index (R1) is equal to or less than the limit (R1+2), the 16-bit address specified by the second operand shall be transferred to the instruction address field of the PSW (16:31). The next instruction executed shall be accessed from the location specified by the new instruction address. When the count is greater than the limit, the instruction following Branch on Index Low shall be executed. The Branch on Index High and the Branch on Index Low instructions are appropriate for raped loop control, particularly when one or more of the instructions in the loop is indexed. General Register 13 is the maximum specification for the R1 field. The Condition Code shall remain unchanged.

(4) Branch and Link. The Branch and Link (BAL and BALR) instructions shall cause the address of the next sequential instruction to be saved in the general register specified by the first operand (R1), and an Unconditional Branch to be executed to the 16-bit address specified by the second operand. the effective second operand shall be derived before the contents of Register R1 are changed. The Branch and Link instruction may be used for entry into subprograms. It differs from the Branch Unconditional instructions in that the incremented Location Counter value is preserved in a specified general register to be used as the subprogram's exit address. Exit from the subprogram is effected by a Branch Unconditional instruction through the general register in which the exit address has been maintained. The effective second operand is derived before the contents of R1 are changed. The Condition Code shall remain unchanged.

h. Floating-Point Instructions. The item shall execute the Floating-Point instructions to provide for loading, storing, adding, subtracting, multiplying, dividing, and comparing of floating-point operands. In order to produce correct normalized results, the Arithmetic instructions require normalized floating-point operands. If the operands are not normalized (with the exception of the floating-point load instructions), the results of the instructions are undefined. The Floating-Point instructions shall normalize an unnormalized floating-point number. The Floating-Point instructions shall manipulate 32-bit operands. The R1 and R2 fields of the Floating-Point instructions shall specify floating-point registers. These floating-point registers shall be reserved-memory locations.

The Floating-Point instructions shall use the RR and RX formats. The exact format, op-code, assembler notation, and diagrammatic representation of each instruction shall be as shown in Figures VI-20 and -21. The operation and resulting Condition Code shall be as follows:

(1) Floating-Point Load. The Floating-Point Load (LE and LER) instructions shall cause the floating-point second operand to be normalized and placed in the floating-point register R1. During normalization, the fraction shall be shifted left four bits at a time until the most significant hexidecimal digit is not zero. The exponent shall be decremented by one for each shift required. Zeros shall be shifted into the least significant bit positions of the fraction. If the fraction is zero, a true zero shall be generated. The second operand shall remain unchanged. If normalization causes exponent underflow, the result shall be set to a true zero, and the Overflow (V) flag is set. In the event of exponent underflow, a Floating-Point Arithmetic Fault Interrupt shall occur (if enabled by Bit 5 of the PSW). The resulting Condition Code shall be:

C	V	G	L	
X	0	0	0	Zero
X	0	0	1	Less than zero
X	0	1	0	Greater than zero
X	1	0	0	Exponent underflow

(2) Floating-Point Store. The Floating-Point Store (STE) instruction shall cause the floating-point first operand to be placed in the memory location specified by A2+(X2). The first operand shall remain unchanged. The resulting Condition Code shall remain unchanged.

(3) Floating-Point Add. The Floating-Point Add (AE and AER) instructions shall cause the following to occur. The exponents of the two operands shall be compared. If the exponents differ, the fraction with the smaller exponent shall be shifted right four bits at a time, and its exponent shall be incremented by one for each shift intil the two exponents agree. The fractions shall then be added algebraically. If a carry results, the exponent of the sum shall be incremented by one; the fraction shall be shifted right four bits; and the carry shall be shifted into the most significant hex digit. If an exponent overflow results, the exponent and fraction of the result shall be set to the maximum value, and the Overflow (V) flag shall be set. The sign of the result shall not be affected by the overflow. If no carry results from the addition of the fractions, the sum shall be normalized. During normalization, the fraction shall be shifted left four bits at a time until the most significant hex digit is not zero. The exponent shall be decremented by one for each shift required. Zeros shall be shifted into the least significant bit positions of the fraction. If normalization causes exponent underflow, a true zero shall be generated, and the Overflow (V) flag is set. If a zero sum is

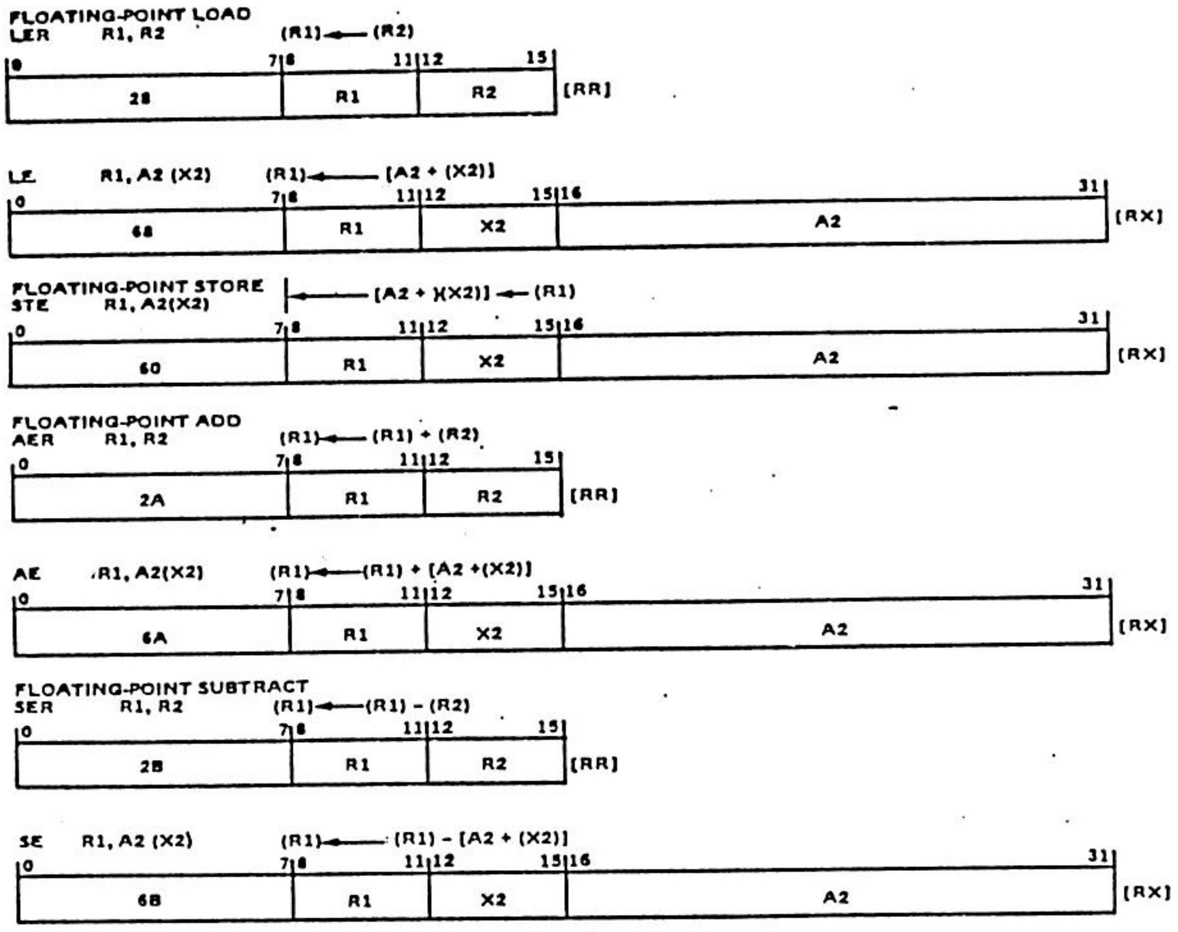


Figure VI-20. Floating-Point Load/Store/Add/Subtract Instructions

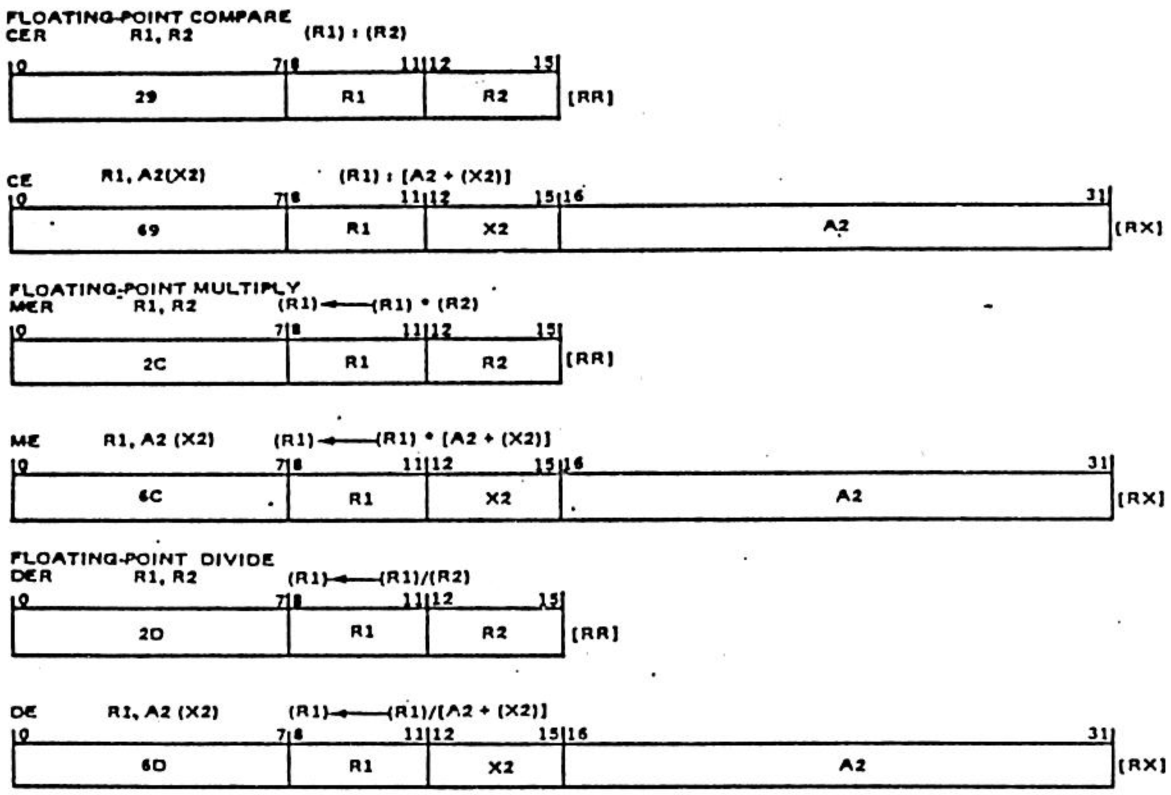


Figure VI-21. Floating-Point Compare/Multiply/Divide Instructions

generated by adding equal fractions with opposite signs, a true zero shall be generated. In the event of exponent overflow or underflow, a Floating-Point Arithmetic Fault Interrupt shall occur (if enabled by Bit 5 of the PSW). The resulting Condition Code shall be:

C	V	G	L	
X	0	0	0	Sum is zero
X	0	0	1	Sum is less than zero
X	0	1	0	Sum is greater than zero
X	1	0	1	Exponent Overflow (negative)
X	1	1	0	Exponent Overflow (positive)
X	1	0	0	Exponent underflow

(4) Floating-Point Subtract. The Floating-Point Subtract (SE and SER) instructions shall cause the following to occur. 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 agree. The fractions are then subtracted algebraically. If a Carry results, the exponent of the difference is incremented by one, and the fraction (result) is shifted right one hexadecimal position (four bits). The Carry is shifted into the most significant hexadecimal digit of the fraction. If an exponent overflow occurs, the exponent and fraction of the result are set to all ones, and the Overflow flag is set. The sign of the result is not affected by the overflow. If no Carry results from the subtraction of fractions, the difference is normalized by shifting the fraction left hexadecimally four bits at a time until the most significant hexadecimal digit is not zero. The exponent is decremented by one for each hexadecimal shift required. Zeros are shifted into the least significant hexadecimal digit of the fraction. If the normalization causes exponent underflow, the entire floating-point result is set to zero, and the Overflow flag is set. In the event of exponent overflow or underflow, a Floating-Point Arithmetic Fault Interrupt shall occur (if enabled by Bit 5 of the PSW). The resulting Condition Code shall be:

C	V	G	L	
X	0	0	0	Difference is zero
X	0	0	1	Difference is less than zero
X	0	1	0	Difference is greater than zero
X	1	0	1	Exponent Overflow (negative)
X	1	1	0	Exponent Overflow (positive)
X	1	0	0	Exponent underflow

(5) Floating-Point Compare. The Floating-Point Compare (CE and CER) instructions shall cause the first operand to be compared to the second operand. The comparison shall be algegraic, taking into account the sign, fraction, and exponent of each operand. Both operands shall remain unchanged. The result shall be indicated by the setting of the Condition Code. The resulting Condition Code shall be:

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

(6) Floating-Point Multiply. The Floating-Point Multiply (ME and MER) instructions shall cause the following to occur. The exponents of the two operands are added to produce the exponent of the result. The resultant exponent is readjusted to excess 64 notation. If an exponent overflow occurs, the exponent and fraction of the product are set to ones, and the Overflow flag is set. The sign of the product is determined by the rules of algebra. If an exponent underflow occurs, the entire floating-point result.is set to zero, and the Overflow flag is set. In either event, the Floating-Point Arithmetic Fault shall occur (if enabled by Bit 5 of the PSW). If an exponent overflow or underflow does not occur, the multiplication takes place. If the product is zero, the entire floating-point result is zero. If the result is not zero, normalization may occur. During nomralization, the fraction is shifted left hexadecimally four bits at a time until the most significant hexadecimal digit is not zero. The exponent of the result is decremented by one for each hexadecimal shift relquired. After normalization, the product is roundet to 24 bits. If normalization causes the exponent to underflow, the entire floating-point result is set to zero and the Overflow flag is set. The sum of the exponents of the two operands must be less than 64, or overflow occurs, producing the maximum possible value as a product (i.e., the multiplication $1/2x16^{63}*1 = 1/2x16^{63}*1/16x16^{1} = 1/32x16^{64}$ causes an overflow, rather than the result $1/2x16^{63}$). The resulting Condition Code shall be:

C	V	G	L	
		0	0	Product is zero
		0	1	Product is less than zero
		1	0	Product is greater than zero
	1	X	X	Exponent Overflow
	1	0	0	Exponent underflow

(7) Floating-Point Divide. The Floating-Point Divide (DE and DER) instructions shall cause the following to occur. The exponents of the two operands are subtracted to produce the exponent of the result. The resultant exponent is readjusted to excess 64 notation. If an exponent overflow occurs, the exponent and fraction of the quotient are set to all ones, and the Overflow flag is set. The sign of the quotient is determined by the rules of algebra. If an exponent overflow occurs, the entire floating-point result is set to zero, and the Overflow flag is set. If the divisor (the second operand) is zero, the operands are unchanged. In the event of exponent overflow, underflow, or division by zero, the Floating-Point Arithmetic Fault Interrupt shall occur (if enabled by Bit 5 of the PSW). If the exponent overflow or underflow does not occur, and if the divisor is not zero, the second operand is divided into the first operand. Division continues until the quotient is normalized, adjusting the exponent for each additional division required. If an exponent underflow occurs, the entire floating-point result is set to zero, and the Overflow flag is set. No remainder is returned to the user. The quotient is rounded to compensate for the loss of the remainder. Division by zero, overflow, or underflow cause a Floating-Point Arithmetic Fault Interrupt (if enabled by Bit 5 of the PSW). Inspection of the Condition Code of the Old PSW indicates the actual cause of the interrupt. If the Carry flag is set, then the divisor was zero. If the Carry flag is not set, then either overflow or underflow caused the interrupt. In this case, if the Greater than (G) or Less than (L) flag is set, the interrupt was caused by an overflow. If the G and L flag is reset, the interrupt was caused by an underflow. The difference of the exponents of the two operands must be less than 64, or overflow occurs, producing the maximum possible values as a quotient, even when normalization of the computed mantissa would bring the resultant exponent within range. The resulting Condition Code shall be:

C	V	G	L	
		0	0	Quotient is zero
		0	1	Quotient is less than zero
		1	0	Quotient is greater than zero
	1	1	X	Exponent Overflow
	1	X	1	Exponent Overflow
	1	0	0	Exponent underflow
1	1	0	0	Divisor Equal to zero

i. System Control Instructions. The item shall execute the System Control instructions to provide a means for the program to set the Program Status Word, swap PSWs, trigger special interrupt handling, and communicate with a supervisor program. Two instructions shall be provided to control the memory bank switching scheme in the item. These two instructions shall be included to extend the addressing range from 32,768 halfwords to 131,072 halfwords. Some of the System Control instructions are privileged and may be executed only with the processor in the Supervisor Mode (i.e., Bit 7 of the PSW reset). Any attempt to execute these instructions in the Protect Mode results in an Illegal Instruction Interrupt. The System Control instructions shall use the RR, SF, RX, and RI formats. The exact format, op-code, assembler notation, and diagrammatic representation of each instruction shall be as shown in Figure VI-22. The operation and resulting Condition Code shall be as follows:

(1) Load Program Status Word. The Load Program Status Word (LPSW) instruction shall cause a 32-bit operand to be loaded into the Current Program Status Word. The second operand shall remain unchanged. The resulting Condition Code shall be determined by the PSW loaded by the instruction. This instruction shall be privileged. The R1 field of a Load PSW instruction shall contain a zero.

(2) Exchange Operand Bank Address. The Exchange Operand Bank Address (EPOR) instruction shall cause the PSW (10:11) to be modified by the bits in the register specified by R1. The bits shall be exchanged between R1 and the PSW. R2 shall be ignored. The Condition Code shall remain unchanged.

(3) Exchange Program Address. The Exchange Program Address (EPPR) instruction shall cause the PSW (8:11 and 15:31) to be modified by the bits in the registers specified by R1 and R1+1. The former value of the PSW (0:31) shall be stored in R1 and R1+1. This instruction is useful in interbank transfers. R2 shall be ignored. The Condition Code shall remain unchanged.

(4) Exchange Program Status. The Exchange Program Status (EPSR) instruction shall cause the Current Program Status [PSW (0:15)] to be stored in the register specified by R1. The content of R2 shall then become the Current Program Status [PSW (0:15)]. Note that if R1=R2, this results in the Program Status being copied into R1, but otherwise remaining unchanged. This instruction is useful for capturing the running program status, enabling or disabling interrupts, or loading the Condition Code with a specified value. This instruction shall be privileged. The Condition Code shall be defined by the New PSW.

(5) Simulate Interrupt. The Simulate Interrupt (SINT) instruction shall cause the least significant eight bits of the second operand [I2+(X2)], to be presented to the Interrupt Handler (software) as a device number. The device number shall index into the Service Pointer Table at X'00D0' and result in either an Immediate Interrupt or an I/O Channel operation. This instruction shall be privileged. The R1 field of the Simulate Interrupt instruction shall contain a zero.

LOAD PROGRAM STATUS WORD

LPSW A2(X2) $[PSW (0:31)] \longleftarrow [A2 + (X2)]$

0 7	8 11	12 15	16 31	
C2	0	X2	A2	[RX]

EXCHANGE OPERAND BANK ADDRESS

EPOR R1, R2 $[PSW (10:11)] \longleftarrow [R1 (10:11)]$
$[R1 (00:15)] \longleftarrow$ FORMER $[PSW (0:15)]$

0 7	8 11	12 15	
2E	R1	R2	[RR]

EXCHANGE PROGRAM ADDRESS

EPPR R1, R2 $[PSW (8:11)] \longleftarrow R1 (8:11)$
$[PSW (16:31)] \longleftarrow R1 + 1$
$R1 \longleftarrow$ FORMER $[PSW (0:15)]$
$R1 + 1 \longleftarrow$ FORMER $[PSW (16:31)]$

0 7	8 11	12 15	
2F	R1	R2	[RR]

EXCHANGE PROGRAM STATUS

EPSR R1, R2 $[PSW (0:15)] \longrightarrow R1$
$[PSW (0:15) \longleftarrow R2$

0 7	8 11	12 15	
95	R1	R2	[RR].

SIMULATE INTERRUPT

SINT I2 (X2)

0 7	8 11	12 15	16 31	
E2	0	X2	I2	[RI]

SUPERVISOR CALL

SVC R1, I2 (X2) $(X'0094') \longleftarrow I2 + (X2)$
$(X'0096') \longleftarrow [PSW (0:31)]$
$(X'009A') \longrightarrow [PSW (0:15)]$
$(X'009C' + 2 * R1) \longrightarrow [PSW (16:31)]$

0 7	8 11	12 15	16 31	
E1	R1	X2	I2	[RI]

Figure VI-22. System Control Instructions

(6) Supervisor Call. The Supervisor Call (SVC) instruction shall provide a means for initiating software functions in the Executive program. The second operand address [A2+(X2)] shall serve as a pointer to the memory location of the parameters the Executive program will need to complete the function specified. The value [A2+(X2)] shall be stored in memory location X'0094'. The Current PSW shall be stored in the fullword memory location at X'0096'. Memory location X'009A' shall contain the New Program Status value. Memory locations X'009~~B~~C' through X'00BB' shall contain sixteen new Location Counter values, one for each type of Supervisor call. The type of Supervisor call shall be specified in the R1 field of the instruction. Sixteen different calls shall be provided for. Return from the Executive program shall be made by executing a Load PSW instruction specifying the stored "Old" PSW in location X'0096'. This instruction provides a convenient means of switching from the Protect Mode to the Supervisor Mode. Return to the Protect Mode is accomplished by a Load PSW or Exchange Program Status instruction. The resulting Condition Code shall be defined by the New PSW.

j. Input/Output Instructions. The item shall execute the Input/Output instructions to provide for the transfer of data between the Processor and the peripheral devices on the I/O Mux Bus. The Block I/O instructions shall provide for the transfer of blocks of data between the I/O device and memory. All of the instructions described in this section are privileged and, if executed with the processor in Protect Mode (PSW Bit 7 set), result in an Illegal Instruction Interrupt. Following I/O instructions, the V flag in the Condition Code shall indicate an instruction time-out. That is, due to an improper device response (either the addressed device does not exist, or it did not respond correctly), the specified I/O operation was not performed. An instruction time-out shall occur thirty microseconds after initiation of the I/O instruction if a synchronize signal has not been received in response to issuing a device address. A time-out shall cause the V flag to be set and the next instruction to be executed. Following Sense Status or Acknowledge Interrupt instructions, the Condition Code (CVGL) also reflects Bits four through seven of the device status. With standard Interdata device controllers, Bit five of the status byte (which is reflected in the V flag in the Condition Code) is defined as Examine Status. This means that the status byte should be examined. Therefore, following Sense Status and Acknowledge Interrupt instructions the occurrence of the V flag, with status Bits zero through three equal to zero, indicates instruction time-out. The I/O instructions shall use the RR and the RX formats. The exact format, op-code, assembler notation, and diagrammatic representation of each instruction shall be as shown in Figures VI-23, -24, -25, and -26. The operation and resulting Condition Code shall be as follows:

(1) Acknowledge Interrupt. The Acknowledge Interrupt (AI and AIR) instructions shall cause the address of the interrupting device to replace the content of the 16-bit general register specified by the first operand (R1). The 8-bit device status byte shall replace the content of the location specified by the second operand. The Condition Code shall be set equal to the right-most four bits in the device status byte. The device interrupt condition shall then be cleared. These instructions shall be privileged. The resulting Condition Code when the addressed device is a standard Interdata controller shall be:

ACKNOWLEDGE INTERRUPT
AIR R1, R2

[R1 (8:15)] ← DEVICE ADDRESS
[R1 (0:7)] ← ZERO
[R2 (8:15)] ← STATUS BYTE
[R2 (0:7)] ← ZERO
[PSW (12:15)] ← STATUS BYTE (4:7)

0 – 7	8 – 11	12 – 15	
9F	R1	R2	[RR]

AI R1, A2 (X2)

[R1 (8:15)] ← DEVICE ADDRESS
[R1 (0:7)] ← ZERO
[A2 + (X2)] ← STATUS BYTE
[PSW (12:15)] ← STATUS BYTE (4:7)

0 – 7	8 – 11	12 – 15	16 – 31	
DF	R1	X2	A2	[RX]

SENSE STATUS
SSR R1, R2

[R2 ((8:15)] ← STATUS BYTE
[R2 (0:7)] ← ZERO
[PSW (12:15)] ← STATUS BYTE (4:7)

0 – 7	8 – 11	12 – 15	
9D	R1	R2	[RR]

SS R1, A2 (X2)

[A2 + (X2)] ← STATUS BYTE
[PSW (12:15)] ← STATUS BYTE (4:7)

0 – 7	8 – 11	12 – 15	16 – 31	
DD	R1	X2	A2	[RX]

OUTPUT COMMAND
OCR R1, R2

DEVICE ← [R2 (8:15)]

0 – 7	8 – 11	12 – 15	
9E	R1	R2	[RR]

OC R1, A2(X2)

DEVICE ← [A2 + (X2)]

[RX]

0 – 7	8 – 11	12 – 15	16 – 31	
DE	R1	X2	A2	[RX]

Figure VI-23. Acknowledge/Status/Command Instructions

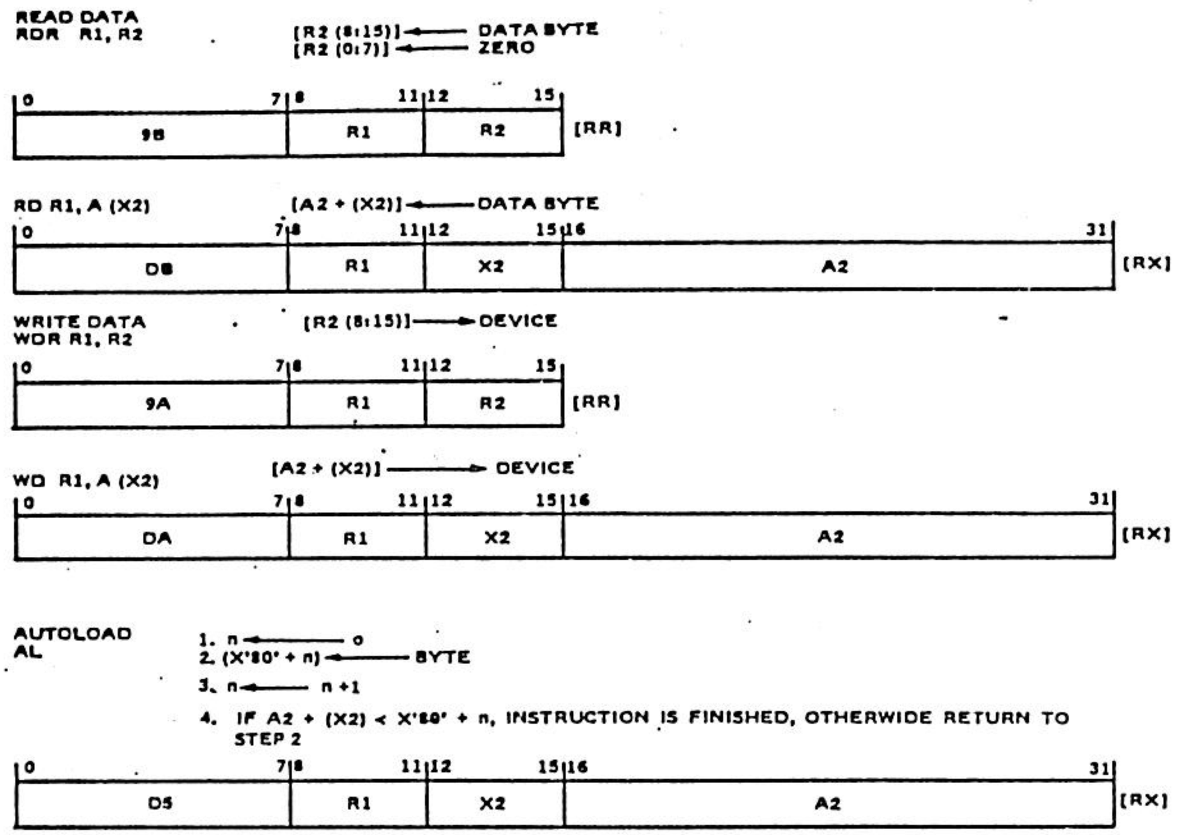


Figure VI-24. Byte I/O Instructions

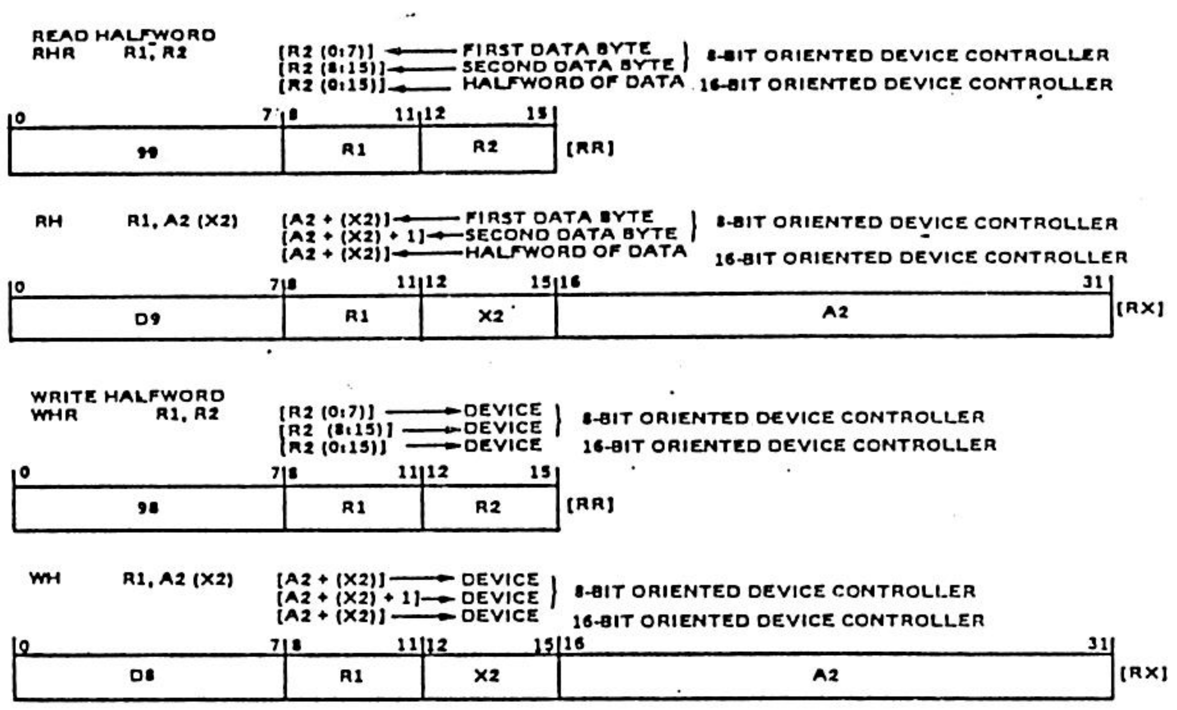
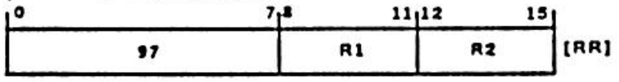

READ HALFWORD
RHR R1, R2
[R2 (0:7)] ← FIRST DATA BYTE
[R2 (8:15)] ← SECOND DATA BYTE
8-BIT ORIENTED DEVICE CONTROLLER
[R2 (0:15)] ← HALFWORD OF DATA
16-BIT ORIENTED DEVICE CONTROLLER
0 7 8 11 12 15
99 R1 R2 [RR]
RH R1, A2 (X2)
[A2 + (X2)] ← FIRST DATA BYTE
[A2 + (X2) + 1] ← SECOND DATA BYTE
8-BIT ORIENTED DEVICE CONTROLLER
[A2 + (X2)] ← HALFWORD OF DATA
16-BIT ORIENTED DEVICE CONTROLLER
0 7 8 11 12 15 16 31
D9 R1 X2 A2 [RX]
WRITE HALFWORD
WHR R1, R2
[R2 (0:7)] → DEVICE
[R2 (8:15)] → DEVICE
8-BIT ORIENTED DEVICE CONTROLLER
[R2 (0:15)] → DEVICE
16-BIT ORIENTED DEVICE CONTROLLER
0 7 8 11 12 15
98 R1 R2 [RR]
WH R1, A2 (X2)
[A2 + (X2)] → DEVICE
[A2 + (X2) + 1] → DEVICE
8-BIT ORIENTED DEVICE CONTROLLER
[A2 + (X2)] → DEVICE
16-BIT ORIENTED DEVICE CONTROLLER
0 7 8 11 12 15 16 31
D8 R1 X2 A2 [RX]

Figure VI-25. Halfword I/O Instructions

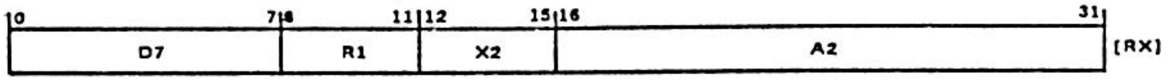
READ BLOCK
RBR   R1, R2

```
1. N = [A + (X2)]
2. IF N > [A + (X2) + 2]
   THEN TERMINATE WITH A CONDITION CODE = 0000
   ELSE:
       3. DEVICE ←— (N)
       4. N ←— N + 1
       5. RETURN TO STEP 2
```



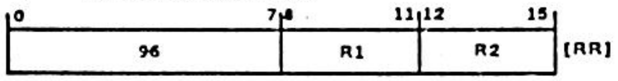
RB   R1, A2(X2)

```
1. N = (R2)
2. IF N > (R2 + 1),
   THEN: TERMINATE WITH A CONDITION CODE = 0000
   ELSE:
       3. DEVICE ←— (N)
       4. N = N + 1
       5. RETURN TO STEP 2
```



WRITE BLOCK
WBR   R1, R2

```
1. N = [A + (X2)]
2. IF N > [A + (X2) + 2],
   THEN: TERMINATE WITH A CONDITION CODE = 0000
   ELSE:
       3. (N) ←— DATA BYTE
       4. N ←— N + 1
       5. RETURN TO STEP 2
```



WB R1, A2(X2)

```
1. N ←— (R2)
2. IF N > (R2 + 1),
   THEN: TERMINATE WITH A CONDITION CODE = 0000
   ELSE:
       3. (N) ←— DATA BYTE
       4. N ←— N + 1
       5. RETURN TO STEP 2
```

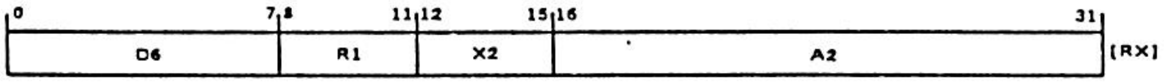


Figure VI-26. Block I/O Instructions

C	V	G	L	
1	0	0	0	Device busy (BSY)
0	1	0	0	Examine status (EX) or time-out
0	0	1	0	End of medium (EOM)
0	0	0	1	Device unavailable (DU)

(2) Sense Status. The Sense Status (SS and SSR) instructions shall provide a means for determining the status of an external I/O device. The 16-bit general recister specified by the first operand (R1) shall contain the device address. The device shall be addressed, and the 8-bit device status byte shall replace the content of the location specified by the second operand. The Condition Code shall be set equal to the right-most four bits of the device status byte. The first operand shall remain unchanged. These instructions shall be privileged. The resulting Condition Code when the addressed device is a standard Interdata controller shall be:

C	V	G	L	
1				Device busy (BSY)
	1			Examine status (EX) or time-out
		1		End of medium (EOM)
			1	Device unavailable (DU)

(3) Output Command. The Output Command (OC and OCR) instructions shall provide a means for commanding external I/O devices. The 16-bit general register specified by the first operand (R1) shall contain the device address. The device shall be addressed, and the 8-bit device command byte specified by the second operand shall be transmitted to the addressed device. Both operands shall remain unchanged. The overflow bit shall be set if the device can not complete the command action. These instructions shall be privileged.

(4) Read Data. The Read Data (RD and RDR) instructions shall address an external I/O device and input a byte of data. The 16-bit general register specified by the first operand (R1) shall contain the device address. The device shall be addressed, and a single 8-bit byte shall be transmitted from the device replacing the content of the location specified by the second operand. These instructions shall be privileged. These instructions should not be used with 16-bit oriented device controllers. For 16-bit oriented devices, use Read Halfword/Write Halfword instructions.

(5) Write Data. The Write Data (WD and WDR) instructions shall address an external I/O device and output a byte of data. The 16-bit general register specified by the first operand (R1) shall contain the device address. The device shall be addressed, and a single 8-bit byte shall be transmitted to the device. Both operands shall remain unchanged. These instructions shall be privileged. These instructions should not be used with 16-bit oriented device controllers. For 16-bit oriented devices, use Read Halfword/Write Halfword instructions.

(6) Autoload. The Autoload (AL) instruction shall load memory with a block of data from a byte oriented input device (e.g., teletype, photoelectric paper tape reader, magnetic tape, etc.). The data shall be read a byte at a time and stored in successive memory locations starting with location X'80'. The last byte shall be loaded into the memory location specified by the address of the second operand [A2+(X2)]. Any blank or zero bytes that are input prior to the first nonzero byte shall be considered to be leader and, therefore, ignored. All other zero bytes shall be stored as data. The input device shall be specified by memory location X'78'. The device command code shall be specified by memory location X'79'. This instruction shall be privileged. The R1 field of an Autoload machine instruction contains zero. This instructions should not be used with 16-bit oriented device controllers. For 16-bit oriented devices, use Read Halfword/Write Halfword instructions. The resulting Condition Code when the addressed device is a standard Interdata controller shall be:

C	V	G	L	
0	0	0	0	Data transfer completed correctly
1				Device busy (BSY)
	1			Examine status (EX) or time-out
		1		End of medium (EOM)
			1	Device unavailable (DU)

(7) Read Halfword. The Read Halfword (RH and RHR) instructions shall address an external I/O device and input a halfword of data. The 16-bit general register specified by R1 shall contain the device address. The device shall be addressed, and a 16-bit halfword shall be received from the device replacing the contents of the second operand. The Read Halfword instruction shall be implemented such that it can work with both 8-bit byte oriented device controllers and with 16-bit halfword oriented device controllers. If the controller is byte oriented, the processor shall input two eight bit bytes. If the controller is halfword oriented, the processor shall input one 16-bit halfword. These instructions shall be privileged. With the RX form (RH), the effective address [A2+(X2)] shall be an even value.

(8) Write Halfword. The Write Halfword (WH and WHR) instructions shall address an external I/O device and output a halfword of data. The 16-bit general register specified by R1 shall contain the device address. The device shall be addressed, and a 16-bit halfword shall be transmitted to the device from the location specified by the second operand. The Write Halfword instruction shall be implemented such that it can work with both 8-bit byte oriented device controllers and with 16-bit halfword oriented device controllers. If the controller is byte oriented, the processor shall output two eight bit bytes. If the controller is halfword oriented, the processor shall output one 16-bit halfword. The Read Halfword and Write Halfword instructions are useful with devices requiring two bytes per transfer. Since the transfer is accomplished with one instruction instead of two, both time and memory are saved. Some examples of devices with which these instructions can be used are Halfword I/O Module, 16-line Interrupt Module, conversion equipment (i.e., D/A and A/D Converters), Card Reader, and Control Panel. With the RX form (WH), the effective address [A2+(X2)] shall be an even value. These instructions shall be privileged.

(9) Read Block. The Read Block (RB and RBR) instructions shall address an external I/O device and input a series of data bytes. The 16-bit general register specified by the first operand (R1) shall contain the device address. The 16-bit second operand location [R2 or A2+(X2)] shall contain the starting address of the data buffer to be transferred. The next sequential halfword [(R2+1) or A2+(X2)+2)] shall contain the ending address of the data buffer. The starting address shall be equal to, or less than, the ending address. Data transfer shall be inclusive of the buffer limits. If the starting address is greater than the ending address, no transfer shall take place, and the instruction shall terminate with the Condition Code equal to zero. The Read Block instruction shall cause the transfer of eight bit data bytes from a device to consecutive memory locations. No other instructions shall be executed during transfer of the data block. The Condition Code portion of the PSW (12:15) shall be set to zero after a normal transfer. In the event of an abnormal block data transfer, the Condition Code shall not be zero. These instructions shall be privileged. These instructions should not be used with 16-bit oriented device controllers. For 16-bit oriented devices, use Read Halfword/Write Halfword instructions. For RBR, General Register 14 shall be the maximum specification for the R2 field. The resulting Condition Code when the addressed device is a standard Interdata controller shall be:

C	V	G	L	
0	0	0	0	Block data transfer completed correctly
1				Device busy (BSY)
	1			Examine status (EX) or time-out
		1		End of medium (EOM)
			1	Device unavailable (DU)

(10) Write Block. The Write Block (WB and WBR) instructions shall address an external I/O device and output a series of data bytes. The 16-bit general register specified by the first operand (R1) shall contain the device address. The 16-bit second operand location [R2 or A2+(X2)] shall contain the starting address of the data buffer to be transferred. The next sequential halfword [(R2+1) or A2+(X2)+2)] shall contain the ending address of the data buffer. The starting address shall be equal to, or less than, the ending address. Data transfer shall be inclusive of the buffer limits. If the starting address is greater than the ending address, no transfer shall take place, and the instruction shall terminate with the Condition Code equal to zero. The Write Block instruction shall cause the transfer of eight bit data bytes from consecutive memory locations to a device. No other instructions shall be executed during transfer of the data block. The Condition Code portion of the PSW (12:15) shall be set to zero after a normal transfer. In the event of an abnormal block data transfer, the Condition Code shall not be zero. These instructions shall be privileged. These instructions should not be used with 16-bit oriented device controllers. For 16-bit oriented devices, use Read Halfword/Write Halfword instructions. For WBR, General Register 14 shall be the maximum specification for the R2 field. The resulting Condition Code when the addressed device is a standard Interdata controller shall be:

C	V	G	L	
0	0	0	0	Block data transfer completed correctly
1				Device busy (BSY)
	1			Examine status (EX) or time-out
		1		End of medium (EOM)
			1	Device unavailable (DU)

k. List Processing Instructions. The item shall execute the List Processing instruction to manipulate a circular list as defined in Figure VI-27. The first two halfwords shall contain the list parameters. The list shall immediately follow the parameter block. The first halfword in the list shall be designated slot zero. The remaining slots shall be designated 1, 2, 3, etc., up to a maximum slot number which is equal to the number in the list minus one. An absolute maximum of 255 halfword slots shall be specifiable. The first paramater byte shall indicate the number of slots (halfwords) in the entire list. The second parameter byte shall indicate the current number of slots beirg used. When this byte equals zero, the list shall be empty; and when this byte equals the number of slots in the list, the list shall be full. Once initialized, this byte shall be maintained automatically. It shall be incremented when elements are added to the list and decremented when elements are removed. The third and fourth bytes of the list parameters shall specify the current top of the list and the next bottom of the list, respectively, as shown in Figure VI-28. These pointers shall also be updated automatically.

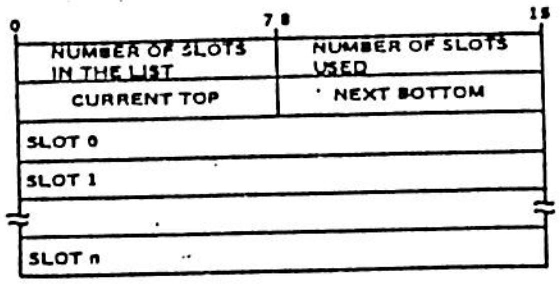


Figure VI-27. List Processing Instruction Format

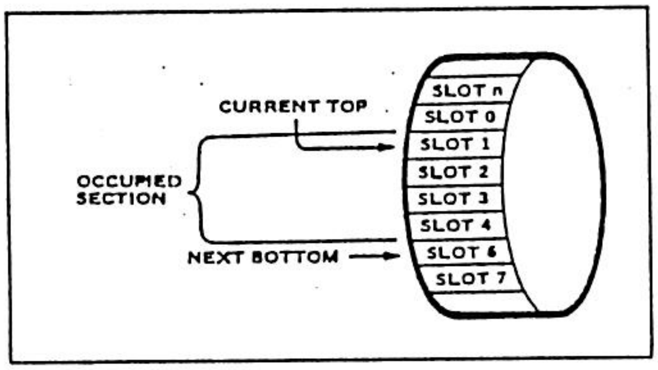


Figure VI-28. Circular List Instruction Processing

These instructions shall use the RX format. The exact format, op-code, assembler notation, and diagrammatic representation of each instruction shall be as shown in Figure VI-29. The operation and resulting Condition Code shall be as follows:

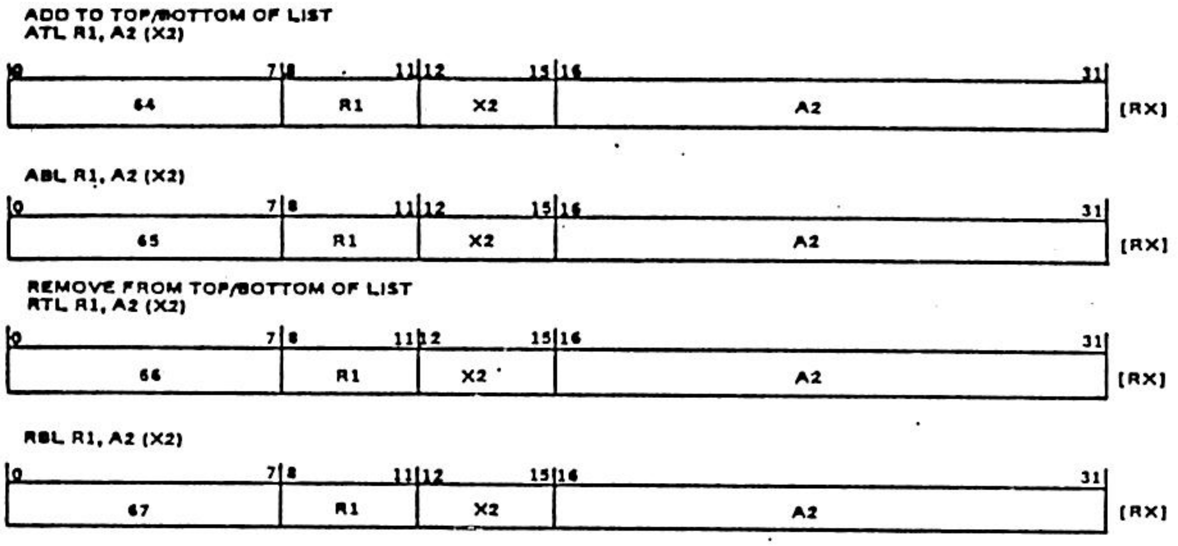


Figure VI-29. List Processing Instructions

(1) Add to Top/Bottom of List. The Add to Top of List (ATL) and Add to Bottom of List (ABL) instructions shall manipulate the list pointers and insert halfwords to the addressed list. The general register specified by R1 shall contain the element to be added to the list. The second operand [A2+(X2)] shall specify the address of the list. The number of slots used tally shall be compared to the number of slots in the list as specified by the first byte of the list. If the number of slots used tally is equal to the number of slots in the list, an overflow condition shall occur, and the element shall not be added to the list. Instead, the instruction shall be terminated with the V flag set in the PSW. If the number of slots used tally is less than the number of slots in the list, it shall be incremented by one; the appropriate pointer shall be changed; the element shall be added to the list; and the instruction shall be terminated with a Condition Code of zero. The ATL instruction shall manipulate the Current Top Pointer in the list. If no overflow occurred, the Current Top Pointer (which points to the last element added to the top of the list) shall be decremented by one, and the element inserted in the slot pointed to by the new Current Top Pointer. If the Current Top Pointer was zero on entering this instruction, the Current Top Pointer shall be set to the maximum slot number in the list. This condition shall be referred to as list wrap. The ABL instruction shall manipulate the Next Bottom Pointer. If no overflow occurred, the element shall be inserted in the slot pointed to by the Next Bottom Pointer, and the Next Bottom Pointer incremented by one. If the incremented Next Bottom Pointer is greater than the maximum slot number in the list, the Next Bottom Pointer shall be set to zero. This condition shall also be referred to as list wrap. The resulting Condition Code shall be:

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

(2) Remove From Top/Bottom of List. The Remove From Top of List (RTL) and Remove From Bottom of List (RBL) instructions shall manipulate the list pointers and remove halfwords from the addressed list. The element removed from the list shall be placed in the general register specified by R1. The second operand [A2+(X2)] shall specify the address of the list. If, on entering the instruction, the number of slots used tally is zero, the list is already empty and the instruction shall be terminated with the V flag set in the PSW. This condition shall be referred to as list underflow. If underflow does not occur, the number of slots used tally is decremented by one; the appropriate pointer shall be changed; and the element shall be extracted and placed in R1. The instruction shall be terminated with a Condition Code equal to zero if the list is now empty or with the G flag set if the list is not yet empty.

The RTL instruction shall manipulate the Current Top Pointer in the list. If no underflow occurred, the Current Top Pointer shall point to the element to be extracted. The element shall be extracted and placed in R1. The Current Top Pointer shall be incremented by one, and compared to the maximum slot number. If the Current Top Pointer is greater than the maximum slot number in the list, the Current Top Pointer shall be set to zero. This condition shall be referred to as list wrap. The RBL instruction shall manipulate the Next Bottom Pointer. If no overflow occurred, and the Next Bottom Pointer is zero, it shall be set to the maximum slot number in the list (list wrap). Otherwise, it shall be decremented by one, and the element now pointed to shall be extracted and placed in R1. The resulting Condition Code shall be:

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

1. Trigonometric Operation Word Formats. The trigonometric operation option instruction repertoire employs two different instruction word formats (register to register, RR; and register to indexed, RX) to perform the trigonometric function. The trigonometric operation option instruction repertoire consists of four instructions.

(1) Trigonometric Operation Data Word Formats. The trigonometric operation data word formats consist of fixed scalars, scalars, and angles (Figure VI-30).

(2) Trigonometric Data. The trigonometric instructions either operate on angles to produce fixed scalars or operate on scalars to produce angles, depending on the type of function (Table VI-1). Trigonometric data is handled with the following conventions:

(a) Fixed scalars are used to represent the sine or cosine of an angle and are defined as fixed-point numbers in the range of -1.0 to +1.0 inclusive. The raddix point is between bits one and two.

(b) Scalars may be arbitrary fixed-point quantities with the radix point anywhere within or outside the data word. The two scalar operands input to the trigonometric operation are in the same scale (their radix points must be in the same relative position).

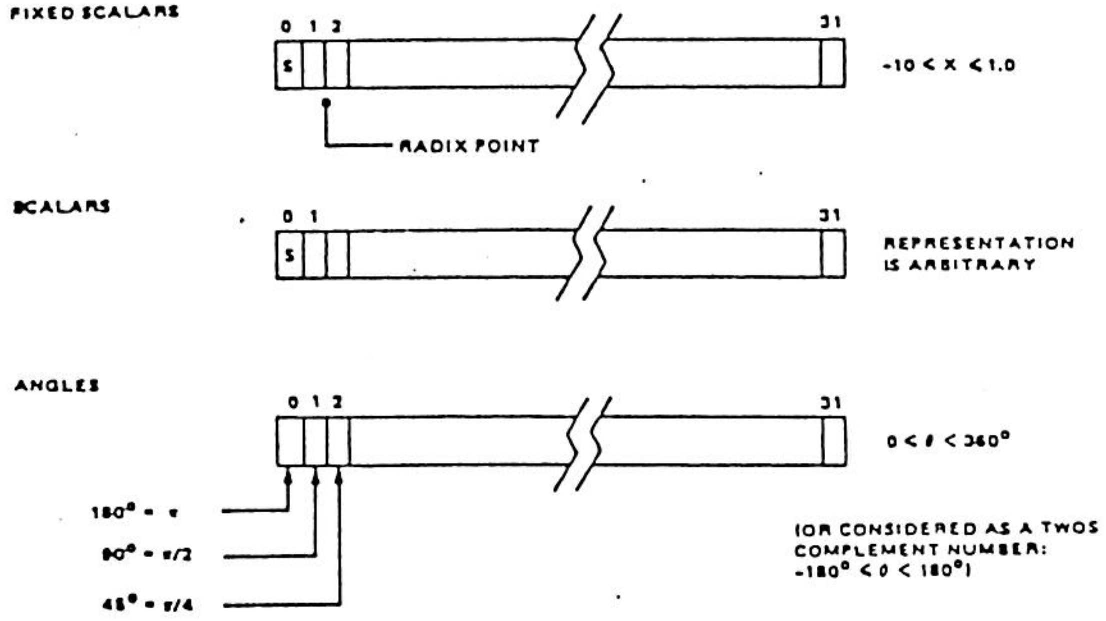


Figure VI-30. Trigonometric Data Formats

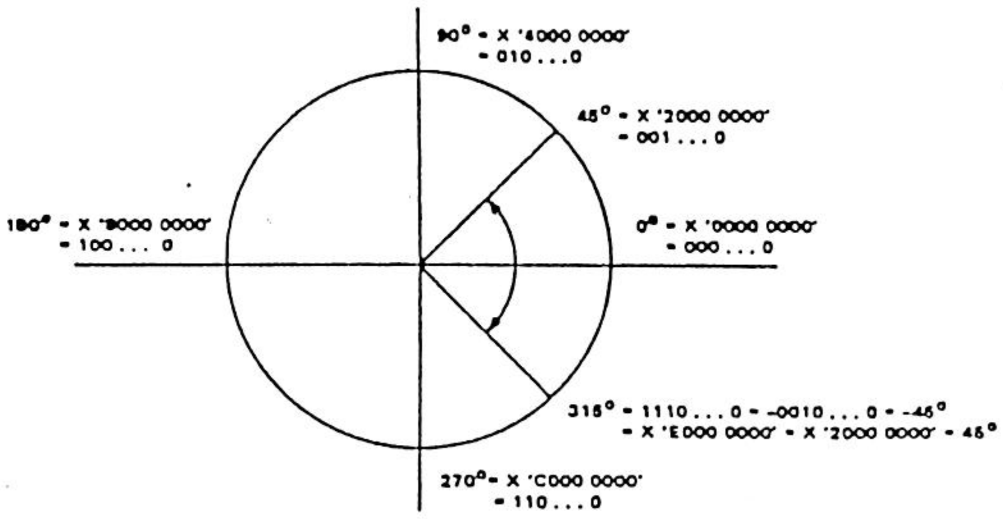


Figure VI-31. Binary Angular Measurement System (BAMS)

Table VI-1. Trigonometric Operands

FUNCTION	INPUT	OUPTUT
Cosine, Sine	Angle (BAMS)	Fixed Scalar
ARC-Sine	Scalar	Angle (BAMS)
ARC-Tangent, $\sqrt{x^2 + y^2}$	Scalar	Angle (BAMS) and Scalar
$\sqrt{x^2 - y^2}$	Scalar	Scalar

(c) Angles are represented in the binary angular measurement system (BAMS, Figure VI-31). Bit 0 represents an angle of 180 degrees (pi radians), bit 1 represents 90 degrees (pi/2 radians), bit 2 represents 45 degrees (pi/4 radians).

NOTE: Angles in this representation behave like twos complement numbers:

Example:

45 degrees = X'20000000' and -45 degrees = 315 degrees

= X'E0000000' = twos complement of X'20000000'

(3) Trigonometric Operation Instruction Word Format. The computer executes the trigonometric operations (Table VI-2). Figure VI-32 shows the format, op-code, assembler notation, and diagrammatic representation of each trigonometric instruction.

(4) CORDIC Implemented Instruction. Since the execution time of the trigonometric operation instructions vary significantly with the accuracy desired, the computer executes the CORDIC algorithm (under program control) to reduce instruction execution times. The precision of the results (of a trigonometric operation) is controlled by the value of the CORDIC precision register (CPR) contents, when a CORDIC implemented instruction is executed. The value of the CPR controls the precision of the results by determining the number of iterations of the CORDIC algorithm (permissable CPR values are 16, 20, 24, or 28). Table VI-3 shows the relationship between the number of iterations and number of accurate bits in the results for each CORDIC implemented instruction.

(5) Exchange CORDIC Precision Register Instruction. The exchange CORDIC precision register (ECPR) instruction provides control for reading and writing the CORDIC precision register (CPR). The contents of the CPR are saved, and R2 is used to load CPR with a new value. The previous value of CPR is placed in R1. Only the following values are permitted for assignment to CPR: 16, 20, 24, and 28. This is automatically guaranteed in the ECPR instruction by the computer masking bits 0 thru 10, 14, and 15 to zero, and forcing bit 11 to one, of the word used to specify the number of iterations. If R1 equals R2, the result of this instruction exchanges the contents of R1 with the contents of CPR. The resulting Conditional Code is not changed.

(6) Cosine/Sine Instructions. The cosine/sine (COS and COSR) instructions calculate both the sine and cosine of the second operand. The second operand represents an angle in BAMS. The cosine of this angle is calculated in fixed scalar format, and is stored in register pair R1, R1 + 1. The sine of the angle is calculated in fixed scalar format, and stored in register pair R1 + 2, R1 + 3, R1 (and R2) is even. R1 is less than or equal to 12. In COSR, R2 may utilize any of the quadruple of registers denoted by R1. The resulting Condition Code is not changed.

Table VI-2. Trigonometric Operation Instructions

Instruction	Mnemonic	Format	Execution time in microseconds(CPR=28)	Comments
Exchange CORDIC Precision Register (CPR)	ECPR	RR	1.4	- - -
Cosine, Sine	COSR	RR	51.0*	
	COS	RX	51.4*	
ARC-Sine	ASNR	RR	113.6/121.9/134.4**	min/avg/max
	ASN	RX	114.8/123.1/135.6*	min/avg/max
ARC-Tangent, $\sqrt{x^2 + y^2}$	ATN	RX	62.8/69.3/73.8*	min/avg/max
$\sqrt{x^2 - y^2}$	SQDR	RR	62.8/65.4/77.2*	min/avg/max
	SQD	RX	64.0/66.6/78.4*	min/avg/min

NOTES: * By reducing CPR to 24, 20, or 16, these execution times can be decreased. For each reduction of four bits in accuracy, the execution time decreases by 6.4 microseconds.

* By reducing CPR to 24, 20, or 16, these execution times can be decreased. For each reduction of four bits in accuracy, the execution time decreases by 12.8 microseconds.

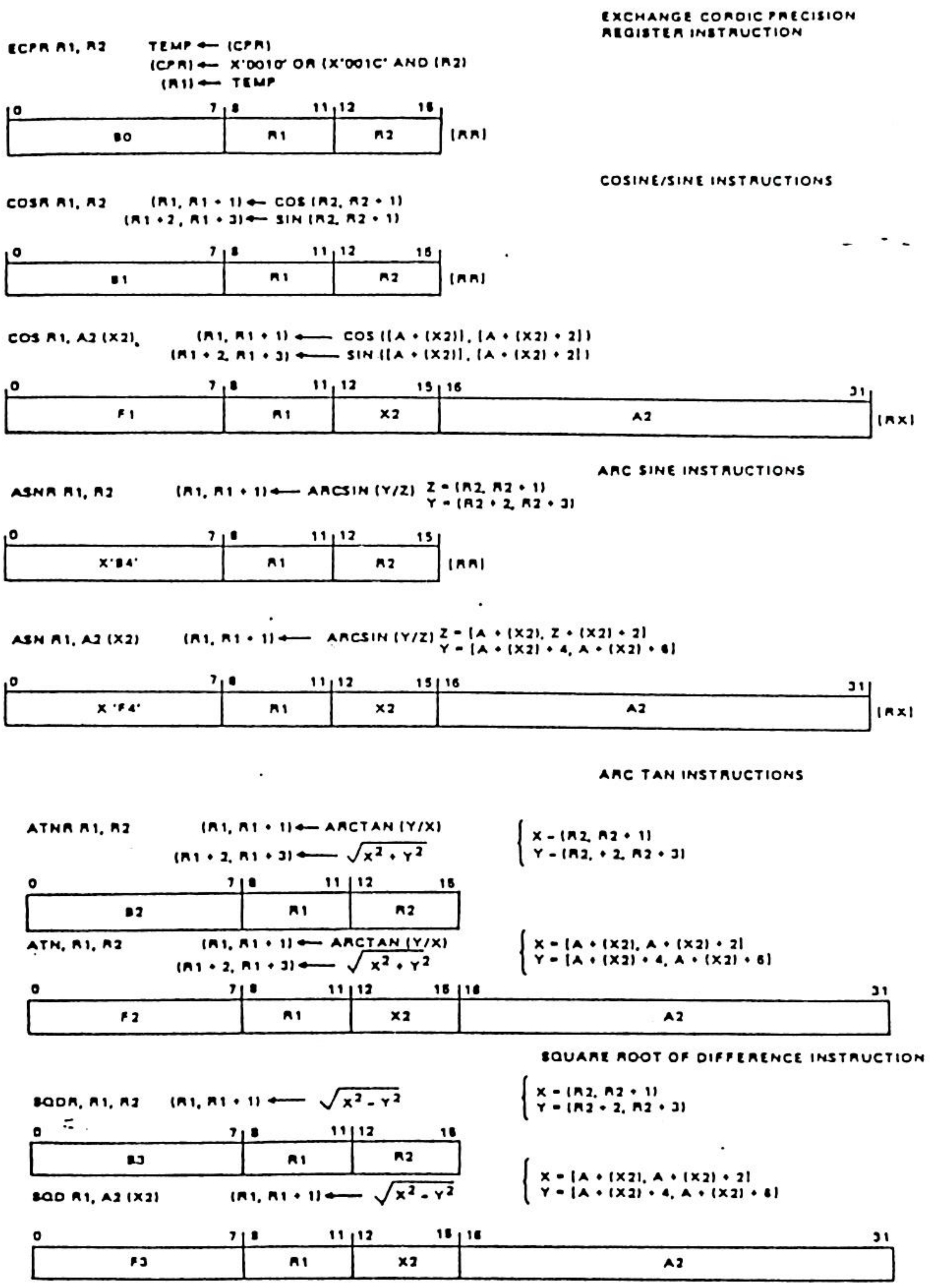

EXCHANGE CORDIC PRECISION REGISTER INSTRUCTION
ECPR R1, R2
TEMP ← (CPR)
(CPR) ← X'0010' OR (X'001C' AND (R2)
(R1) ← TEMP
0
7
8
11
12
15
80
R1
R2
(RR)
COSINE/SINE INSTRUCTIONS
COSR R1, R2
(R1, R1 + 1) ← COS (R2, R2 + 1)
(R1 + 2, R1 + 3) ← SIN (R2, R2 + 1)
81
R1
R2
(RR)
COS R1, A2 (X2)
(R1, R1 + 1) ← COS ([A + (X2)], [A + (X2) + 2])
(R1 + 2, R1 + 3) ← SIN ([A + (X2)], [A + (X2) + 2])
16
31
F1
R1
X2
A2
(RX)
ARC SINE INSTRUCTIONS
ASNR R1, R2
(R1, R1 + 1) ← ARCSIN (Y/Z)
Z = (R2, R2 + 1)
Y = (R2 + 2, R2 + 3)
X'84'
R1
R2
(RR)
ASN R1, A2 (X2)
(R1, R1 + 1) ← ARCSIN (Y/Z)
Z = [A + (X2), Z + (X2) + 2]
Y = [A + (X2) + 4, A + (X2) + 6]
X 'F4'
R1
X2
A2
(RX)
ARC TAN INSTRUCTIONS
ATNR R1, R2
(R1, R1 + 1) ← ARCTAN (Y/X)
(R1 + 2, R1 + 3) ← $\sqrt{X^2 + Y^2}$
X - (R2, R2 + 1)
Y - (R2, + 2, R2 + 3)
82
R1
R2
ATN, R1, R2
(R1, R1 + 1) ← ARCTAN (Y/X)
(R1 + 2, R1 + 3) ← $\sqrt{X^2 + Y^2}$
X = [A + (X2), A + (X2) + 2]
Y = [A + (X2) + 4, A + (X2) + 6]
F2
R1
X2
A2
SQUARE ROOT OF DIFFERENCE INSTRUCTION
SQDR, R1, R2
(R1, R1 + 1) ← $\sqrt{X^2 - Y^2}$
X = (R2, R2 + 1)
Y = (R2 + 2, R2 + 3)
83
R1
R2
SQD R1, A2 (X2)
(R1, R1 + 1) ← $\sqrt{X^2 - Y^2}$
X = [A + (X2), A + (X2) + 2]
Y = [A + (X2) + 4, A + (X2) + 6]
F3
R1
X2
A2

Figure VI-32. Trigonometric Operation Instructions

Table VI-3. Accuracy of CORDIC Routine

Instruction	Function	Error of Results (hexadecimal) with CPR Value equal to			
		16	20	24	28
COSR, COS	$a \leftarrow \cos(\theta)$	00008000	00000800	00000080	00000016
	$b \leftarrow \sin(\theta)$	00008000	00000800	00000080	00000016
ATNR, ATN	$\theta \leftarrow \arctan(y/x)$	00005000	00000500	00000050	00000013
	$z \leftarrow \sqrt{x^2 + y^2}$	00008000	00000800	00000080	00000016
SQDR ,SQD	$z \leftarrow \sqrt{x^2 - y^2}$	00008000	00000800	00000080	00000017
ASNR, ASN	$\theta \leftarrow \arcsin(y/z)$	0000D000	00000D00	000000D0	0000002A

θ is "angle" in BAMS: $-1.0 \leq 1.0$

a, b are "fixed scalars": $|a| \leq 1.0$

x, y, z, are "arbitrary scalars"

The meaning of the accuracy figures is as follows:

$|\text{Result}_{\text{computed}} - \text{Result}_{\text{exact}}| \leq \text{Error}$

(7) Arc Sine Instructions. The arc sine (ASN and ASNR) instructions calculate the arc sine of the ratio of two quantities. The second operand denotes a pair of 32-bit, twos complement, arbitrary scale quantities; say "z" and "y". The arc sin of y/z is calculated in BAMS and placed in register pair R1, R1 + 1. R1 (and R2) is even. In ASNR, R2 is less than or equal to 12; R1 may utilize any of the quadruple of registers denoted by R2. The arc sine is in the range $- /2 \leq$ arc sine $y/z \leq /2$. The sign of z is ignored (that is, if $z < 0$ then z is replaced by $-z$). nIf $z < y$ or if $z = y = 0$, then (R1, R1 + 1) is not changed, and the V flab is set. This is considered an overflow.

Resulting Condition Code: CVGL

0 normal case

1 $|z| < |y|$ or $(z = y = 0)$

(8) Arc Tan, $\sqrt{x^2 + y^2}$ Instructions. The arc tan (ATN and ATNR) instructions calculate the arc tan of the ratio of two quantities, and also lthe square root of the sum of the squares of the quantities. The second operand denotes a pair of 32-bit, twos complement, arbitrary scale quantities; say "x" and "y". The arc tangent of the ratio y/x is calculated in the BAMS, and placed in register pair R1, R1 + 1. The square root of the sum of the squares is calculated and placed in register pair R1 + 2, R1 + 3. The scale of x and y are the same, and the square root result is in the same scale. R1 (and R2) is even and is less than or equal to 12. In ATNR, R1 and R2 may be the same, or may denote overlapping quadruples of registers.

If both x and y are zero, then the answer for both the arc tangent and the square root of the sum of the squares will be zero. If the arguments x and y are such that the square root of the sum of the squares is breater than X'7FFFFFFF', then no registers are changed and the V flag is set. This constitutes an overflow. (Note that inherent errors in the calculation can cause a result that would be slightly less than X'7FFFFFFF' ti come out greater than X'FFFFFFF' and cause the overflow condition.)

Resulting Condition Code: CVGL

0 normal case

1 $\sqrt{x^2 + y^2}$ < X'7FFFFFFF'

Table VI-4. Instruction Augment Set

Instruction	Data (Bits)	Mnemonic			Op-Code (HBX)			Execution Time in microseconds		
		RR/SF	RI	RX	RR/SF	RI	RX	RR/SR	RI	RX
Set Status Bits	8		SESB			E4			1.6	
Reset Status Bits	8		RESB			E3			1.4	
Load Complement Halfword	16	LCHR		LCH	12		53	1.0		2.0
Load and Change Number Base Halfword	16	LCNHR		LCNH	10		51	1.2		2.4
Load Absolute Value Halfword	16	LAVR		LAV	11		52	1.2		2.4

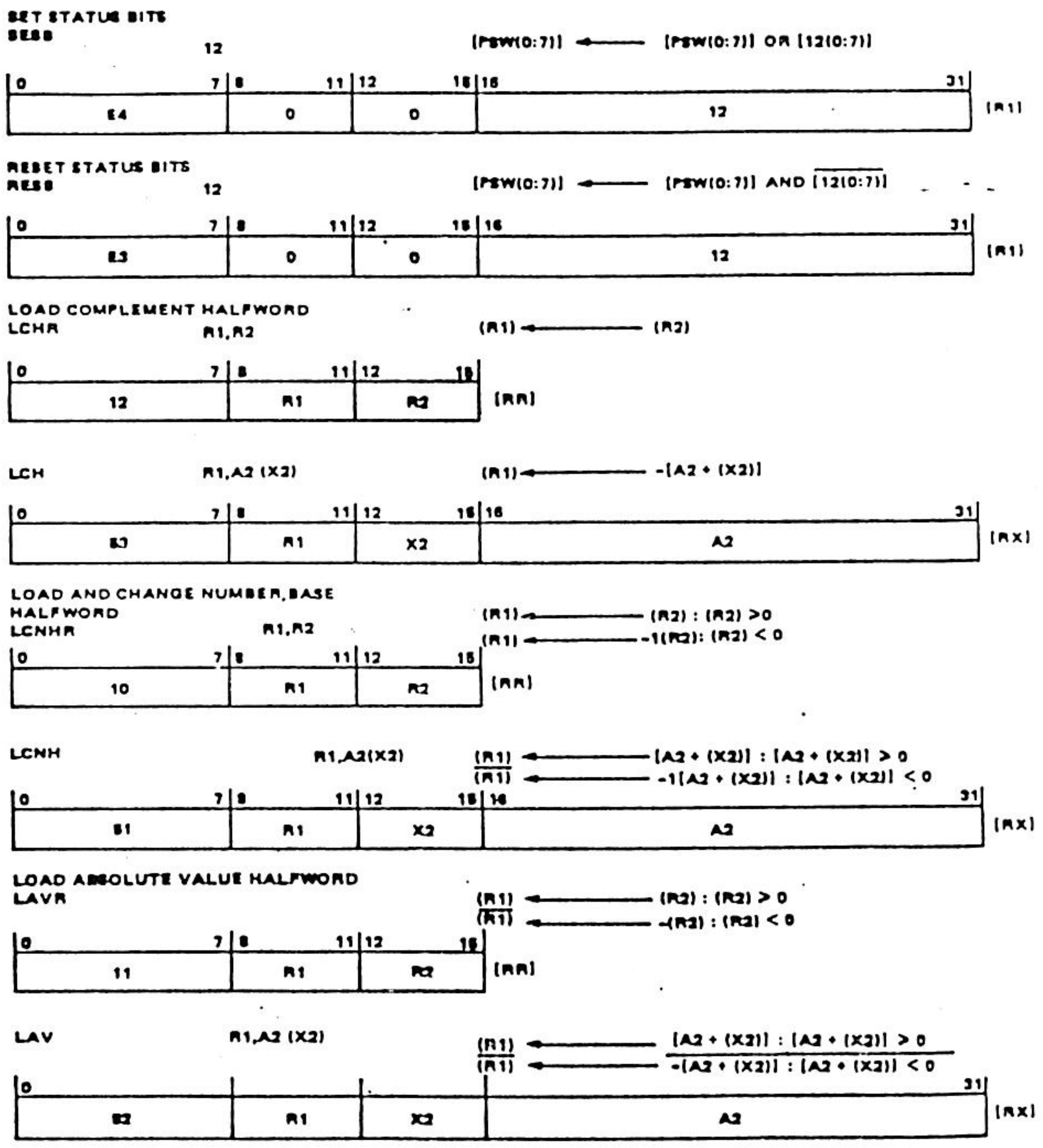


Figure VI-33. Instruction Augment Set

(9) $\sqrt{x^2 - y^2}$ Instructions. The square root of the difference (SQDR and SQD) instructions calculate the square root of the difference of the squares of the two quantities. The second operand denotes a pair of 32-bit, twos complement, arbitrary scale quantities; say "x" and "y". The square root of the difference of the squares is calculated and placed in register pair R1 + 1, R1 + 2. The scale of x and y are the same, and the square root result is in the same scale. R1 is even and is less than or equal to 14. In SQDR, R2 is less than or equal to 12. R1 may utilize any of the quadruples of registers denoted by R2. While there is some inaccuracy in the general operation, if x = y then an exact zero will be the result. If x < y then (R1, R1 + 1) is not changed, and the V flag is set. This constitutes an overflow.

Resulting Condition Code	CVGL
	0 normal case
	1 $|x| < |y|$

m. Instruction Augment Set. The instruction autment set consists of five instructions added to the basic computer instruction repertoir (Table VI-4). These instructions provide the computer with the capability to alter Program Status Bits; perform twos complement arithmetic operations; convert data from a sign and magnitude number representation to a twos complement number representation and vice versa; and perform an absolute magnitude operation on twos complement data. These instructions use the Register to Register (RR), the Short Format (SF), the Register to Indexed Storage (RX), and the Register Immediate (RI) formats. The exact format, op-code, assembler notation, and diagrammatioc representation of each instruction are shown in Figure VI-33. The operation and resulting Condition Code are as follows:

(1) Set Status Bits. The set status bits (SESB) instruction causes PSW bits (0:7) to be set according to the corresponding bits (0:7) set in the I2 field of the instruction. R1 and X2 are zero. I2 field bits (0:7), which are reset, and I2 bits (8:15) do not change the corresponding PSW bits. The Condition Code remains unchanged. This instruction is privileged.

(2) Reset Status Bits. The reset status bits (RESB) instruction causes PSW bits (0:7) to be reset according to the corresponding bits (0:7) set in the I2 field of the instruction. I2 field bits (0:7), which are reset, and I2 bits (8:15) do not change the corresponding PSW bits. The Condition Code remains unchanged. This instruction is privileged.

(3) Load Complement Halfword. The load complement halfword (LCHR and LCH) instructions cause the twos complement of the second operand to be loaded into the general register specified by R1. The second operand is unchanged. In the RX format, the second operand is located on a halfword boundary. If the operand is 8000_{16}, the result is 8000_{16}. The resulting Condition Code shall be:

C	V	G	L	
0	0	0	0	Result is zero
1	0	0	1	Result is less than zero (operand is not 8000_{16})
1	0	1	0	Result is greater than zero
1	1	0	1	Arithmetic overflow (operand is 8000_{16})

(4) Load and Change Number Base Halfword. The load and change number halfword (LCNHR and LCNH) instructions change the number representation of the second operand from twos complement to sign and magnitude, or from sign and magnitude to twos complement. This is accomplished as follows. If the second operand is positive (bit zero equals zero), the unmodified second operand is loaded into the general register specified by R1. Number base conversion is equivalent to subtracting 80000_{16} and performing a twos complement on the result. The number 8000_{16} is changed to 000_{16}. (Note that this is correct for sign and magnitude conversion.) The second operand is unchanged. In the RX format, the second operand is located on a halfword boundary. The resulting Condition Code shall be:

C	V	G	L	
0	0	0	0	Result is zero (operand is zero)
0	0	0	1	Result is less than zero (operand is less than zero, but not 8000_{16})
0	0	1	0	Result is greater than zero
1	0	0	0	Result is zero (operand is 8000_{16})

Table VI-5. Unimplemented 48-Bit Floating-Point Instructions

Mnemonic	Op-Code	Title
LF	78	48-bit floating-point load
LFR	38	48-bit floating-point load, register to register
STF	70	48-bit floating-point store
AF	7A	48-bit floating-point add
AFR	3A	48-bit floating-point add, register to register
SF	7B	48-bit floating-point subtract
SFR	3B	48-bit floating-point subtract, register to register
CF	79	48-bit floating-point compare
CFR	39	48-bit floating-point compare, register to register
MF	7C	48-bit floating-point multiply
MFR	3C	48-bit floating-point multiply, register to register
DF	7D	48-bit floating-point divide
DFR	3D	48-bit floating-point divide, register to register

(5) Load Absolute Value Halfword. The load absolute value halfword (LAVR and LAV) instructions cause the absolute magnitude (positive value) of the second operand to be loaded into the general register specified by R1. The second operand is unchanged. If the second operand is negative, a twos complement is performed prior to loading the second operand into the R1 general register. If the operand is 8000_{16}, the result is 8000_{16}. In the RX format, the second operand is located on a halfword boundary. The resulting Condition Code shall be:

C	V	G	L	
0	0	0	0	Result is zero
0	0	1	0	Result is greater than zero (operand is greater than zero)
1	0	1	0	Result is greater than zero (oprerand is less zero, but not 8000_{16})
1	1	0	1	Arithmetic overflow (operand is 8000_{16})

Table VI-5 contains the 48-bit floating-point instructions that are not implemented in JSS HMP-1116 controller computers. These instructions are treated as no-ops if executed.

VI-2. CENTRAL COMPUTER INSTRUCTIONS. The Central Computer (CC) will be capable of executing instructions as specified in the following instruction repertoire, which shall consist of 64 instructions. Use of all 64 instructions shall be possible, however, the use of the eleven serial I/O instructions out of the set may not be applicable for the application described herein.

a. Control Operations:

OP CODE	R	Y
0 4	5 8	9 17

(1) TRU - Transfer Unconditionally (OP CODE 00). The computer will take the next instruction from the location in memory specified by Y and R.

(2) TRN - Transfer on Accumulator Negative (OP CODE 01). The sign bit of the accumulator is sensed. If it is negative (1), control is transferred to the memory location specified by Y and R. If the accumulator sign is positive (0), the computer will take the next instruction in sequence.

(3) TRZ - Transfer on Accumulator Zero (OP CODE 02). The contents of the accumulator are tested for a zero value. The sign bit is not tested. If the contents of the accumulator are zero, control is transferred to the memory location specified by Y and R. If the contents of the accumulator are not zero, the computer will take the next sequential instruction.