Alphabetical List of HMP-1116 Operations

Code (RR/SF)	Mnemonic	Description
0E	ACHR 14	Add With Carry Halfword
3A	AER 39	Floating-Point Add
0A	AHR 10	Add Halfword
9F	AIR 45	Acknowledge Interrupt
26	AIS 10	Add Immediate Short
01	BALR 35	Branch and Link
22	BFBS 32	Branch on False Backward Short
03	BFCR 32	Branch on False Condition
23	BFFS 32	Branch on False Forward Short
20	BTBS 32	Branch on True Backward Short
02	BTCR 32	Branch on True Condition
21	BTFS 32	Branch on True Forward Short
29	CER 40	Floating-Point Compare
09	CHR 21	Compare Halfword
05	CLHR 21	Compare Logical Halfword
2D	DER 41	Floating-Point Divide
0D	DHR 16	Divide Halfword
ZE	EPOR 42	Exchange Operand Bank Address
ZF	EPPR 42	Exchange Program Address
95	EPSR 44	Exchange Program Status
94	EXBR 22	Exchange Byte
11	LAVR	Load Absolute Value Halfword
93	LBR 22	Load Byte
12	LCHR	Load Complement Halfword
10	LCNHR	Load and Change Number Base Halfword
25	LCS 3	Load Complement Short
28	LER 36	Floating-Point Load
08	LHR 7	Load Halfword
24	LIS 3	Load Immediate Short
2C	MER 40	Floating-Point Multiply
0C	MHR 16	Multiply Halfword

Alphabetical List of HMP-1116 Operations (Continued)

Code	Mnemonic	Description
(RR/SF)		
9C	MHUR 16	Multiply Halfword Unsigned
04	NHR 17	AND Halfword
9E	OCR 50	Output Command
06	OHR 17	OR Halfword
97	RBR 51	Read Block
9B	RDR 50	Read Data (Byte)
99	RHR 51	Read Halfword
0F	SCHR 16	Subtract With Carry Halfword
2B	SER 39	Floating-Point Subtract
0B	SHR 15	Subtract Halfword
27	SIS 15	Subtract Immediate Short
91	SLLS 24	Shift Left Logical Short
90	SRLS 29	Shift Right Logical Short
9D	SSR 45	Sense Status
92	STBR 22	Store Byte
96	WBR 52	Write Block
9A	WDR 50	Write Data (Byte)
98	WHR 51	Write Halfword
07	XHR 20	Exclusive OR Halfword
(RI)		
CA	AHI 10	Add Halfword
C0	BXH	Branch on Index High
C1	BXLE 35	Branch on Index Low or Equal
C9	CHI 21	Compare Halfword
C5	CLHI 21	Compare Logical Halfword
C8	LHI 7	Load Halfword
C2	LPSW 42	Load Program Status Word
C4	NHI 17	AND Halfword
C6	OHI 17	OR Halfword
E3	RESB	Reset Status Bits
EB	RLL 29	Rotate Left Fullword Logical

Alphabetical List of HMP-1116 Operations (Continued)

Code	Mnemonic	Description
(RI)		
EA	RRL 30	Rotate Right Fullword Logical
E4	SESB	Set Status Bits
CB	SHI 15	Subtract Halfword
E2	SINT 44	Simulate Interrupt
EF	SLA 30	Shift Left Fullword Arithmetic
CF	SLHA 30	Shift Left Halfword Arithmetic
CD	SLHL 24	Shift Left Halfword Logical
ED	SLL 24	Shift Left Fullword Logical
EE	SRA 31	Shift Right Fullword Arithmetic
CE	SRHA 31	Shift Right Halfword Arithmetic
CC	SRHL 29	Shift Right Halfword Logical
EC	SRL 29	Shift Right Fullword Logical
E1	SVC 44	Supervisor Call
C3	THI 20	Test Halfword Immediate
C7	XHRI 20	Exclusive OR Halfword
(RX)		
65	ABL 53	Add to Bottom of List
4E	ACH 14	Add with Carry Halfword
6A	AE 39	Floating-Point Add
4A	AH 10	Add Halfword
61	AHM 10	Add Halfword to Memory
DF	AI 45	Acknowledge Interrupt
D5	AL 50	Autoload
64	ATL 53	Add to Top of List
41	BAL 35	Branch and Link
43	BFC 32	Branch on False Condition
42	BTC 32	Branch on True Condition
69	CE 40	Floating-Point Compare
49	CH 21	Compare Halfword
D4	CLB 22	Compare Logical Byte
45	CLH 21	Compare Logical Halfword

Alphabetical List of HMP-1116 Operations (Continued)

Code	Mnemonic	Description
(RX)		
6D	DE 41	Floating-Point Divide
4D	DH 16	Divide Halfword
52	LAV	Load Absolute Value Halfword
D3	LB 22	Load Byte
53	LCH	Load Complement Halfword
51	LCNH	Load and Change Number Base Halfword
68	LE 36	Floating-Point Load
48	LH 7	Load Halfword
74	LH0 7	Load from Bank 0
75	LH1 7	Load from Bank 1
76	LH2 8	Load from Bank 2
77	LH3 8	Load from Bank 3
D1	LM 8	Load Multiple
6C	ME 40	Floating-Point Multiply
4C	MH 16	Multiply Halfword
DC	MHU 16	Multiply Halfword
44	NH 17	AND Halfword
DE	OC 50	Output Command
46	OH 17	OR Halfword
07	RB 51	Read Block
67	RBL 55	Remove from Bottom of List
DB	RD 50	Read Data (Byte)
D9	RH 51	Read Halfword
66	RTL 55	Remove from Top of List
4F	SCH 15	Subtract with Carry Halfword
6B	SE 39	Floating-Point Subtract
4B	SH 15	Subtract Halfword
DD	SS 45	Sense Status
D2	STB 22	Store Byte
60	STE 36	Floating-Point Store
40	STH 9	Store Halfword
F8	STH0 9	Store in Bank 0

Alphabetical List of HMP-1116 Operations (Continued)

Code	Mnemonic	Description
(RX)		
F9	STH1 9	Store in Bank 1
FA	STH2 9	Store in Bank 2
FB	STH3 9	Store in Bank 3
D0	STM 8	Store Multiple
D6	WB 52	Write Block
DA	WD 50	Write Data (Byte)
D8	WH 51	Write Halfword
47	XH 20	Exclusive OR Halfword
(Double Precision)		
5A	ADP	Add Fullword (RX)
1A	ADPR	Add Fullword (RR/SF)
59	CDP	Compare Fullword (RX)
19	CDPR	Compare Fullword (RR/SF)
55	CLDP	Compare Logical Fullword (RX)
15	CLDPR	Compare Logical Fullword (RR/SF)
5D	DDP	Divide Fullword (RX)
1D	DDPR	Divide Fullword (RR/SF)
58	LDP	Load Fullword (RX)
18	LDPR	Load Fullword (RR/SF)
5C	MDP	Multiply Fullword (RX)
1C	MDPR	Multiply Fullword (RR/SF)
5B	SDP	Subtract Fullword (RX)
1B	SDPR	Subtract Fullword (RR/SF)
E9	SLQA	Shift Left Doubleword Arithmetic (RI)
E7	SLQR	Shift Left Logical Doubleword (RI)
E8	SRQA	Shift Right Doubleword Arithmetic (RI)
E6	SRQL	Shift Right Logical Doubleword (RI)
50	STDP	Store Fullword (RX)

Appendix 3

Numerical Listing of Computer Operation Codes

Numerical List of HMP-1116 Operation Codes

Code (RR/SF)	Mnemonic	Description
01	BALR	Branch and Link
02	BTCR	Branch on True Condition
03	BFCR	Branch on False Condition
04	NHR	AND Halfword
05	CLHR	Compare Logical Halfword
06	OHR	OR Halfword
07	XHR	Exclusive OR Halfword
08	LHR	Load Halfword
09	CHR	Compare Halfword
0A	AHR	Add Halfword
0B	SHR	Subtract Halfword
0C	MHR	Multiply Halfword
0D	DHR	Divide Halfword
0E	ACHR	Add With Carry Halfword
0F	SCHR	Subtract with Carry Halfword
10	LCNHR	Load and Change Number Base Halfword
11	LAVR	Load Absolute Value Halfword
12	LCHR	Load Complement Halfword
20	BTBS	Branch on True Backward Short
21	BTFS	Branch on True Forward Short
22	BFBS	Branch on False Backward Short
23	BFFS	Branch on False Forward Short
24	LIS	Load Immediate Short
25	LCS	Load Complement Short
26	AIS	Add Immediate Short
27	SIS	Subtract Immediate Short
28	LER	Floating-Point Load
29	CER	Floating-Point Compare
2B	SER	Floating-Point Subtract
2C	MER	Floating-Point Multiply
2D	DER	Floating-Point Divide
2E	EPOR	Exchange Operand Band Address

Numerical List of HMP-1116 Operation Codes (Continued)

Code	Mnemonic	Description
(RR/SF)		
2F	EPPR	Exchange Program Address
3A	AER	Floating-Point Add
90	SRLS	Shift Right Logical Short
91	SLLS	Shift Left Logical Short
92	STBR	Store Byte
93	LBR	Load Byte
94	EXBR	Exchange Byte
95	EPSR	Exchange Program Status
96	WBR	Write Block
97	RBR	Read Block
98	WHR	Write Halfword
99	RHR	Read Halfword
9A	WDR	Write Data (Byte)
9B	RDR	Read Data (Byte)
9C	MHUR	Multiply Halfword Unsigned
9D	SSR	Sense Status
9E	OCR	Output Command
9F	AIR	Acknowledge Interrupt
(RI)		
C0	BXH	Branch on Index High
C1	BXLE	Branch on Index Low or Equal
C2	LPSW	Load Program Status Word
C3	THI	Test Halfword Immediate
C4	NHI	AND Halfword
C5	CLHI	Compare Logical Halfword
C6	OHR	OH Halfword
C7	XHR	Exclusive OR Halfword
C8	LHI	Load Halfword
C9	CHI	Compare Halfword
CA	AHI	Add Halfword
CB	SHI	Subtract Halfword

Numerical List of HMP-1116 Operation Codes (Continued)

Code	Mnemonic	Description
(RI)		
CC	SRHI	Shift Right Halfword Logical
CD	SLHI	Shift Left Halfword Logical
CE	SRHA	Shift Right Halfword Arithmetic
CF	SLHA	Shift Left Halfword Arithmetic
E1	SVC	Supervisor Call
E2	SINT	Simulate Interrupt
E3	RESB	Reset Status Bits
E4	SESB	Set Status Bits
EA	RRL	Rotate Right Fullword Logical
EB	RLL	Rotate Left Fullword Logical
EC	SRL	Shift Right Fullword Logical
ED	SLL	Shift Left Fullword Logical
EE	SRA	Shift Right Fullword Arithmetic
EF	SLA	Shift Left Fullword Arithmetic
(RX)		
40	STH	Store Halfword
41	BAL	Branch and Link
42	BTC	Branch on True Condition
43	BFC	Branch on False Condition
44	NH	AND Halfword
45	CLH	Compare Logical Halfword
46	OH	OR Halfword
47	XH	Exclusive OR Halfword
48	LH	Load Halfword
49	CH	Compare Halfword
4A	AH	Add Halfword
4B	SH	Subtract Halfword
4C	MH	Multiply Halfword
4D	DH	Divide Halfword
4E	ACH	Add With Carry Halfword
4F	SCH	Subtract With Carry Halfword

Numerical List of HMP-1116 Operation Codes (Continued)

Code	Mnemonic	Description
(RX)		
51	LCNH	Load and Change Number Base Halfword
52	LAV	Load Absolute Value Halfword
53	LCH	Load Complement Halfword
60	STE	Floating-Point Store
61	AHM	Add Halfword to Memory
64	ATL	Add to Top of List
65	ABL	Add to Bottom of List
66	RTL	Remove From Top of List
67	RBL	Remove From Bottom of List
68	LE	Floating-Point Load
69	CE	Floating-Point Compare
6A	AE	Floating-Point Add
6B	SE	Floating-Point Subtract
6C	ME	Floating-Point Multiply
6D	DE	Floating-Point Divide
74	LH0	Load From Bank 0
75	LH1	Load From Bank 1
76	LH2	Load From Bank 2
77	LH3	Load From Bank 3
D0	STM	Store Multiple
D1	LM	Load Multiple
D2	STB	Store Byte
D3	LB	Load Byte
D4	CLB	Compare Logical Byte
D5	AL	Autoload
D6	WB	Write Block
D7	RB	Read Block
D8	WH	Write Halfword
D9	RH	Read Halfword
DA	WD	Write Data (Byte)
DB	RD	Read Data (Byte)
DC	MHU	Multiply Halfword Unsigned

Numerical List of HMP-1116 Operation Codes (Continued)

Code	Mnemonic	Description
(RX)		
DD	SS	Sense Status
DE	OC	Output Command
DF	AI	Acknowledge Interrupt
F8	STH0	Store in Bank 0
F9	STH1	Store in Bank 1
FA	STH2	Store in Bank 2
FB	STH3	Store in Bank 3
(Double Precision)		
15	CLDPR	Compare Logical Fullword (RR/SF)
18	LDPR	Load Fullword (RR/SF)
19	CDPR	Compare Fullword (RR/SF)
1A	ADPR	Add Fullword (RR/SF)
1B	SDPR	Subtract Fullword (RR/SF)
1C	MDPR	Multiply Fullword (RR/SF)
1D	DDPR	Divide Fullword (RR/SF)
50	STDP	Store Fullword (RX)
55	CLDP	Compare Logical Fullword (RX)
58	LDP	Load Fullword (RX)
59	CDP	Compare Fullword (RX)
5A	ADP	Add Fullword (RX)
5B	SDP	Subtract Fullword (RX)
5C	MDP	Multiply Fullword (RX)
5D	DDP	Divide Fullword (RX)
E6	SRQL	Shift Right Logical Doubleword (RI)
E7	SLQL	Shift Left Logical Doubleword (RI)
E8	SRQA	Shift Right Doubleword Arithmetic (RI)
E9	SLQA	Shift Left Doubleword Arithmetic (RI)

Appendix 10

Instructions

10.0 Controller Computer Instructions Description

The execution of each instruction shall effect the Condition Code in the Program Status Register as specified in the CVGL (reference paragraph 3.2.1.1.2) 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 3.2.1.1.6. For the purpose of this specification the instructions are grouped in the following categories:

Category of Instruction	No. of Types	Paragraph/Table
Fixed-Point Load/Store	16	10.2
Fixed-Point Arithmetic	19	10.3
Logical and Compare	16	10.4
Byte Handling	6	10.5
Shift/Rotate	12	10.6
Branch	12	10.7
Floating-Point	13	10.8
System Control	6	10.9
Input/Output	19	10.10
List Processing	4	10.11
TOTAL	123	

The symbols and abbreviations used in the following subordinate paragraphs 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 Branch 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.

10.1 Effective Address Generation

The effective address for accessing memory will be generated by combining two MSBs and 16 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 16 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.

10.2 Fixed-Point Load/Store Instructions

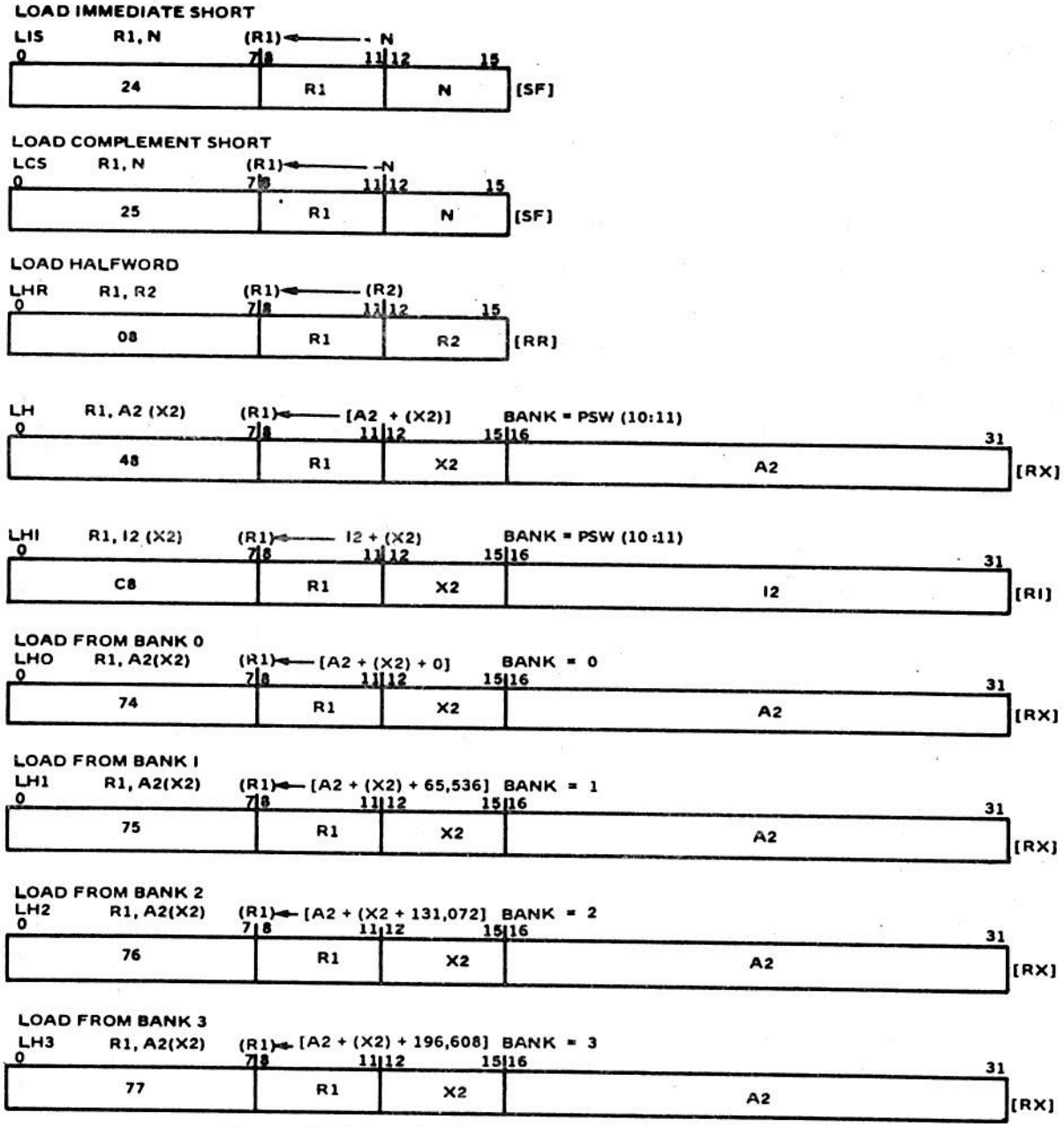
The Fixed-Point Load/Store instructions may be used to preset a register with an index value, load a register with the first operand for a subsequent arithmetic operation (e.g., add, multiply), or set the Condition Code for supplemental testing by a Branch on Condition instruction. These instructions 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 10.2-1, -2 and -3. The operation and resulting Condition Code shall be as follows:

a. 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.

b. 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.



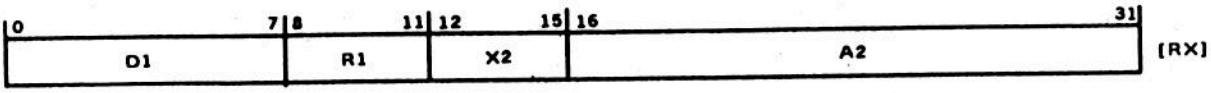
72582-10R

Figure 10.2-1. Fixed Point Load Instructions

73042-25

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



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

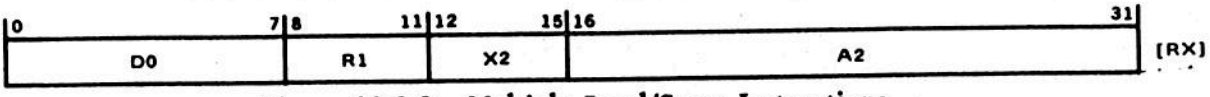


Figure 10.2-2. Multiple Load/Store Instructions

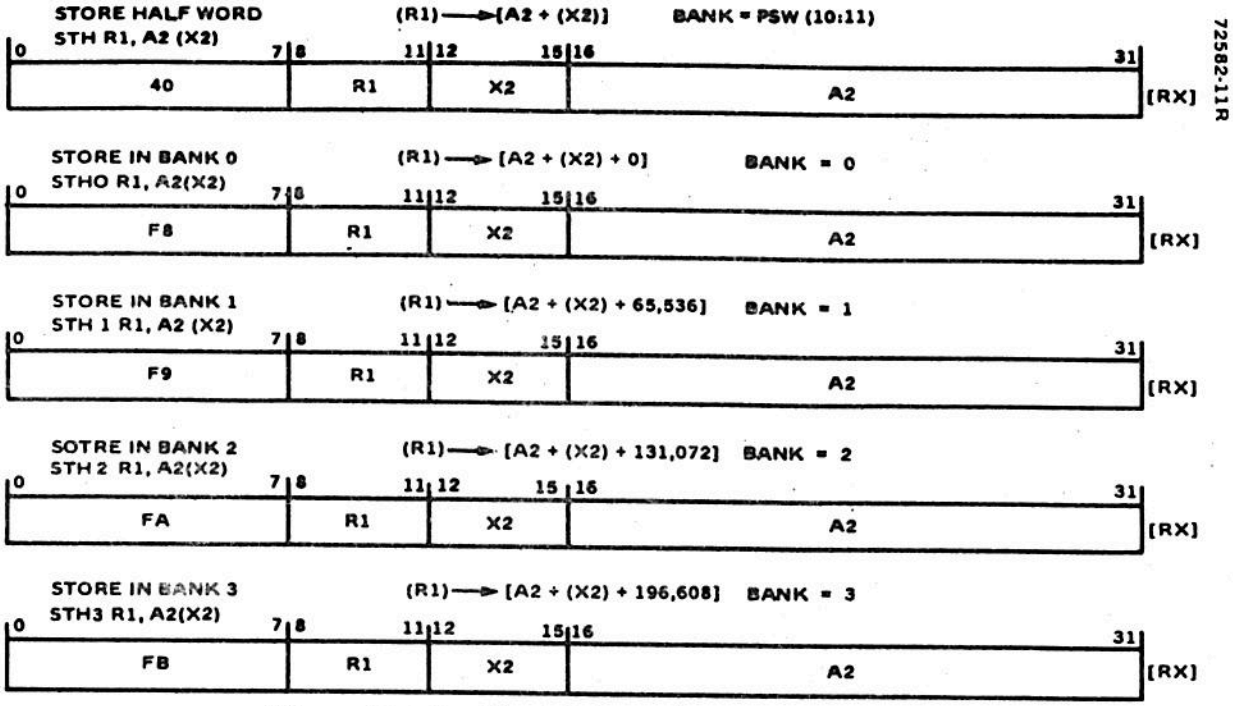


Figure 10.2-3. Fixed-Point Store Instructions

c. 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.

d. Load From Bank 0: The Load From Bank 0 (LH0) 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.

e. 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.

f. 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.

g. 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.

h. 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.

i. 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.

j. 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.

k. 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.

l. 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.

m. 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.

n. 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.

10.3 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 registers 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 10.3-1, -2, -3, and -4. The operation and resulting Condition Code shall be as follows:

a. 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.

b. 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	Result is zero.
X	X	0	1	Result is less than zero.
X	X	1	0	Result is greater than zero.
X	1	X	X	Arithmetic overflow.
1	X	X	X	Carry.

c. 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

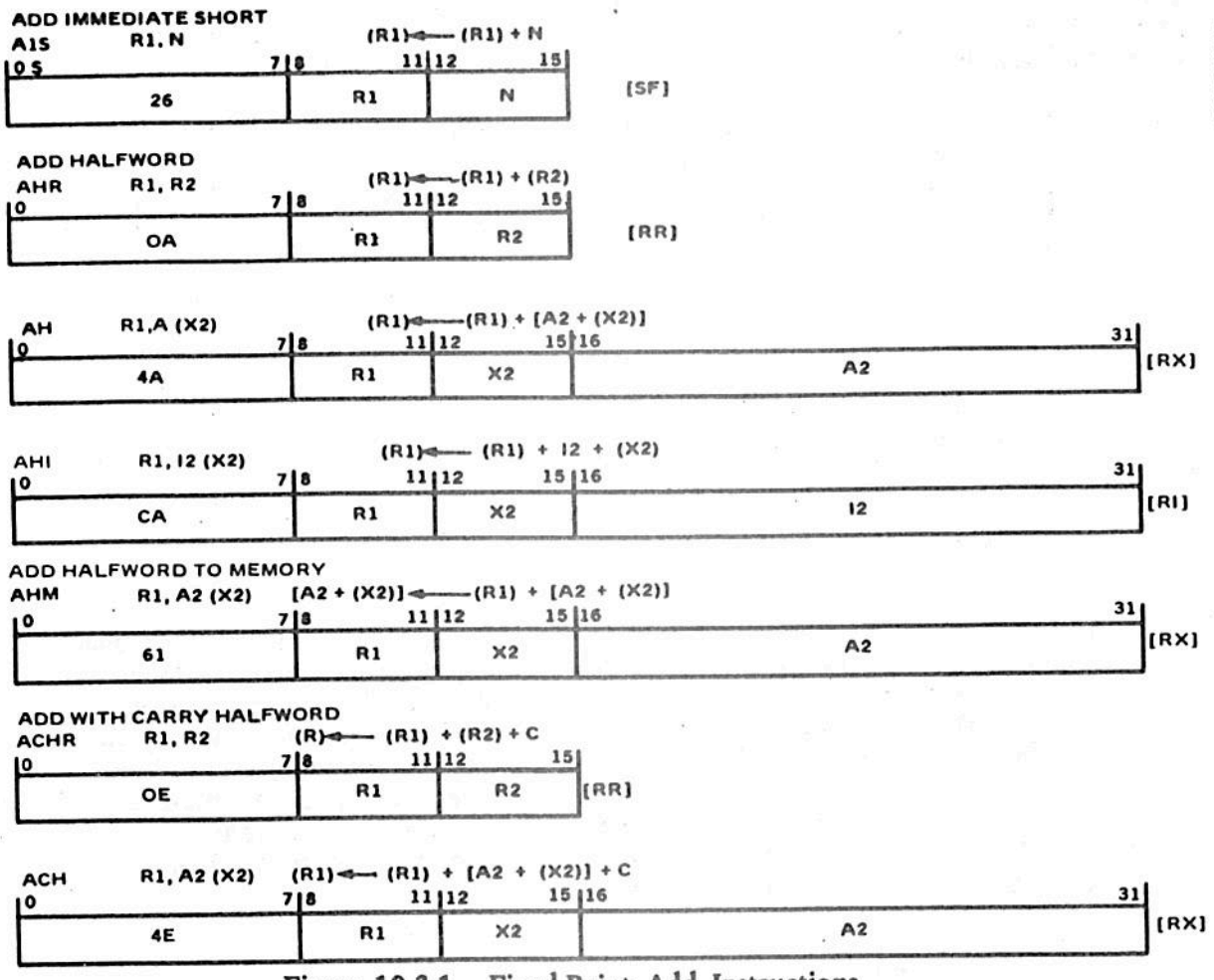


Figure 10.3-1. Fixed-Point Add Instructions

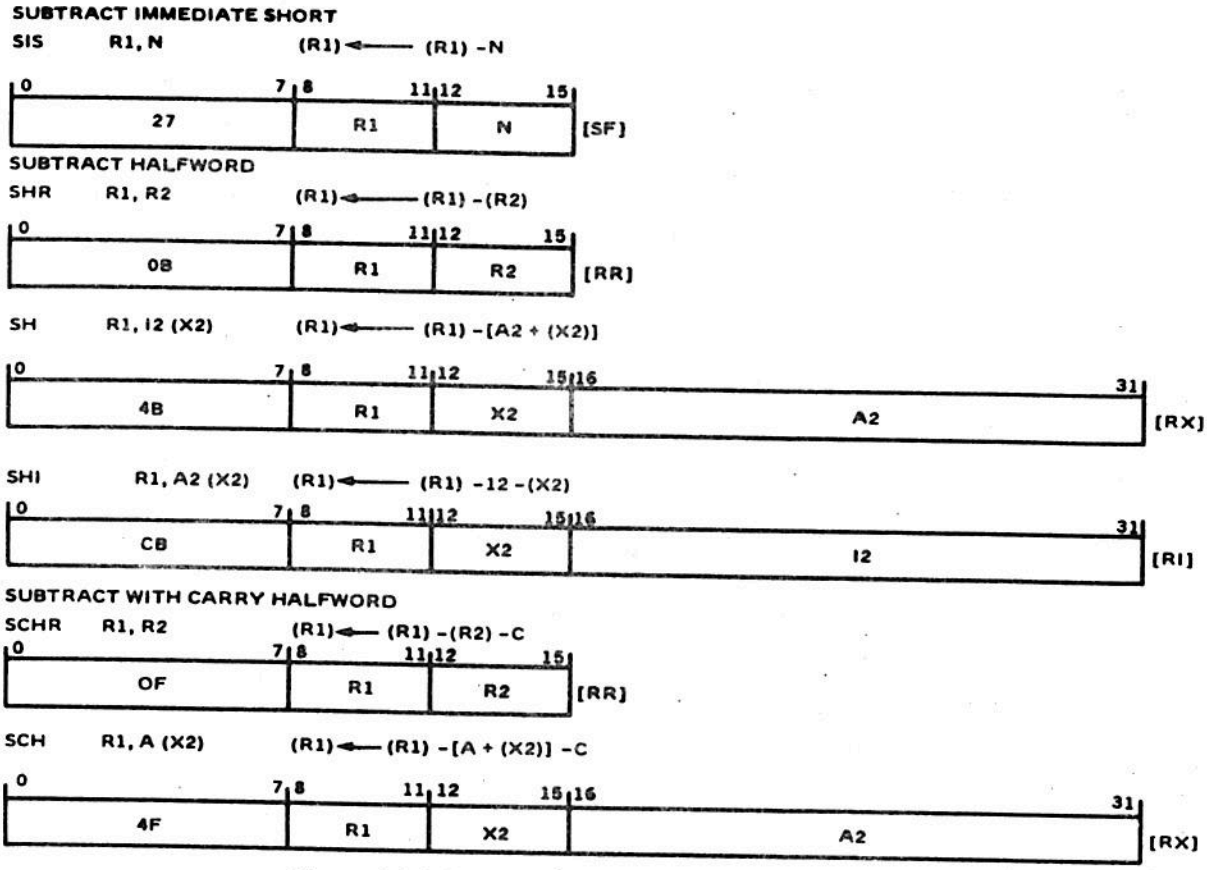


Figure 10.3-2. Fixed-Point Subtract Instructions

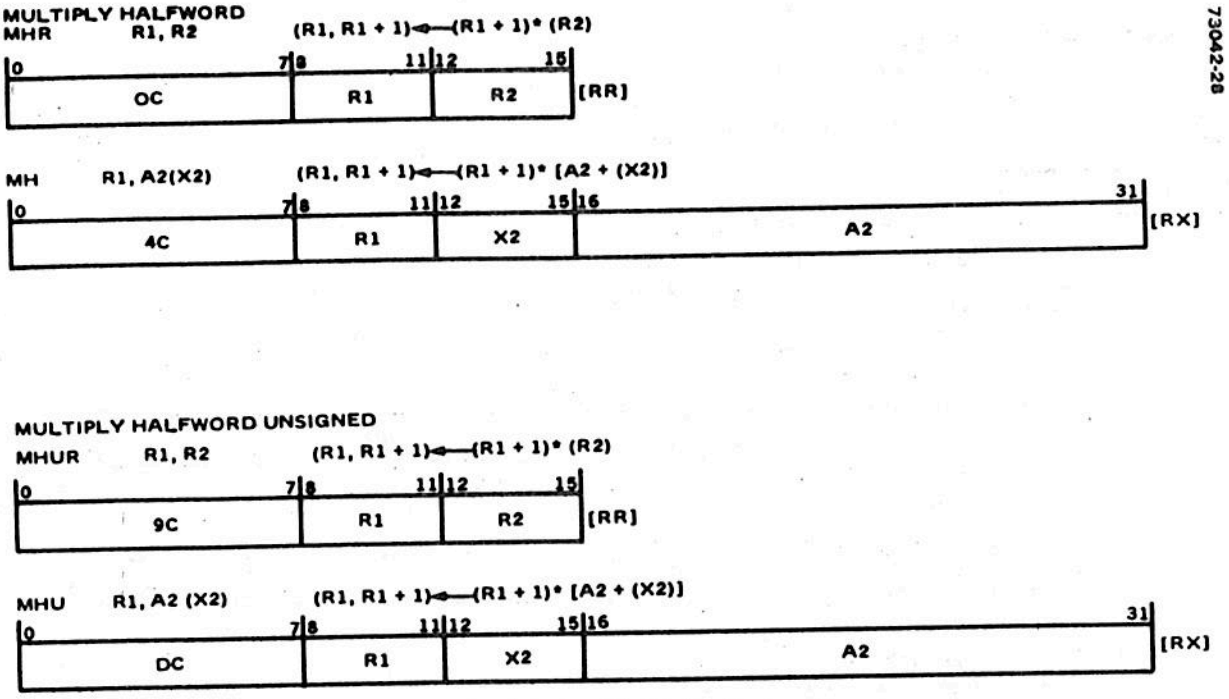


Figure 10.3-3. Fixed-Point Multiply Instructions

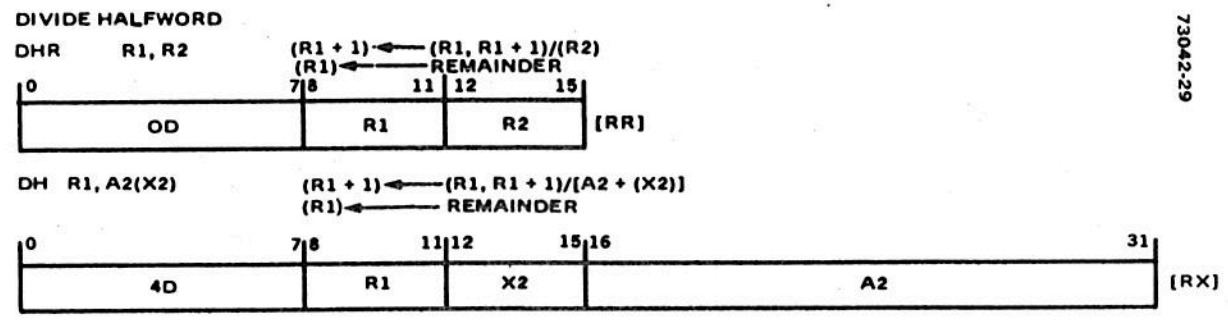


Figure 10.3-4. Fixed-Point Divide Instructions

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	Result is zero.
X	X	0	1	Result is less than zero.
X	X	1	0	Result is greater than zero.
X	1	X	X	Arithmetical overflow.
1	X	X	X	Carry.

d. 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	Result is zero.
X	X	0	1	Result is less than zero.
X	X	1	0	Result is greater than zero.
X	1	X	X	Arithmetic overflow.
1	X	X	X	Carry.

e. 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, 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	Result is zero.
X	X	0	1	Result is less than zero.
X	X	1	0	Result is greater than zero.
X	1	X	X	Arithmetic overflow.
1	X	X	X	Borrow.

f. 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	Result is zero.
X	X	0	1	Result is less than zero.
X	X	1	0	Result is greater than zero.
X	1	X	X	Arithmetic overflow.
1	X	X	X	Borrow.

g. 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	Result is zero.
X	X	0	1	Result is less than zero.
X	X	1	0	Result is greater than zero
X	1	X	X	Arithmetic overflow.
1	X	X	X	Borrow.

h. 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 R1+1. 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 15 bits with sign shall be contained in R1. The least significant 16 bits shall be contained in R1+1. The Condition Code shall remain unchanged.

i. 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 16-bits of both operands shall be considered to be magnitude. The resulting 32-bit product shall be contained in R1 and R1+1, 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.

j. 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 15-bit quotient with sign shall be contained by R1+1; a 15-bit remainder with sign shall be contained in R1, the second operand shall be unchanged. The sign of the result shall be determined 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 remain unchanged in all cases.

10.4 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 10.4-1 and -2. The operation and resulting Condition Code shall be as follows:

a. 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	

b. 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 the General Register index (X2) with the first operand General Register (R1). The truth table for the OR function shall be:

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

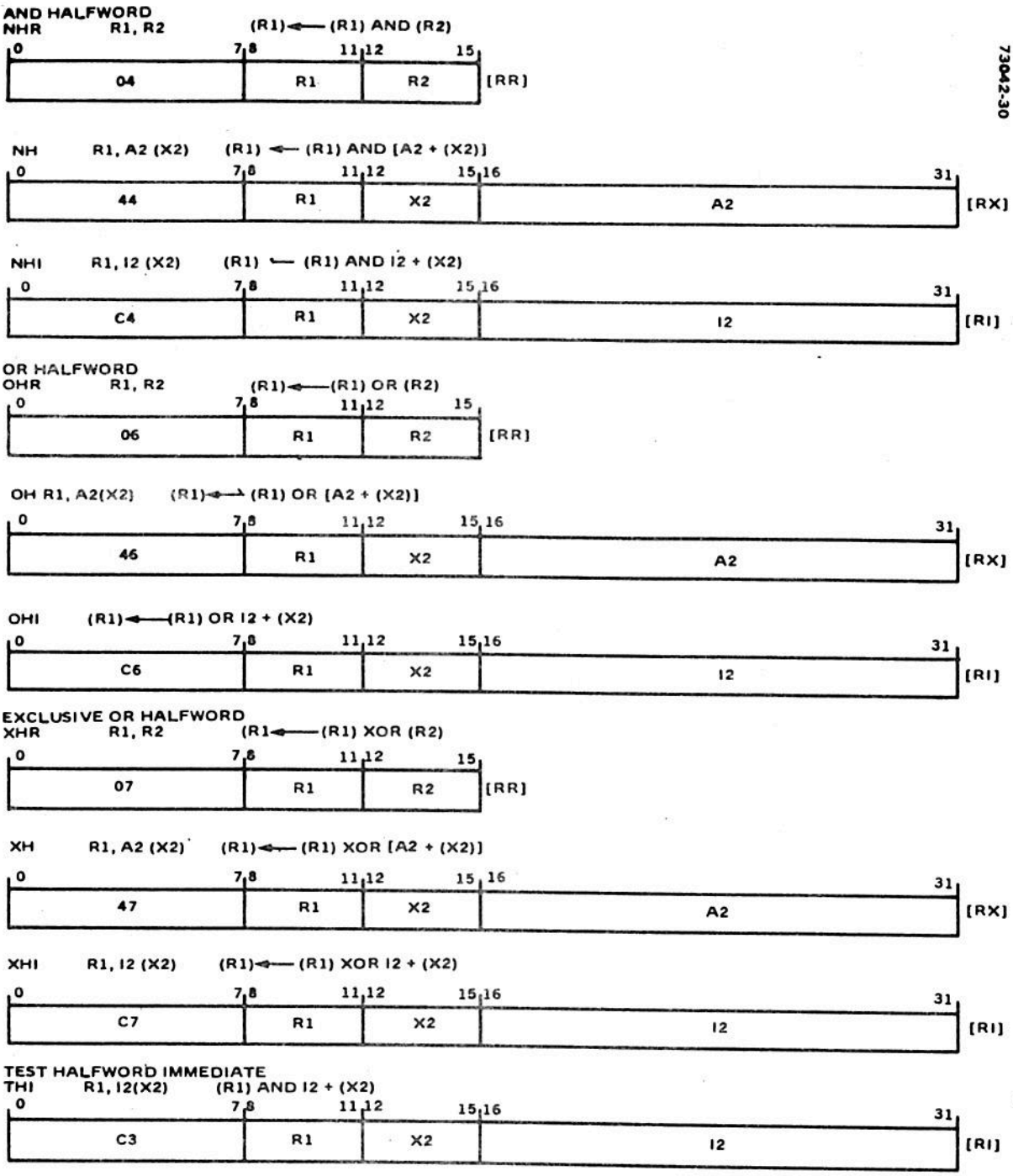


Figure 10.4-1. Logical and Bit Manipulating Instructions

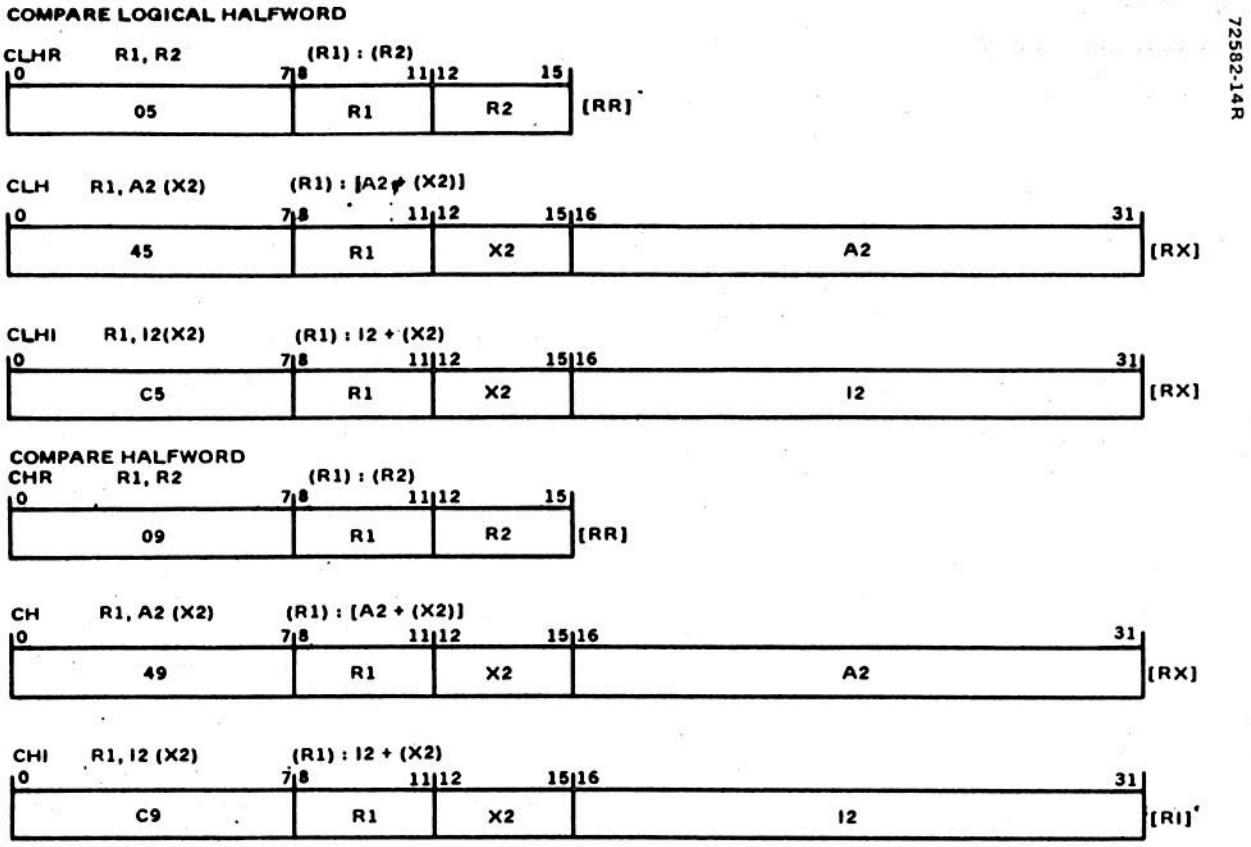


Figure 10.4-2. Compare Instructions

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	

c. 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 the General Register index (X2) with the first operand General Register (R1). The truth table for the Exclusive OR function shall be:

0 XOR 0 = 0

0 XOR 1 = 1

1 XOR 0 = 1

1 XOR 1 = 0

The resulting Condition Code shall be:

C	V	G	L	
X	X	0	0	Logical difference is zero.
X	X	0	1	Logical difference is not zero.
X	X	1	0	

d. 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 (THI) 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.

e. 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, 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 content 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.

f. 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.

10.5 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 10.5-1. The operation and resulting Condition Code shall be as follows:

a. Load Byte: The Load Byte (LB, LBR) instructions shall cause the eight-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.

b. Store Byte: The Store Byte (STB, STBR) instructions shall cause the right-most (least significant) eight-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 Register-to-Register (RR) form of this instruction, the left-most byte of R2, (0:7), shall be unchanged; the eight-bit quantity shall be stored in Bits 8:15 of the register specified by R2. The Condition Code shall remain unchanged.

c. Exchange Byte: The Exchange Byte Register (EXBR) instruction shall cause the two eight-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.

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

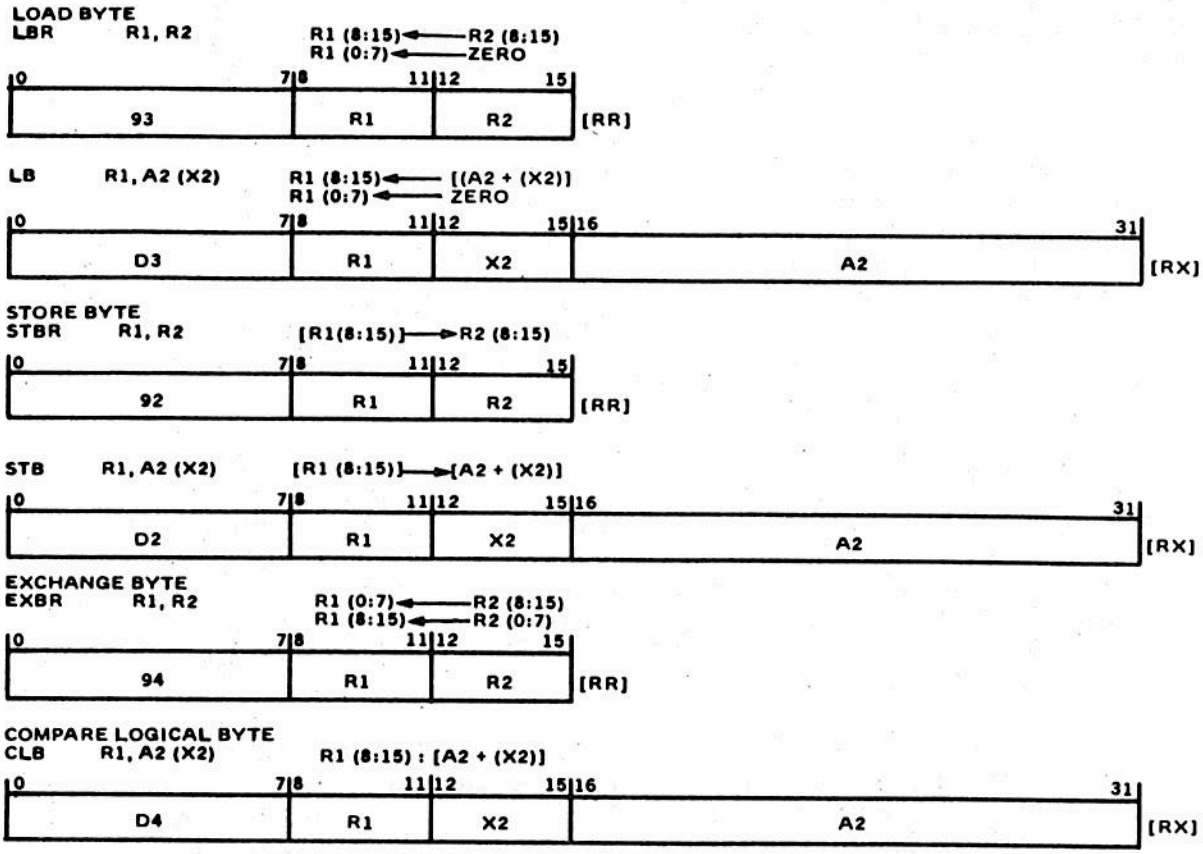


Figure 10.5-1. Byte Handling Instructions

10.6 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 16 bits of the fullword operand. The next sequential general register shall contain the least significant 16 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 10.6-1, -2, -3, -4, -5, and -6. The operation and resulting Condition Code shall be as follows:

a. 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 through 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 through 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 through 15) of the value I2+(X2). The Carry shall be formed by the output of R1. The resulting Condition Code shall be:

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.

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

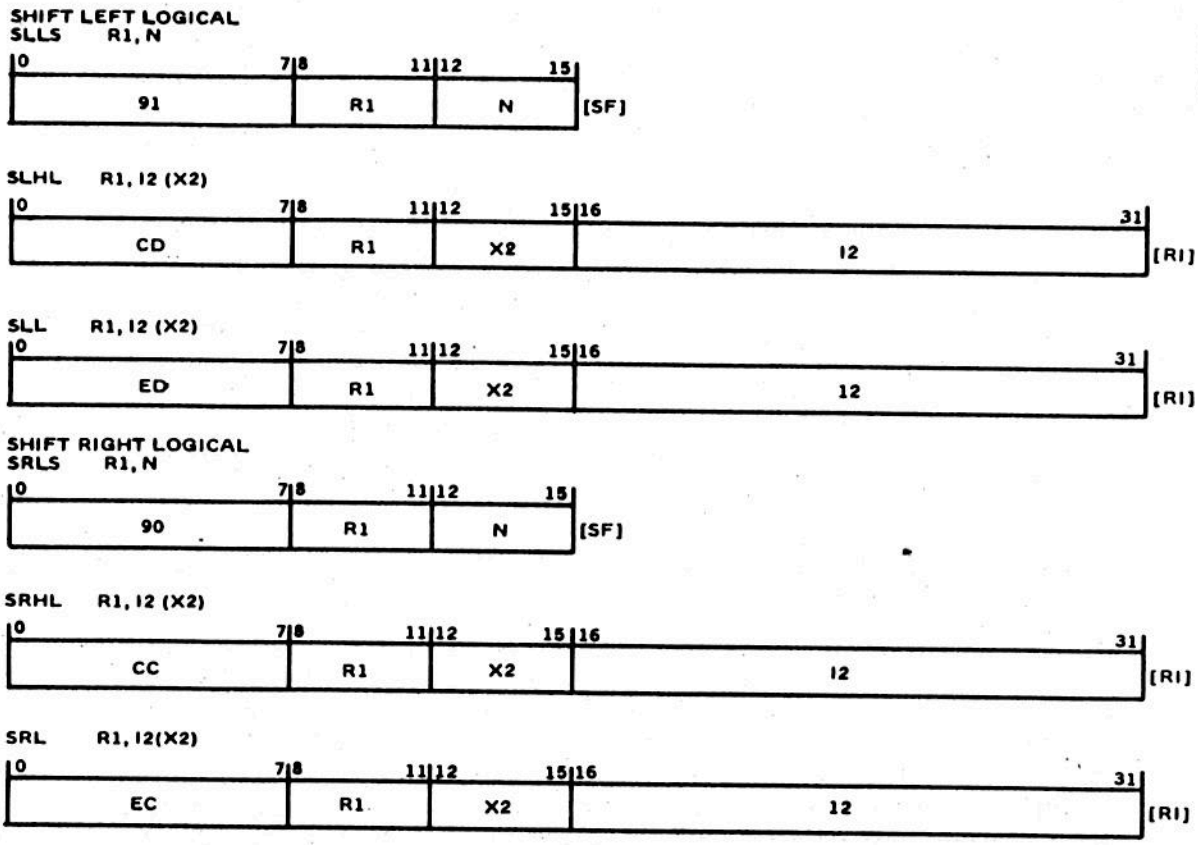


Figure 10.6-1. Logical Shift Instructions

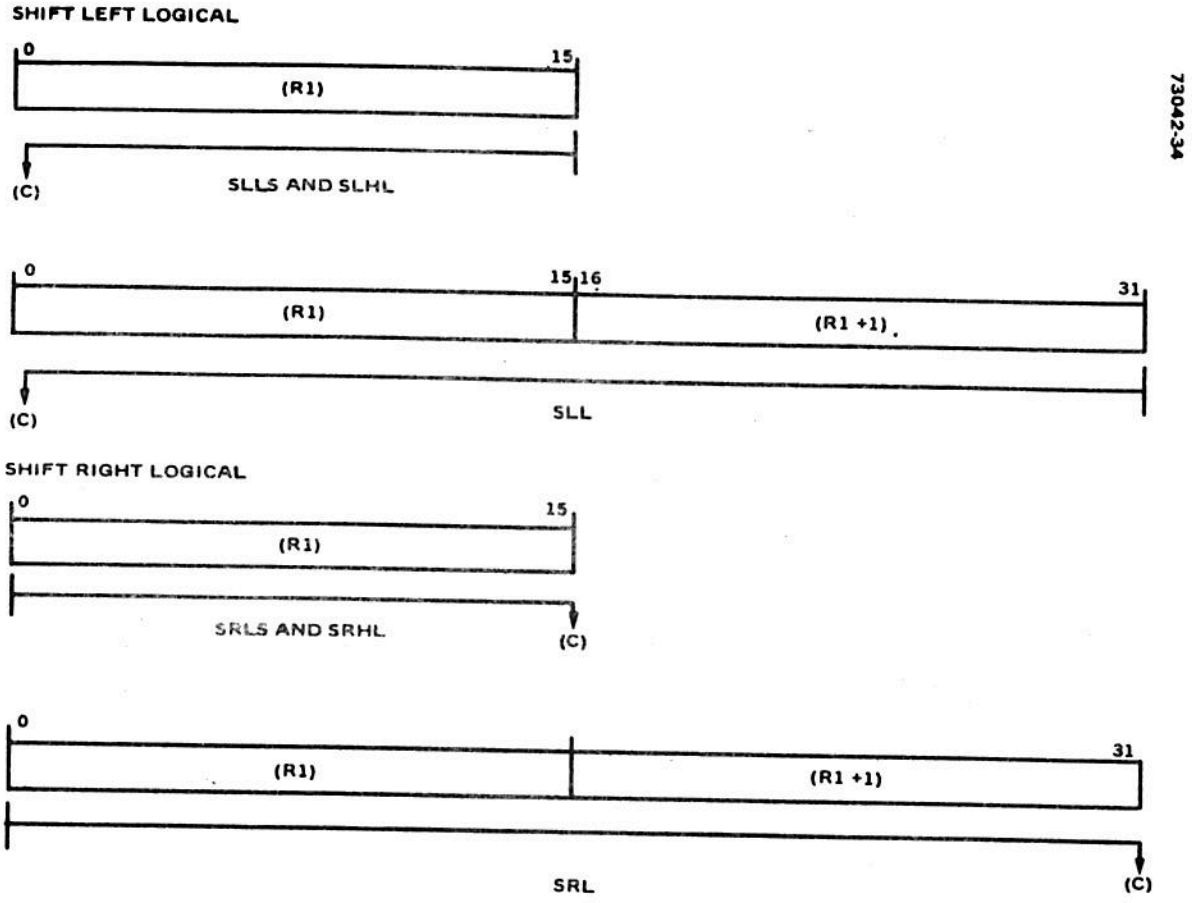


Figure 10.6-2. Logical Shifts Illustration

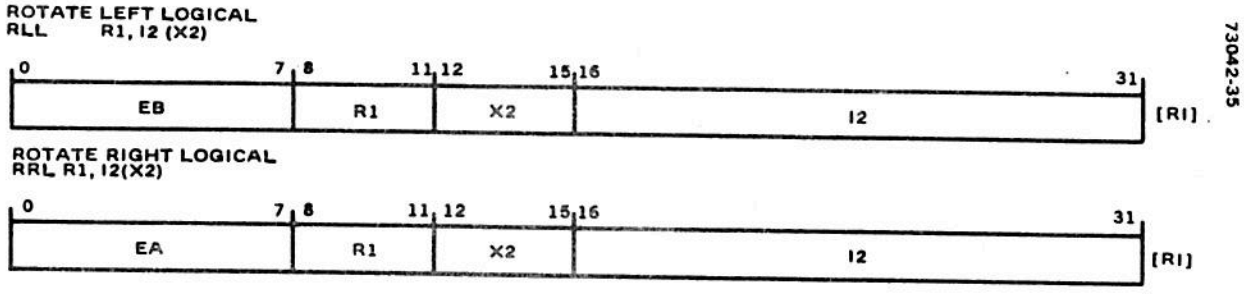


Figure 10.6-3. Logical Rotate Instructions

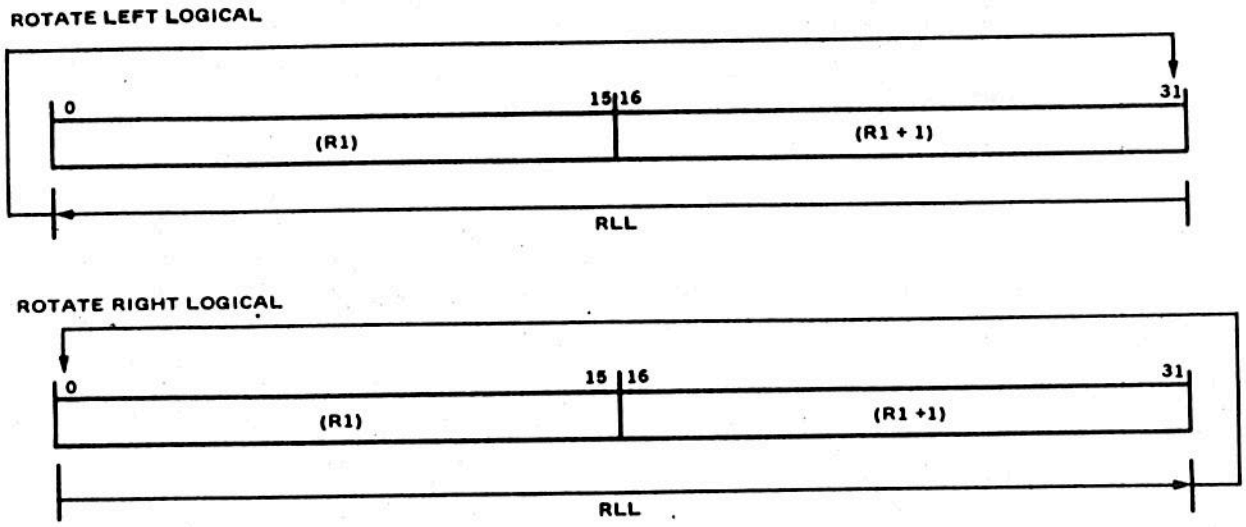


Figure 10.6-4. Logical Rotate Illustration

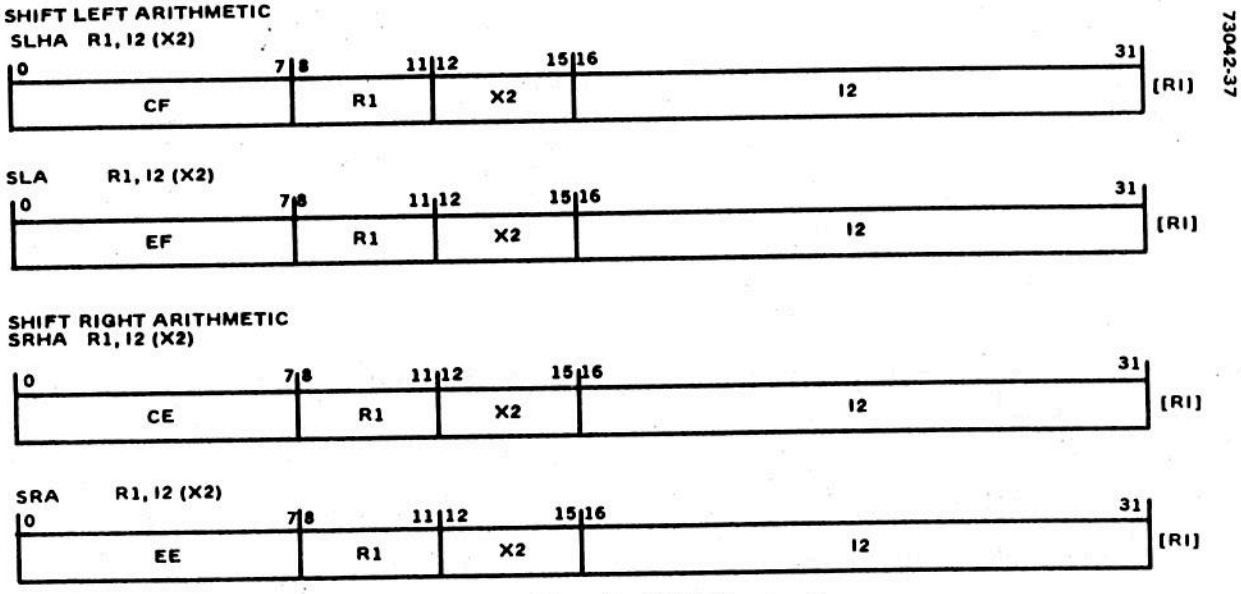
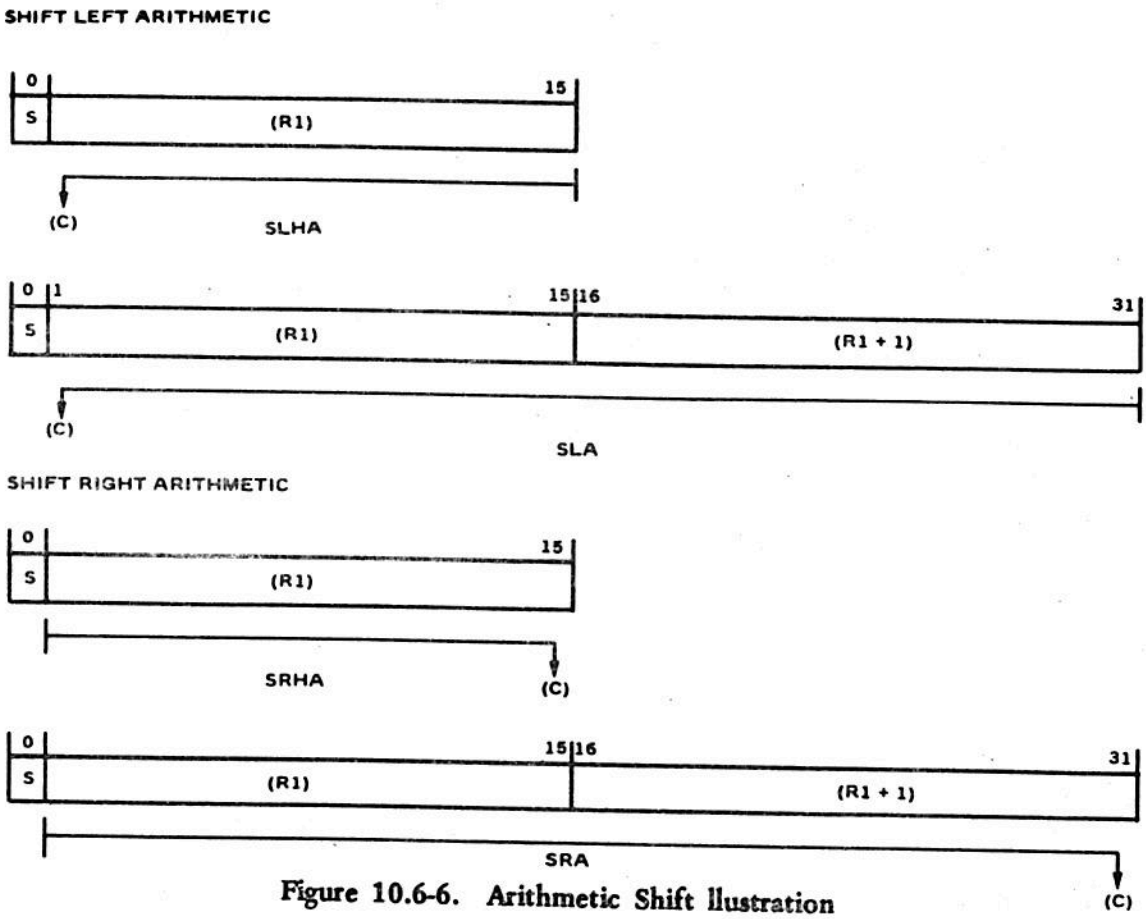


Figure 10.6-5. Arithmetic Shift Instructions



73042-38

Figure 10.6-6. Arithmetic Shift Illustration

b. 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 instructions or Position 31 for the fullword instruction shall be shifted thru 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 through 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 through 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.

c. 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 32-bits of the fullword shall be shifted. Bits shifted out of Position 0 shall be shifted into Position 31. A shift specification of 16-bits shall interchange the two halves (R1, R1+1) of the first operand. The Rotate Left Logical (RLL) 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.

d. 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 32-bits of the fullword shall be shifted. Bits shifted out of Position 31 shall be shifted into Position 0. A shift specification of 16-bits shall interchange the two halves (R1, R1+1) of the first operand. The Rotate Right Logical (RRL) instruction, rotates 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.

e. 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 through 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 through 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.

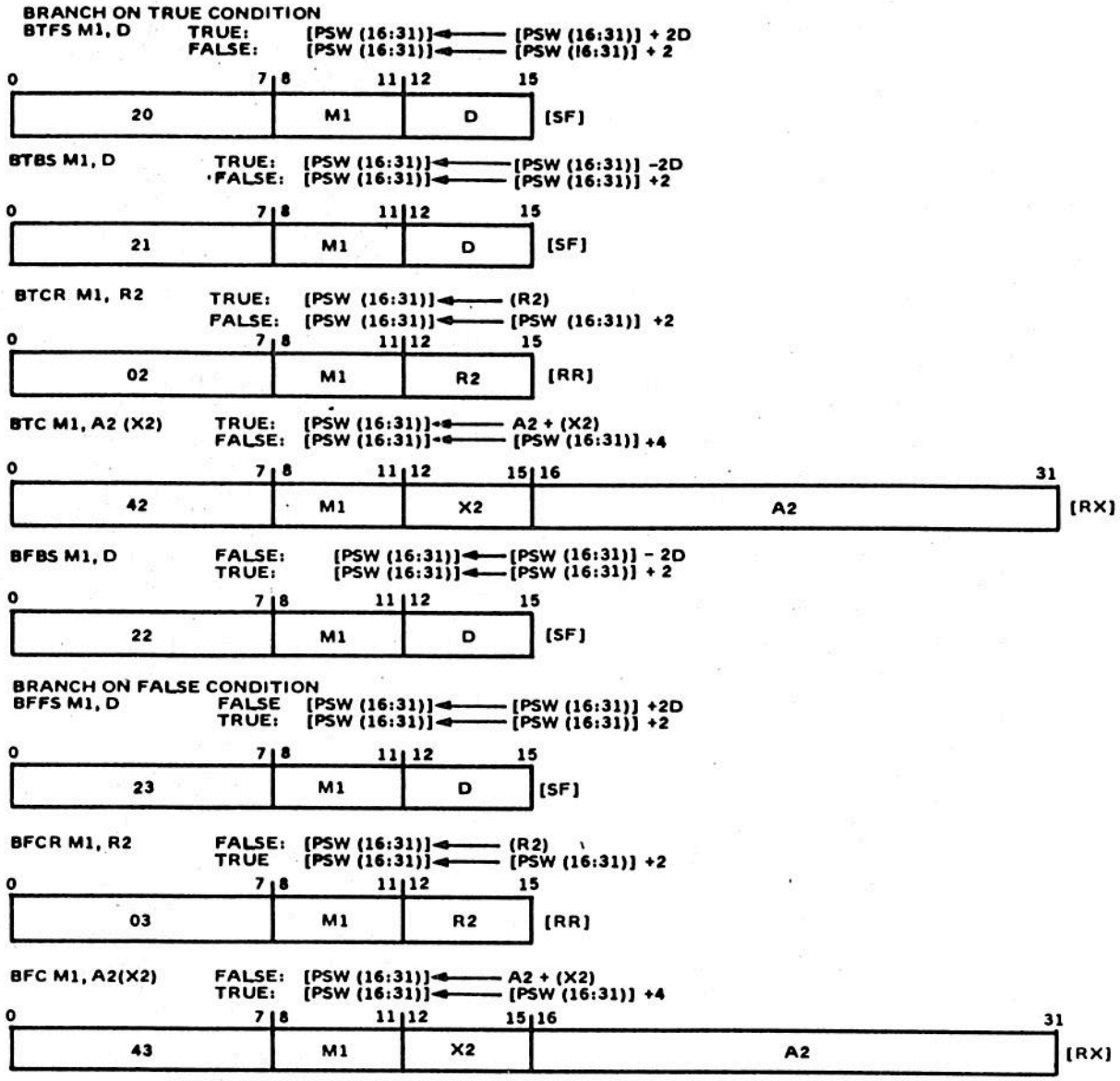
f. Shift Right Arithmetic: The Shift Right Arithmetic (SRA, SRHA) instruction shall cause the content of the first operand to be shifted right the number of bit positions specified by the second operand. The Sign Bit, Bit 0, of R1 shall be unchanged and shall be shifted right into Bit 1; therefore, Bit 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 through 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 through 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.

10.7 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 Program Status Word. The Branch on Condition instructions shall implement the testing of the Condition Code through use of a mask field contained in the instruction itself (M1 field). The 4-bit M1 field shall contain an image of the Condition Code to be tested. The Branch instructions shall use the RX, RR, and SF formats. The exact format, op-code, assembler notation and diagrammatic representation of each instruction shall be as shown in Figures 10.7-1, -2 and -3. The operation and resulting Condition Code shall be as follows:

a. Branch on True: The Branch on True (BTBS, BTFS, BTCR, BTC) instructions shall cause the Condition Code field of the Program Status Word 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 its 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 through 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 through 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, for example, in sense status loops used for program controlled I/O.

b. Branch on False: The Branch on False (BFBS, BFFS, BFCR, BFC) instructions shall cause the Condition Code field of the Program Status Word [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 changed. 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



72582-16R

Figure 10.7-1. Branch on True/False Instructions

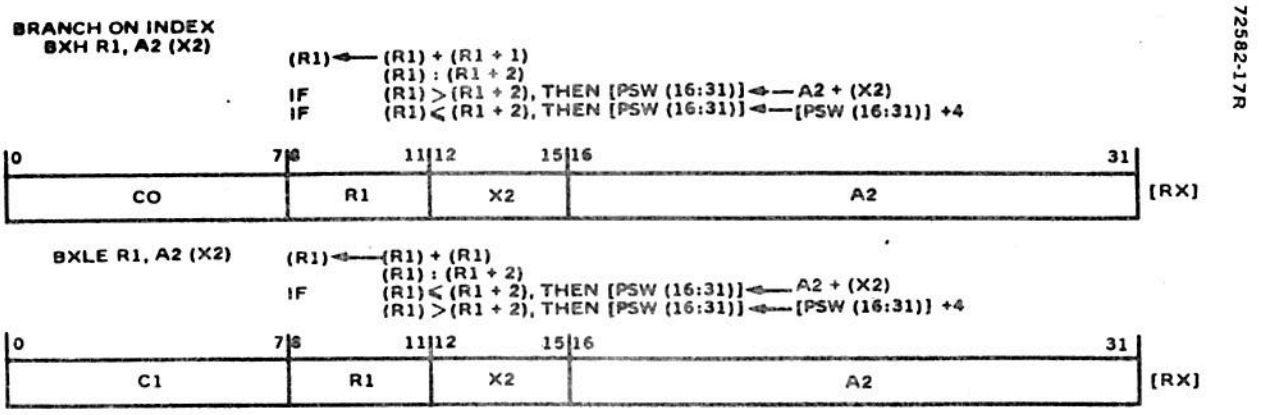


Figure 10.7-2. Branch on Index Instructions

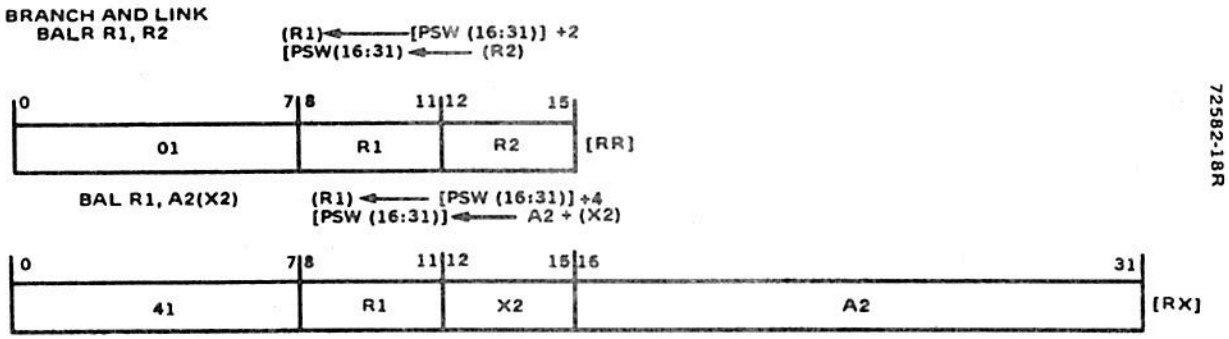


Figure 10.7-3. Branch and Link Instructions

Short (BFBS) 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 through 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 through 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. Branch on False Condition with a mask of 0 shall be an Unconditional Branch.

c. 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 index 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 (BXH) 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 Program Status Word [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 (BXLE) 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 Program Status Word [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 Branch on Index Low instructions are appropriate for rapid 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.

d. 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 to subprograms. It differs from the Branch Unconditional instruction in that the incremented Location Counter value is preserved in a specified General Register to be used as the subprogram

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.

10.8 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 Load instruction shall normalize an unnormalized floating-point number. The Floating-Point instructions shall manipulate 32-bit operands. The data format for the operands shall be as specified in 3.2.1.1.1.3. 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 10.8-1 and -2. The operation and resulting Condition Code shall be as follows:

a. 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 4 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.

b. 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.

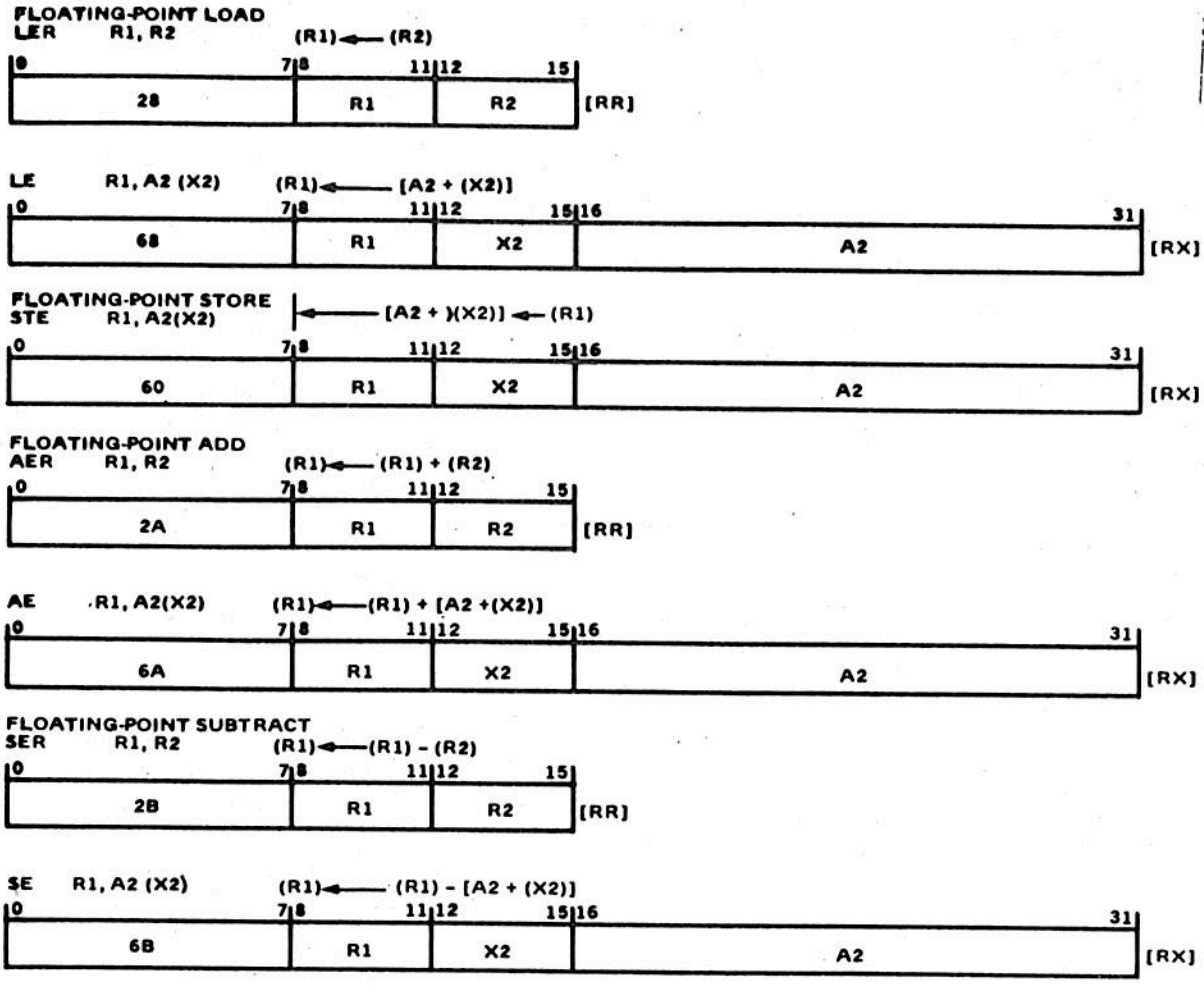


Figure 10.8-1. Floating-Point Load/Store/Add/Subtract Instructions

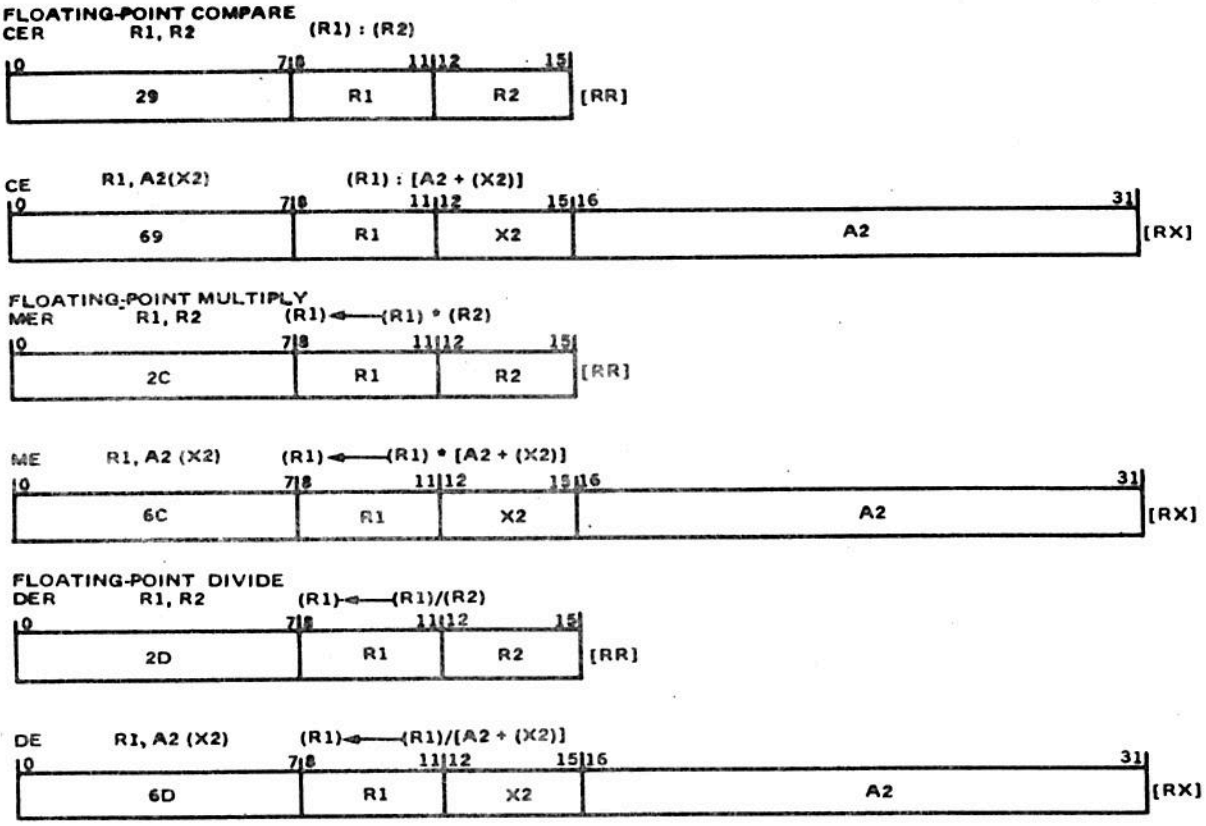


Figure 10.8-2. Floating-Point Compare/Multiply/Divide Instructions

c. 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 4 bits at a time, and its exponent shall be incremented by one for each shift until 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 4 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 4 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 generated by adding equal fractions with opposite signs, a true zero shall be generated. In the event of exponent overflow or underflow, the 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.

d. 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, the Floating-Point Arithemetic 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.

e. 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 algebraic, 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.

f. 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 Arithemetic Fault Interrupt is caused, if enabled by Bit 5 in 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 normalization, 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 required. After normalization, the product is rounded 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. For example, the multiplication $1/2 \times 16^{63} * 1 = 1/2 \times 16^{63} * 1/16 \times 16^{1} = 1/32 \times 16^{64}$ causes an overflow, rather than the result $1/2 \times 16^{63}$. 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.

g. 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 is caused, 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	
	1	0	0	Exponent underflow.
1	1	0	0	Divisor Equal to zero.

10.9 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 PSW's, 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 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 Figure 10.9-1. The operation and resulting Condition Code shall be as follows:

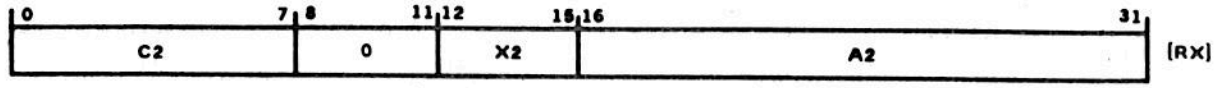
a. Load Program Status Word: The Load Program Status Word (LPSW) instruction shall cause a 32-bit operand to be loaded into the Current Program Status Word. The second operand shall remain unchanged. The resulting Condition Code shall be determined by the PSW loaded by the instruction. This instruction shall be privileged. The R1 field of a Load PSW instruction shall contain 0.

b. 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.

c. 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 registers specified in 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.

LOAD PROGRAM STATUS WORD

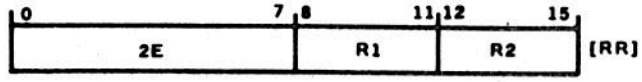
LPSW A2(X2) [PSW (0:31)] ← [A2 + (X2)]



72582-20R

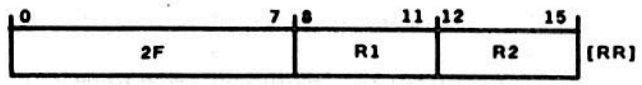
EXCHANGE OPERAND BANK ADDRESS

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



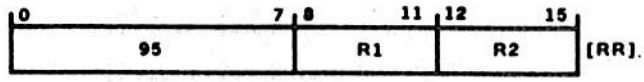
EXCHANGE PROGRAM ADDRESS

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



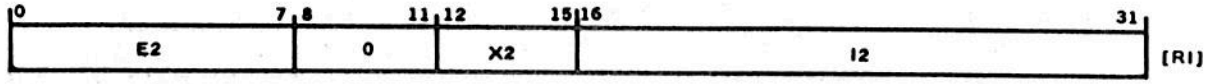
EXCHANGE PROGRAM STATUS

EPSR R1, R2 [PSW (0:15)] → R1
[PSW (0:15) ← R2



SIMULATE INTERRUPT

SINT I2 (X2)



SUPERVISOR CALL

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

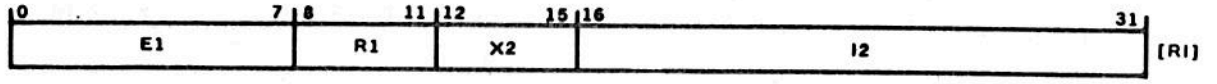


Figure 10.9-1. System Control Instructions

d. Exchange Program Status: The Exchange Program Status (EPSR) instruction shall cause the Current Program Status, PSW (0:15), to be stored into 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.

e. 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 as described in 3.2.1.1.9.4.c and 3.2.1.3.2.1 respectively. This instruction shall be privileged. The R1 field of a Simulate Interrupt instruction shall contain 0.

f. 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 Program Status Word 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'009C' through X'00BB' shall contain sixteen new Location Counter values, one for each type of Supervisor call. The type of Supervisor 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 Program Status Word 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 Program Status Word or Exchange Program Status instruction. The resulting Condition Code shall be defined by the New PSW.

10.10 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 30 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 4 through 7 of the device status. With standard Interdata device controllers, Bit 5 of the status byte, which is reflected in the V flag in the Condition Code, is defined as Examine Status. This means that status byte should be examined. Following Sense Status and Acknowledge Interrupt instructions, therefore, the occurrence of the V flag with status Bits 0 through 3 equal zero indicates instruction time-out. The I/O instructions shall use the Register to Register (RR), and the Register to Indexed Storage (RX) formats. The exact format, op-code, assembler notation and diagrammatic representation of each instruction shall be as shown in Figures 10.10-1, -2, -3, and -4. The operation and resulting Condition Code shall be as follows:

a. 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 eight-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:

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)

b. 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 Register specified by the first operand (R1) shall contain the device address. The device shall be addressed and the eight-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)

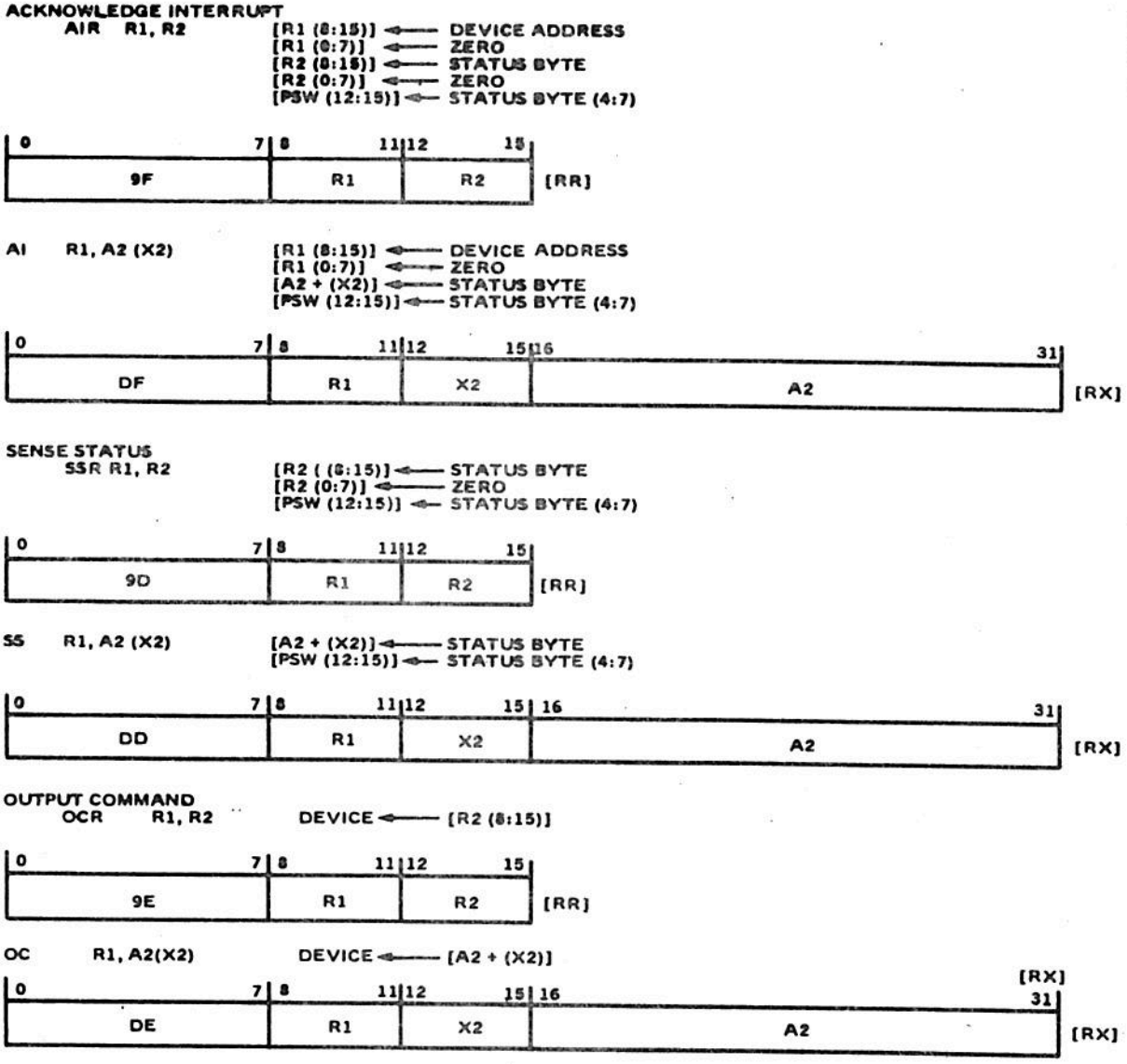


Figure 10.10-1. Acknowledge/Status/Command Instructions

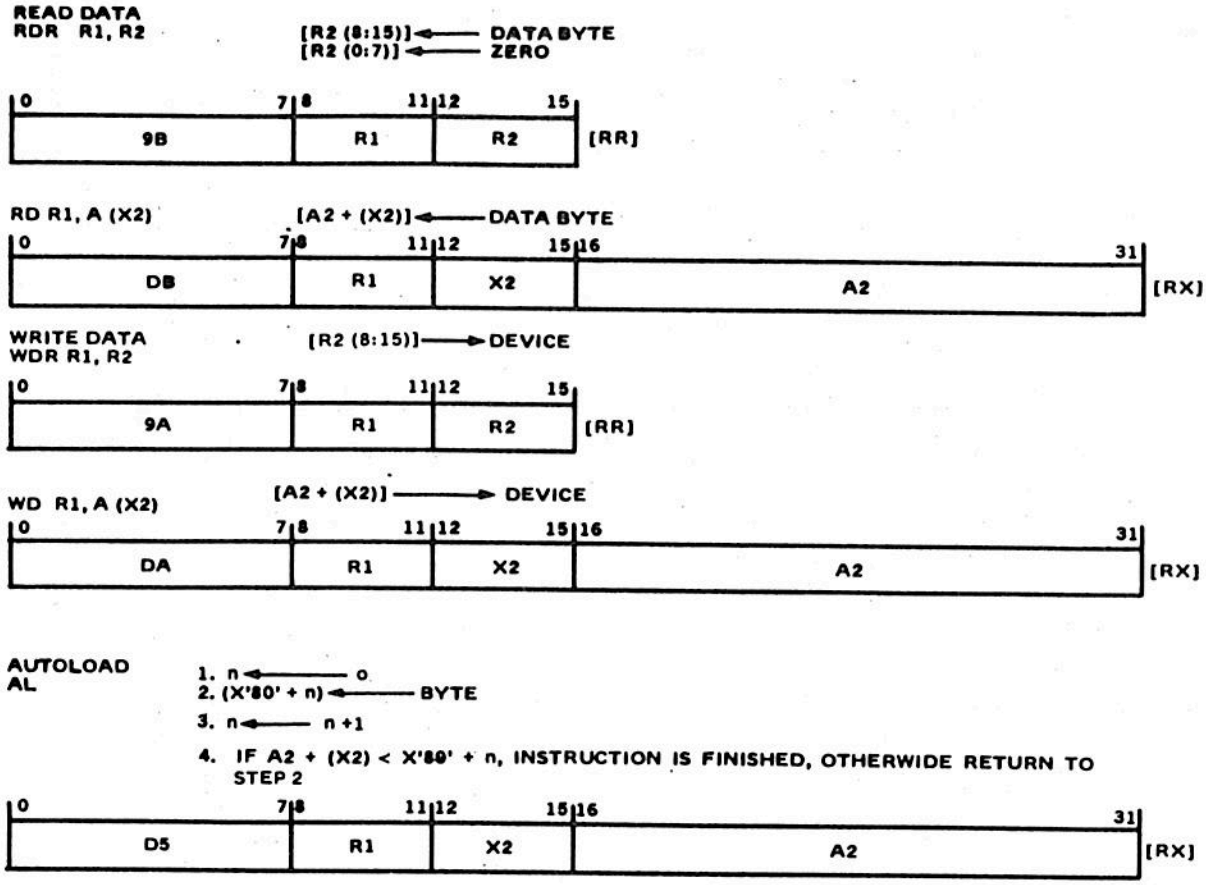
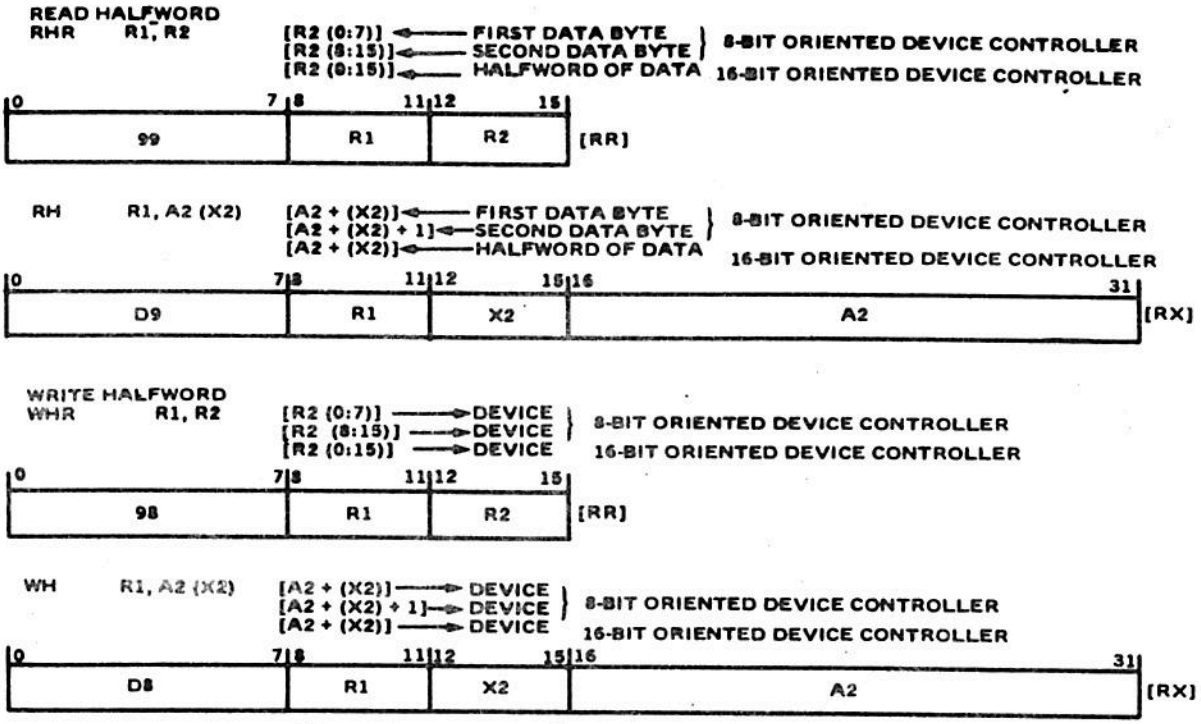


Figure 10.10-2. Byte I/O Instructions

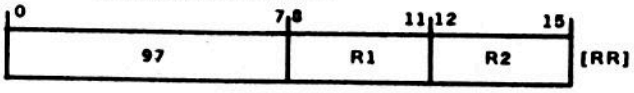


73046-47

Figure 10.10-3. Halfword I/O Instructions

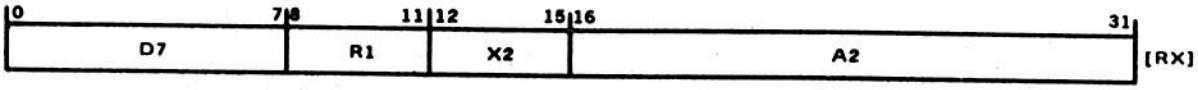
READ BLOCK
RBR R1, R2

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



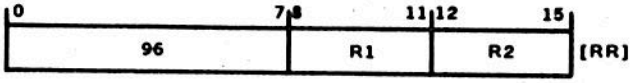
RB
R1, A2(X2)

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

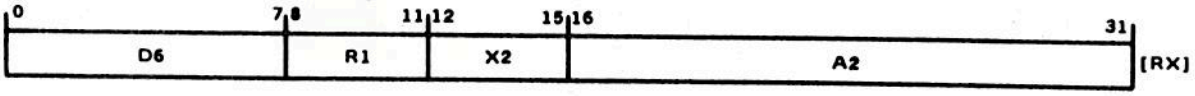


Figure 10.10-4. Block I/O Instructions

c. 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 eight-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 cannot complete the command action. These instructions shall be privileged.

d. 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 eight-bit data 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.

e. 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 eight-bit data 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.

f. 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 shall contain 0. This instruction should not be used with 16-bit oriented device controllers. For 16-bit oriented devices, use Read Halfword/Write Halfword instruction. 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)

g. 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 8-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.

h. 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 8-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.

i. 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 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 Program Status Word [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 complete correctly
1				Device Busy (BSY)
	1			Examine status (EX) or time out
		1		End of Medium (EOM)
			1	Device Unavailable (DU)

j. Write Block: The Write Block instructions (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 [A + (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 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 Program Status Word [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 are privileged. This instruction 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 is 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)

10.11 List Processing Instructions. The item shall execute the List Processing instructions to manipulate a circular list as defined in Figure 10.11-1. 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 0. 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 parameter byte shall indicate the number of slots (halfwords) in the entire list. The second parameter byte shall indicate the current number of slots being used. When this byte equals zero, the list shall be empty; 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 10.11-2. These pointers shall also be updated automatically.

These instructions shall use the Register to Indexed Storage (RX) format. The exact format, op-code, assembler notation and diagrammatic representation of each instruction shall be as shown in Figure 10.11-3. The operation and resulting Condition Code shall be as follows:

a. 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 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 changed, the element added

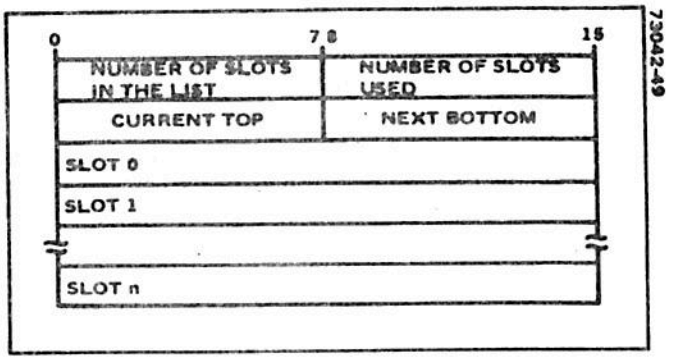


Figure 10.11-1. List Processing Instruction Format

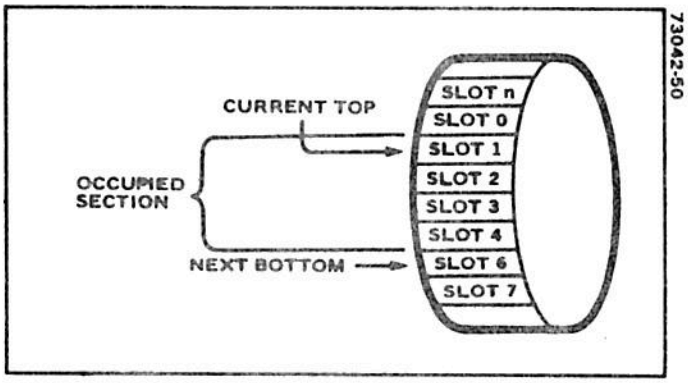


Figure 10.11-2. Circular List Instruction Processing

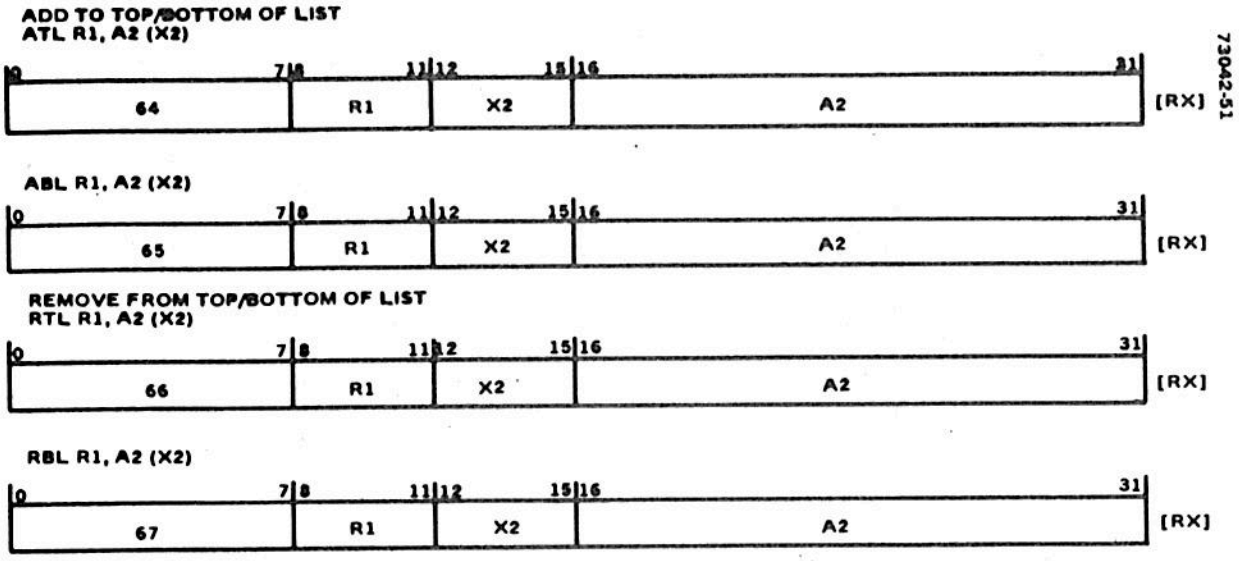


Figure 10.11-3. List Processing Instructions

to the list, and the instruction terminated with a Condition Code of zero. The Add to Top of List (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 Add to Bottom of List (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

b. 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

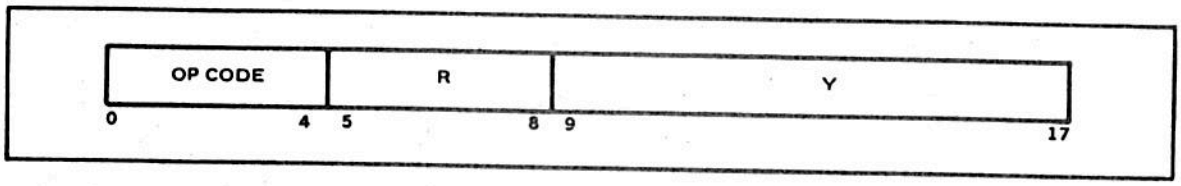
address of the list. If, on entering the instruction the "number of slots used" tally is zero, the list is already empty and the instruction terminate with the V flag set in the PSW. This condition shall be referred to as list underflow. If underflow does not occur the number of slots used tally shall be decremented by one, the appropriate pointer changed, and the element extracted and placed in R1. The instruction shall terminate with the 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 Remove from Top of List (RTL) instruction, shall manipulate the Current Top Pointer. 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 and compared to the maximum slot number. If the Current Top Pointer is greater than the maximum slot number, the Current Top Pointer shall be set to zero. This condition shall be referred to as list wrap. The Remove from Bottom of List (RBL) instruction, shall manipulate the Next Bottom Pointer. If no underflow occurred, and the Next Bottom Pointer is zero it shall be set to the maximum slot number (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

10.12 Central Computer Instructions

10.12.1 Instruction definition. The central computer 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 use of the 11 serial I/O instructions out of the set may not be applicable for the application described herein.

a. CONTROL OPERATIONS



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.

4. TOF – Transfer on Overflow (OP CODE 03). If the Overflow Indicator is on, the indicator is turned off and the computer takes the next instruction from the memory location specified by Y and R. If the Overflow Indicator is off, the computer takes the next instruction in sequence.

5. TSX – Transfer and Set Return Address in Index Register 12 (OP CODE 12). The contents of the program counter plus one are placed in index register 12 of the specified index register bank. The computer will then take its next instruction from the memory location specified by Y and R. The value placed into the index register represents the memory location immediately following the location of this TSX instruction. The transfer portion of the Transfer and Set Index instruction may utilize all of the indexing options.

6. CMA – Compare Algebraic (OP CODE 30). The contents of the accumulator are algebraically compared with the contents of the memory word at the location specified by Y and R. Two compare indicators, high, and low, are treated as follows: if the accumulator is lower, the low indicator is turned on and the other off; if the accumulator is higher, and high indicator is turned on and the other off. If they are equal, both are turned on. For the purposes of this instruction + zero is greater than - zero, $+377777_8$ is the highest number, and -377777_8 is the lowest number. Only the execution of another CMA, CML or PAI instruction will alter these indicators.

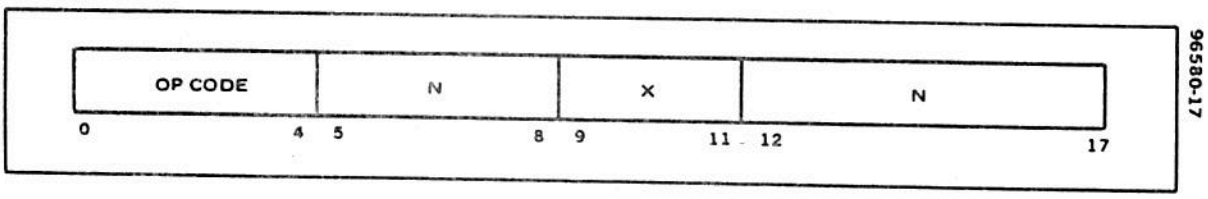
7. CML – Compare Logical (OP CODE 31). The entire contents of the accumulator, including sign, are logically compared with the contents of the memory word at the location specified by Y and R. Two compare indicators, high and low, are treated as follows: if the accumulator is lower, the low indicator is turned on and the other off; if the accumulator is higher, the high indicator is turned on and the other off. If they are equal, both are turned on. For the purposes of this instruction 000000_8 is the lowest number and 777777_8 is the highest number. Only the execution of another CMA, CML or PAI instruction will alter these indicators.

8. TRE – Transfer Equal (OP CODE 05). If both compare indicators are on, the computer takes the next instruction from the memory location specified by Y and R. If either compare indicator is off, the computer takes the next instruction in sequence. The execution of this instruction will not alter the state of the compare indicators.

9. TRH – Transfer High (OP CODE 06). If only the high compare indicator is on, the computer takes the next instruction from the memory location specified by Y and R. If the high compare indicator is off or both compare indicators are on, the computer takes the next instruction in sequence. The execution of this instruction will not alter the state of the compare indicators.

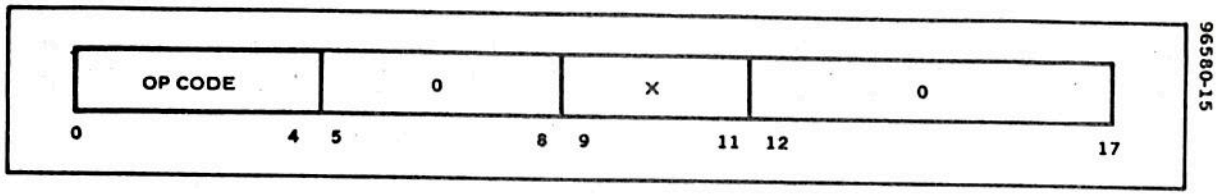
10. TRL – Transfer Low (OP CODE 07). If only the low compare indicator is on, the computer takes the next instruction from the memory location specified by Y and R. If the low compare indicator is off or both compare indicators are on, the computer takes the next instruction in sequence. The execution of this instruction will not alter the state of the compare indicators.

11. LCK – Lock (OP CODE 14) (X=4).



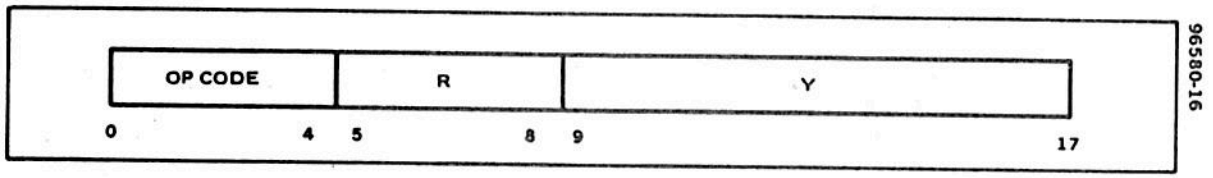
In a dual processor system, the value N is placed in the lock register of processor A. The number is then compared with the contents of the lock register of processor B. If equal, the lock register of processor A is reset and the computer will take the next instruction in sequence. If unequal, one instruction is skipped. In any case, the lock register and instruction sequence of processor B are unchanged. In case of both computers executing a sequence involving the lock instruction with the same number simultaneously, the processor having indicator 7 set at that time will have its lock register set first. In a single processor system with the ignore lock switch on, this instruction will act as a no-operation and the computer will take the next instruction in sequence.

12. ULK – Unlock (OP CODE 14) (X=0).



The lock register of the processor is reset. In a single processor system this instruction will act as a no-operation and the computer will take the next instruction in sequence.

b. ARITHMETIC OPERATIONS



1. ADD – Add (OP CODE 22). The contents of the memory location specified by Y and R are algebraically added to the contents of the accumulator. The result is placed in the accumulator. The overflow indicator will be set (turned on) if an overflow occurs. If the result is zero, the sign of the result is the original sign of the accumulator.

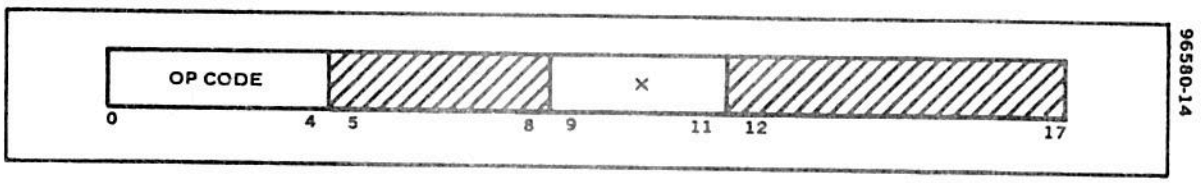
2. SUB – Subtract (OP CODE 23). The contents of the memory location specified by Y and R are algebraically subtracted from the contents of the accumulator. The result is placed in the accumulator. The overflow indicator will be turned on if an overflow occurs. If the result is zero, the sign of the result is the original sign of the accumulator.

3. MLY – Multiply (OP CODE 24). The contents of the memory location specified by Y and R are multiplied by the contents of the accumulator. The product, which is a double length word, is placed in the combined accumulator and 0 register. The high order bits are in the accumulator and the low order bits are in the 0 register. The sign of the product is in the sign bits of both the accumulator and Q register. Overflow is not possible. The initial value of the Q register does not affect the result of this instruction.

4. DIV – Divide (OP CODE 25). The double-length dividend in the combined accumulator and Q register is divided by the contents of the memory location specified by Y and R. Most significant bits of the dividend are located in the accumulator. The sign of the dividend is the sign of the accumulator. The single-length quotient is in the accumulator with the appropriate sign, and the remainder is in the Q register with the sign of the dividend. The divisor must be greater in magnitude than the higher-order half of the dividend. It is is not, the division does not take place and the divide check indicator is turned on, the accumulator and Q register remain unchanged, and the computer takes the next sequential instruction.

5. DPA – Double Precision Add (OP CODE 04). The double-length contents of Y + 1 and Y are algebraically added to the double-length word in the combined accumulator and Q register. Y + 1 contains the high-order bits of the double-length operand and Y contains the low-order bits. These two words must have the same sign. The double-length sum is left in the accumulator and Q register with the high-order bits in the accumulator and the low-order bits in the Q register. The accumulator and Q register must have the same sign. The overflow indicator will be turned on if any overflow occurs out of the accumulator. If the result is zero in both registers, the sign of the result is the original sign of the accumulator.

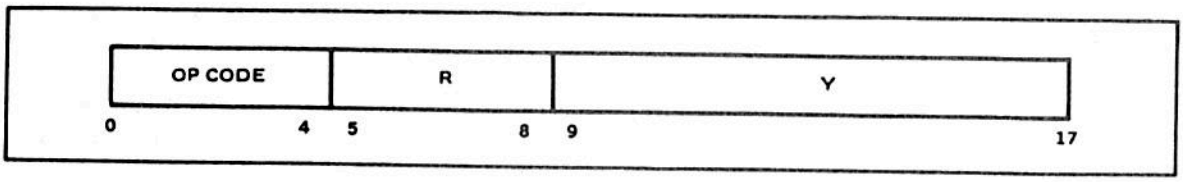
6. SQR – Square Root (OP CODE 37) (X=1).



The single-length square root of the double-length argument in the combined accumulator and Q register is put in the accumulator. The contents of the Q register are destroyed. If the sign of the agrument is negative (sign of the accumulator), the square root is not performed and the divide check indicator is turned

on, the accumulator and Q register remain unchanged, and the computer takes the next sequential instruction. The binary point for this instruction is located between Ao and A1.

c. WORD TRANSMISSION OPERATIONS



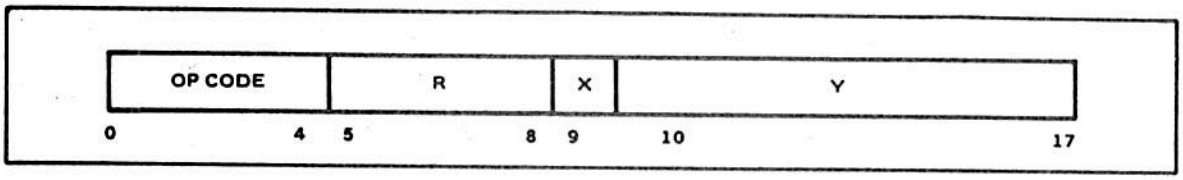
1. CLA – Clear and ADD (OP CODE 20). The contents of the memory location specified by Y and R replace the contents of the accumulator.

2. STR – Store Accumulator (OP CODE 26). The contents of the accumulator are stored in the memory location specified by Y and R. The contents of the accumulator remain unchanged.

3. LDQ – Load Q Register (OP CODE 21). The contents of the memory location specified by Y and R replace the contents of the Q register.

4. STQ – Store Q Register (OP CODE 27). The contents of the Q register are stored in the memory location specified by Y and R. The contents of the Q register remain unchanged.

d. INDEX REGISTER OPERATIONS



1. INX – Increment Index (OP CODE 10) (X=1). The Y field is added to the contents of the index register specified by R. If R=0 or 13, no operation will result. If R=14 or 15, the computer will add the 17 bit Y^1 field of the memory location specified by Y to bits 01-17 of index register 1 if R^1 is a one. Otherwise no operation will result. A negative Y value for relative addressing is not possible for this instruction.

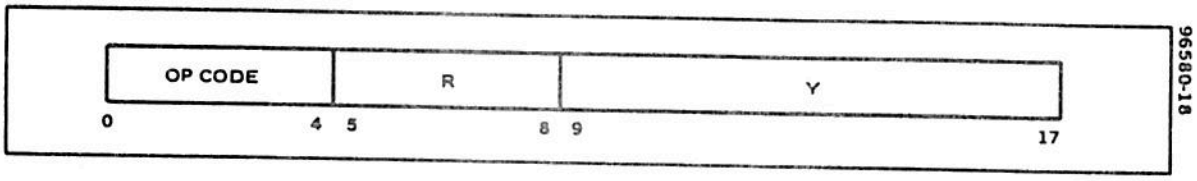
2. DCX – Decrement Index (OP CODE 10) (X=0). The Y field is subtracted from the contents of the index register specified by R. If R=0 or 13, no operation will result. If R=14 or 15, the computer will subtract the 17 bits of the Y^1 field of the memory

location specified by Y from bits 01-17 of Index Register 1 if R^1 is a one. Otherwise no operation will result. A negative Y value for relative addressing is not possible for this instruction.

3. TRX – Transfer on Index (OP CODE 13). If the contents of the index register specified by R are zero, the computer takes the next sequential instruction. If the contents of the index register specified by R are not equal to zero, the contents are decremented by one and the computer takes its next instruction from the memory location specified by Y, which is always relative. If R=14 or 15, the computer will use the 17-bit indirect address and index register 1 if R^1 is a one. If R=0 or 13, no operation will result and the computer takes the next sequential instruction.

4. SXH – Skip on Index High (OP CODE 11). The contents of the index register specified by R are compared to the number in the Y field. If the contents of the index register are greater than the value of Y, one instruction is skipped; otherwise, the computer will take the next sequential instruction. If R=14 or 15, the computer will use the 17-bit indirect address and index register 1 if R^1 is a one. If R=0 or 13, no operation will result and the computer takes the next sequential instruction.

e. LOGICAL OPERATIONS



1. LGA – Logical Add (OP CODE 32). Bits of the accumulator corresponding to 1's in the contents of the memory location specified by Y and R are set to 1. All other bits remain unchanged. This is the inclusive OR function.

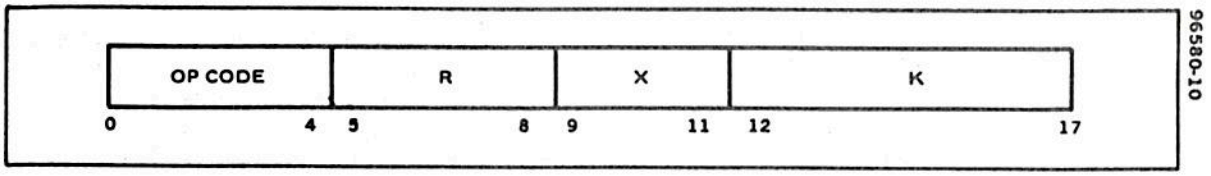
2. LGM – Logical Multiply (OP CODE 33). When corresponding bits of the accumulator and the contents of the memory location specified by Y and R are both 1's, those bits in the accumulator will remain 1. All other bits in the accumulator will be set to zero. This is the AND function.

3. LGC – Logical Complement (OP CODE 34). Bits of the accumulator corresponding to 1's in the contents of the memory location specified by Y and R are inverted. All other bits remain unchanged. This is the exclusive OR function.

4. CGC – Convert Gray Code (OP CODE 35). The contents of the memory location specified by Y and R are converted from gray

code to sign and magnitude, and the result is placed in the accumulator.

f. SHIFT OPERATIONS



1. LSA – Left Shift Accumulator (OP CODE 36) (X=1). The contents of the accumulator are shifted left the number of bits specified by K added to the contents of the index register specified by R, modulo 2^6. The sign bit is unchanged. Zeros are shifted into the least significant bit; bits shifted out of the most significant bit are lost and will cause an overflow if equal to 1. An overflow will turn on the overflow indicator.

2. LSB – Left Shift Both (Accumulator and Q Register) (OP CODE 36) (X = 3). The double-length word in the combined accumulator and Q register is shifted left the number of bits specified by K added to the contents of the index register specified by R, modulo 2^6. The sign bit of the Q register is unchanged and is always copied into the sign bit of the accumulator. Zeros are shifted into the least-significant bit of the Q register. Bits shifted out of the most-significant bit (sign not included) of the accumulator are lost and will cause an overflow if equal to 1. An overflow will turn on the overflow indicator.

3. RSA – Right Shift Accumulator (OP CODE 36) (X=0). The contents of the accumulator are shifted right the number of bits specified by K added to the contents of the index register specified by R, modulo 2^6. The sign bit is unchanged. Zeros are shifted into the most significant bit and bits shifted out of the least significant bit are lost. Overflow cannot occur.

4. RSB – Right Shift Both (Accumulator and Q Register) (OP CODE 36) (X = 4). The double-length word in the combined accumulator and Q register is shifted right the number of bits specified by K added to the contents of the index register specified by R, modulo 2^6. The sign bit of the accumulator is unchanged and is copied always into the sign bit of the Q register. Zeros are shifted into the most significant bit of the accumulator. Bits shifted out of the least significant bit of the accumulator are shifted into the most significant bit (sign not included) of the Q register. Bits shifted out of the least significant bit of the Q register are lost. Overflow cannot occur.

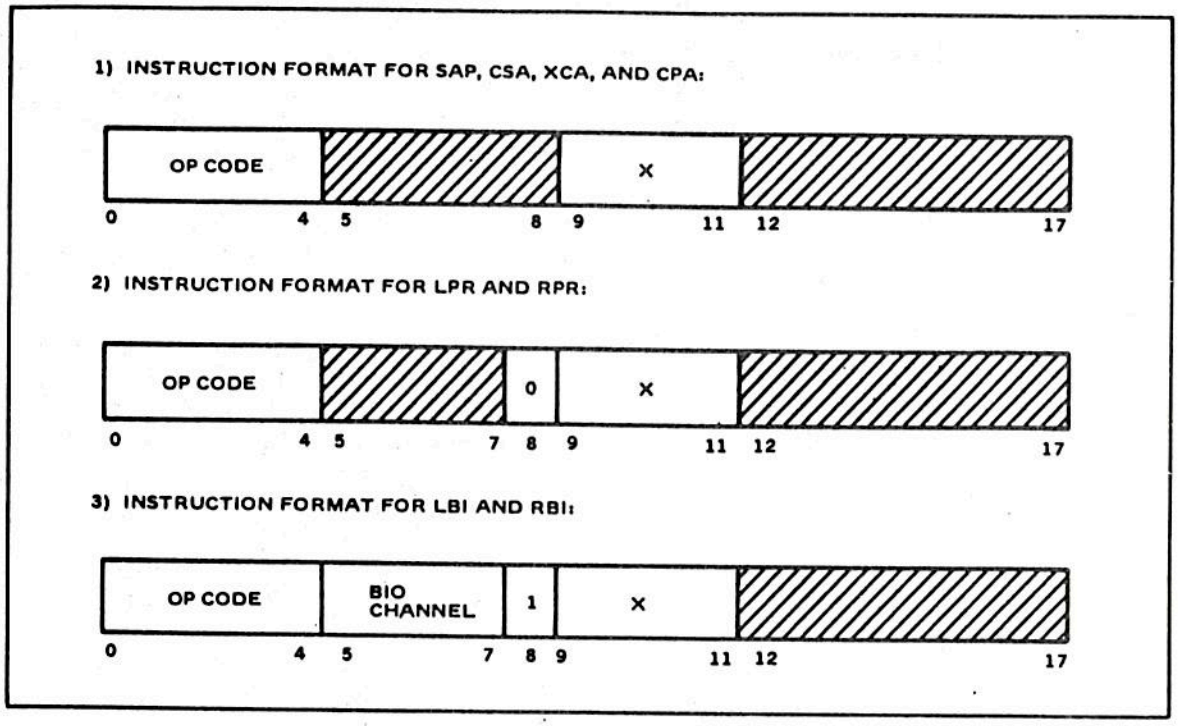
5. RRA – Right Rotate Accumulator (OP CODE 36) (X=5). The entire contents of the accumulator are shifted circularly to the right the number of bits specified by K added to the contents of the index register specified by R, modulo 2^6. The sign bit is included in the shift. Bits shifted out of the least significant bit are shifted into the sign bit. Overflow cannot occur.

6. RRQ – Right Rotate Q Register (OP CODE 36) (X=6). This instruction is similar to Right Rotate Accumulator, except that the Q register is shifted.

7. LRB – Left Rotate Both (Accumulator and Q register) (OP CODE 36) (X = 2). The double-length logical word in the combined accumulator and Q register is shifted circularly to the left the number of bits specified by K added to the contents of the index register specified by R, modulo 2^6. The sign bits are included in the shift. Bits shifted out of the bit sign of the accumulator are shifted into the least significant bits of the Q register. Bits shifted out of the sign bit of the Q register are shifted into the least significant bit of the accumulator. Overflow cannot occur.

8. RRB – Right Rotate Both (Accumulator and Q Register) Parity Generation (OP CODE 36) (X=7). The double-length logical word in the combined accumulator and Q register is shifted circularly to the right the number of bits specified by K added to the contents of the index register specified by R, modulo 2^6. The sign bits are included in the shift. Bits shifted out of the least significant bit of the Q register are shifted into the sign bit of the accumulator. Bits shifted out of the least significant bit of the accumulator are shifted into the sign bit of the Q register. Overflow cannot occur. During the execution of this instruction the ring sum of the bits shifted out of Q17 into the sign bit of the accumulator is set into sense indicator 2. The effect of this is to generate the parity of the field of bits shifted. If odd, sense indicator 2 is inverted, and if even, sense indicator 2 is not inverted. Sense indicator 2 should be reset by the program since the sense indicator is not automatically reset before execution of the instruction.

g. OPERATIONS ON ACCUMULATOR AND Q REGISTER



1. SAP – Set Accumulator Positive (OP CODE 37) (X=0). The sign bit of the accumulator is set to zero (positive).

2. CSA – Change Sign of Accumulator (OP CODE 37) (X=4). The sign bit of the accumulator is inverted.

3. XCA – Exchange Accumulator and Q Register (OP CODE 37) (X=3). The contents of the accumulator and Q register, including the sign bits, are exchanged.

4. CPA – Two's Complement Accumulator (OP CODE 37) (X=5). If the sign of the accumulator is negative, the contents of the accumulator, excluding sign, are replaced with the two's complement of that number. The sign remains unchanged. If the sign of the accumulator is positive, the accumulator remains unchanged.

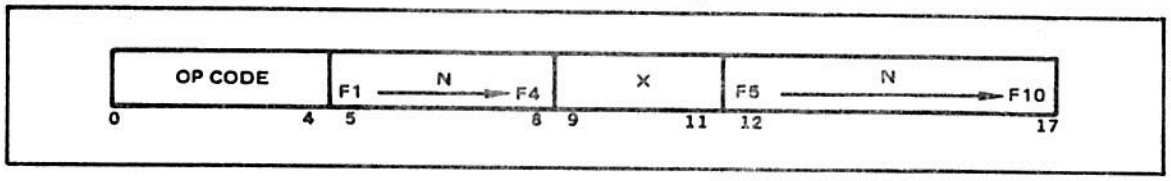
5. LPR – Load Page Register (OP CODE 37) (X=2). Page register whose address is specified by A register is loaded with the content of Q register, and the contents of A and Q are incremented by 1. A total of 255 page registers shall be loadable under program control. A load page register zero (0) instruction shall be treated as a NoOp. Bit 8 of the instruction must be set to zero (0).

6. RPR – Read Page Register (OP CODE 37) (X=6). Page register whose address is specified by A register is placed into the Q register, and the contents of A is incremented by 1. Bit 8 of the instruction must be set to zero (0). All 256 page registers shall be accessible by RPR.

7. LBI – Load Block Indicators (OP CODE 37) (X=2). The LBI instruction shall set the Block Indicators when bit 8 is set to one (1), with bits 5, 6, and 7 specifying the BIO channel. Bits 14, 15, 16, and 17 of the Q register shall be stored into the Block Indicators for the specified channel. Bits 14 and 15 shall be stored into spare Block Indicators and bits 16 and 17 stores into the usable Block Indicators (see 3.2.1.4.5).

8. RBI – Read Block Indicators (OP CODE 37) (X=6). The RBI instruction shall read the Block Indicators when bit 8 is set to one (1) and bits 5, 6, and 7 specifying the BIO channel. Block Indicators for the specified channel shall be stored into the Q Register bits 6, 7, 8, and 9. Bits 6 and 7 shall correspond to the spare Block Indicators (not used) and bits 8 and 9 correspond to the usable Block Indicators (see 3.2.1.4.5).

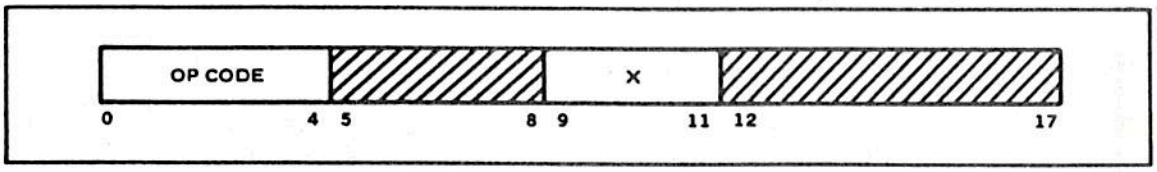
h. OPERATIONS ON INDICATORS AND SWITCHES (See 3.2.1.14 for description of indicators).



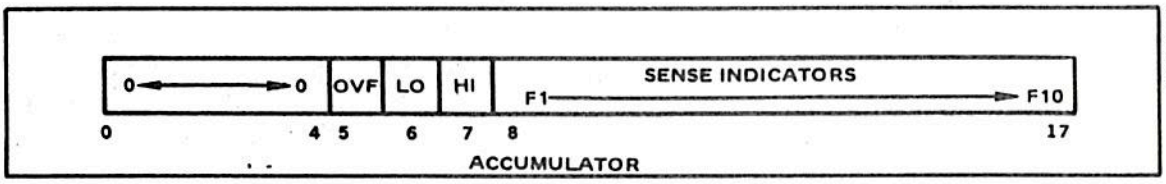
1. SEI – Set Indicators (OP CODE 14) (X=2). The indicators specified by a 1-bit in the corresponding position in the N fields are set (1). All other indicators remain unchanged. Each of the 10 bits in the N field corresponds to one of the 10 indicators, F1 through F10. Indicator F3 is not affected by this instruction. Indicator F7 is always set in the single processor configuration.

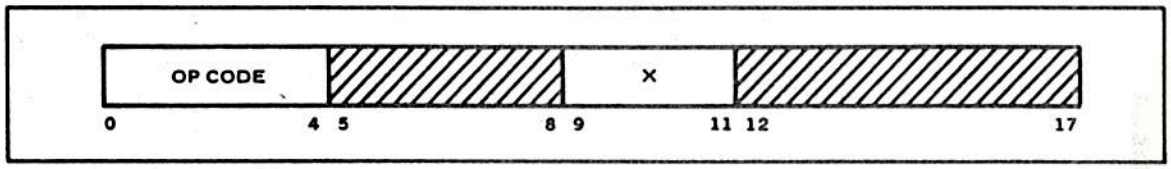
2. REI – Reset Indicators (OP CODE 14) (X=3). This instruction is similar to Set Indicators, except that the indicators specified by a 1-bit in the corresponding position of the N fields are reset (0). Indicator F3 is not affected by this instruction. Indicator F7 is always set in the single processor configuration.

3. SKI – Skip on Indicators (OP CODE 14) (X=1). If all indicators specified by 1 bits in the N fields are on (1), one instruction is skipped. If any of the specified indicators are off (0), the next instruction in sequence is taken. Each of the 10 bits in the N fields corresponds to one of the 10 indicators.

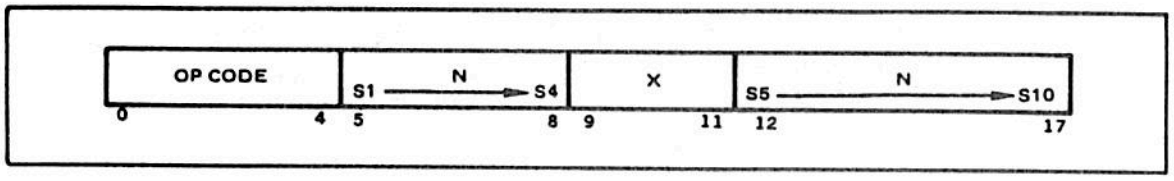


4. PIA – Place Indicators in Accumulator (OP CODE 14) (X=6). The contents of the thirteen sense indicators are placed in the accumulator as shown below. The remaining bits of the accumulator are cleared.



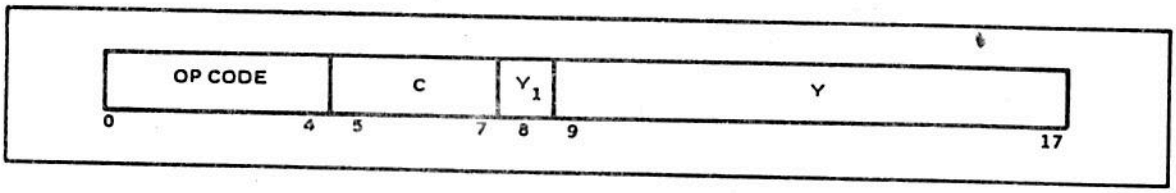


5. PAI – Place Accumulator in Indicators (OP CODE 14) (X=7). The 13 least significant bits of the accumulator are used to set or reset the sense indicators plus the two high-low indicators and the overflow indicator. Indicators 1, 3 and 7 are not changed by the instruction. The position of the bits in the accumulator is shown under the PIA (Place Indicators in Accumulator) instruction.



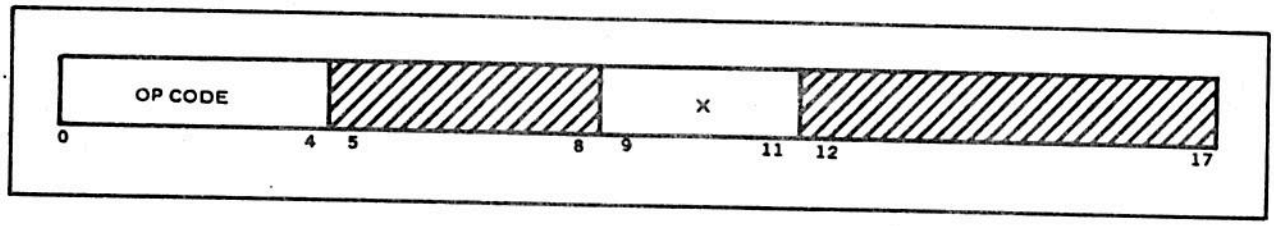
6. SKW – Skip on switches (OP CODE 14) (X=5). If all the switches specified by 1 bits in the N fields are on, one instruction is skipped. If any of the specified switches are off, the next instruction in sequence is taken. Each of the 10 bits in the N fields corresponds to the 10 sense control switches on the computer maintenance control panel.

i. BUFFERED I/O



1. EXF – External Function (OP CODE 17). If $Y_1 = 0$, the contents of the memory word specified by the absolute value of Y are sent via the input/output module to the device on the appropriate channel. If $Y_1 = 1$, the contents of the memory word specified by the program counter plus the value of Y are sent. C shall be coded 0 through 7, corresponding to one of the eight channels. If the channel designated should be in a four-channel group which does not exist, the instruction becomes a no-op. All EXF command words shall be resident in the first block of memory (address 0 to 131,071).

j. SERIAL I/O INSTRUCTIONS



1. TWA – Keyboard Activate (OP CODE 15) (X=6). The Keyboard Enable Light is turned on, and the Console Not Busy Indicator is turned off. This will enable the data terminal to allow keyboard inputs. The SIO unit translates the ASCII coded characters from the teleprinter to equivalent selectric character codes during Keyboard entry operations. No operation will occur if the Console Not Busy Indicator is off previous to this instruction, and the computer will continue with the next instruction in sequence.

2. TWF – Keyboard Off (OP CODE 15) (X=7). The Keyboard Enable Light is turned off and the Console Not Busy Indicator is turned on. This instruction will inhibit any attempt to input from the keyboard.

3. OTW – Output to Printer (OP CODE 16) (X-3). The BCD character in the least significant 6 bits of the Q Register is output to the teleprinter through the Console Buffer Register. The SIO unit translates selectric character codes to equivalent ASCII codes. The computer continues with the next instruction in sequence immediately after the character in the Q Register is placed in the Console Buffer Register. The Console Not Busy Indicator is turned off and remains so until the character has been typed. No operations will occur if the Console Not Busy Indicator is off previous to this instruction, and the computer will continue with the next instruction in sequence.

4. IPC – Input from Console (OP CODE 16) (X=0). If the 7 Level pushbutton on the Maintenance/Control Panel is not enabled (light off), the least significant 6 bits of the Console Buffer Register are placed in the least significant 6 bits of the Q Register. Bit positions 0 through 11 of the Q Register are not altered. If the 7 Level pushbutton is enabled (lighted), the least significant 7 bits of the Console Buffer Register are placed in the least significant 7 bits of the Q Register. Bit positions 0 through 10 of the Q Register are not altered.

5. CTA – Console Tape Advance (OP CODE 15) (X=1). CTA causes the SIO unit to input data into the computer from the block buffer if it has valid data. After the block buffer has been unloaded or if it contains no data at the time CTA is received, the SIO unit causes data to be read from the teleprinter cassette unit which is in the playback mode.

   The Console Not Busy Indicator is turned off. No operation will occur if the Console Not Busy Indicator is off previous to this instruction, and the computer will continue with the next instruction in sequence. The SIO unit issues control codes Load and DCI required by the teleprinter for this instruction.

6. CTS – Console Tape Stop (OP CODE 15) (X=2). CTS causes the SIO to complete inputting any character which is being processed at the time CTS is received. In addition, the SIO causes the teleprinter cassette unit, which is in the playback mode, to stop sending data to the computer.

   The Console Not Busy Indicator is turned on. The SIO issues the DC3 control code required by the teleprinter for this instruction.

7. CTR – Console Tape Rewind (OP CODE 15) (X=3). CTR causes the teleprinter to rewind the cassette which is in the playback mode. The tape motion will be stopped by the teleprinter when it senses clear leader. There will be a program interrupt upon completion of the rewind. No operation will occur if the Console Not Busy Indicator is off previous to this instruction, and the

computer will continue with the next instruction in sequence. The SIO issues the rewind control code required by the teleprinter for this instruction.

8. PMO – Punch Motor On (OP CODE 15) (X-4). PMO causes the teleprinter cassette unit which is to be used for recording to be placed in the Record-On Mode and enables it to begin recording data from the computer. The SIO unit issues control codes Load and DC2 necessary to control the teleprinter with this instruction.

9. PMF – Punch Motor Off (OP CODE 15) (X=5). PMF causes the SIO to fill the block buffer in the teleprinter to ensure that the last data bytes output to the teleprinter get recorded on tape. This is done by the transmittal of 86 NULL characters. In addition, PMF causes the teleprinter cassette unit which is in the record mode to cease recording. The SIO issues control code DC4 to control the teleprinter with this instruction.

10. OPA – Output 6 Bits to Tape Punch (OP CODE 16) (X=1). The least significant 6 bits of the Q Register are output to the data terminal through the Console Buffer Register. Odd parity, computed using these six bits and sense indicator F04, is also output to the teleprinter. The computer continues with the next instruction in sequence immediately after the 6 bits in the Q Register are placed in the Console Buffer Register. The Console Not Busy Indicator is turned off and remains so until the frame has been recorded. No operation will occur if the Console Not Busy Indicator is off previous to this instruction, and the computer will continue with the next instruction in sequence.

11. OPB Output 7 Bits to Tape Punch (OP CODE 16) (X=2). This instruction is similar to OPA (Output 6 Bits to Tape Punch) except that odd parity is computed using these 7 bits and sense indicator F04 and the least significant 7 bits in the Q Register are output to the teleprinter through the Console Buffer Register.

10.12.2 Illegal instruction operation. When the processor detects an illegal op code or an illegal R field (R=13, 14, 15 for shift instructions) it shall set an illegal instruction indicator on the maintenance/control panel. If the illegal instruction restart pushbutton is enabled from the maintenance/control panel, the processor shall store its program counter in temporary register T, the program counter is reset to location zero prior to the next instruction fetch, and processing continues from location zero. If illegal instruction restart is not enabled an illegal instruction is treated as a no-op and the next instruction is taken in sequence.