SLAC Scanner Processor

SSP II

User Guide

Hanoch Brafman, Jeff Olsen

SLAC

January 29, 1986

Andrew Daviel

TRIUMF

November 1993

SSP II SPECIFICATIONS

Package:
Double width FASTBUS Module

- CPU Board
- Control Board

Front Panel:

- LED's indicating:

  Slave on Cable - Yellow

  Slave on Crate - Yellow

  Master on Cable - Green

  Master on Crate - Green

  Internal SR (DONE) - Red

  SR on Cable - Red

  SR on Crate - Red

  CPU running - Red

  Single step/instruction mode - Red

  Power Fail - Red (optional)
- Locking toggle switch for Reset

  Resets the SSP only; Puts SSP to 'sleep'
- Push Button for Single step/instruction
- LEMO (NIM) input for START

  Starts SSP

  Similar to TRIGGER bit in CSR#0
- LEMO (TTL open collector) output for DONE

  Same as internal SR
- DIPswitch to select Cable Segment Geographic Address

Memory and registers:

- CSR#0

Read/Write from FASTBUS or the processor

- CSR#8

  Write only from FASTBUS

  Arbitration level common to both Crate and Cable segment

- Program memory - 4K words or 64K words, switchable

  Read/Write from FASTBUS

  Addressed in CSR space

  40000000 - 4000FFFF

- Data memory - 512K words, expandable to 2048K words

  Read/Write from FASTBUS or the processor

  Byte addressable from the processor

  Addressed in DATA space

  00000000 - 007FFFFC

  Addressed in CSR space

  80000000 - 807FFFFC

Broadcast modes:

- Case 1 (X'01') General - SSP attaches
- Case 2 (X'N5') Class N - N is loaded into CSR#0
- Case 3a (X'19') SSP asserts TP on segment issuing the broadcast if not busy
- Case 4 (X'0D') SSP asserts TP on segment issuing the broadcast
- Case 5 (X'1D') SSP asserts TP if it is asserting SR on segment issuing the broadcast
- Case 6 (X'2D') SSP asserts TP if DONE (SR flag) is set
- Case 7 (X'{8-F}D') User may enable by burning a PROM

Data Cycles:

- Secondary Address (Read/write)
- Random Data (Read/Write)
- Block Transfers - Handshake (Read/Write)

Timing – Typical for Backplane

Slave:

Primary Address: AK UP response to AS UP - 40 ns

Random Data : DK UP response to DS UP - 215 ns read
DK UP response to DS UP - 140 ns write
DK DN response to DK DN - 50 ns

Block Transfer : DK T response to DS T - 120 ns

Master:

Random Data : DS DN response to DK UP - 140 ns read
DS DN response to DK DN - 50 ns write

Block Transfer : DS T response to DK T - 110 ns

Maximum Transfer rate (Slave response = 0 ns): 9 Mhz (110 ns/Word).

SS Responses:

- SLAVE

  SS = 0, Normal

  SS = 1, SSP is busy

  SS = 6, Addressing error

- MASTER

  Any SS other than 0 will generate an error except where SS=2 terminates the block transfer.

Parity:

- No parity checking is done

Reset Bus:

- Issued by another Master:

  Resets CSR#0. The value of N in CSR#0 remains unchanged.

  Puts SSP to 'sleep'.

  Leaves all memory and other registers unchanged.

- Issued by the SSP:

Ignored by the SSP

Segment determined by a bit in CSR#0

Level or Pulse selected by jumper; if level, Duration determined by setting and resetting a bit in CSR#0

- Issued through the scanner

  Ignored by the SSP

  Segment determined by a bit in CSR#0

  Duration determined by setting and resetting a bit in CSR#0

START:

- CSR#0 Trigger
- TTL - LEMO

Multiple starts are ignored while the processor is running

CPU:

- 32 Bit
- Emulates IBM integer machine language instructions. All instructions are 32 bits long.
- Separated Data and Program memory.
- Executes single or combined FASTBUS I/O operations.
- Basic cycle time of 100 ns. Narrowed to 80 during multiply, divide, and shifts.
- Users I/O and Error handler starts at location 0 in Program memory. Updated PSW in location 0 in data memory. Memory Address register contents in location 4 in Data memory.
- Interruptible from either the Crate or Cable segment at the end of an instruction or group of I/O instructions. System mask must be on.

SCANNER CSR's

The SSP has only 2 CSR's, CSR#0 and CSR#8. CSR#0 is a read/write register, CSR#8 is a write only register.

CSR#0: The format of CSR#0 is shown in Table 1.

Table 1: Format of CSR#0

Bit	Hex	Read	Write
31	80000000		Disable EPM
30	40000000	Bits 16-31	Reset (disable Class N)
29	20000000	Manufacturer's	
28	10000000	I.D.	Enable EPM
27	08000000	0106 (no EPM)	Enable Class N response
26	04000000		Enable Time-outs
25	02000000	0107 (with EPM)	Disable CI SR
24	01000000		Enable CI Mastership
23	00800000		Disable GK up
22	00400000		Disable single cycle
21	00200000		Reset SR
20	00100000		Disable CR SR
19	00080000		Reset RB
18	00040000		Reset RUN, HALT
17	00020000		Enable CI RB
16	00010000	EPM enabled	Enable writing of class
15			
14	0000X000	Bits 12-15	Bits 12-15
13		Class for case 2	Set Class for case 2
12		Broadcasts	Broadcasts
11	00000800	Class N enabled	Trigger
10	00000400	Time-outs Disabled	Disable Time-outs
9	00000200	CI SR Enabled	Enable CI SR
8	00000100	CR Mastership Enabled	Enable CR Mastership
7	00000080	Hold GK Enabled	Enable Hold GK
6	00000040	Single cycle mode	Set single cycle mode
5	00000020	SR set (DONE)	Set SR
4	00000010	CR SR Enabled	Enable CR SR
3	00000008	Cable SR is set	Set RB
2	00000004	RUN Bit Set	Set RUN Bit
1	00000002	Crate SR is set	Enable CR RB
0	00000001	Power fault	Set Soft Halt

CR - Crate segment, CI - Cable segment

I.D.
On reads, bits 16 through 31 indicate the manufacturer's I.D. For the SSP, the I.D. is 0106(hex) with EPM disabled. For EPM enabled, the I.D. is 0107.

EPM
Extended Program Memory (EPM) may be enabled or disabled by writing bits 28 and 31. EPM is disabled at power-up. See page 12 for a more complete description.

Reset
Response to Class N broadcasts may be enabled or disabled. In accordance with the FASTBUS specification, Class N response is disabled at power-up and by writing bit 30 (the Reset bit). Class N response is enabled by writing bit 27 (since CSR#7 is not implemented). Class N enabled is indicated by bit 11.

Time-outs
Time-outs are disabled by setting bit 10. and enabled by setting bit 26. With time-outs disabled, the SSP will wait forever to get responses back from a slave. With time-outs enabled, the SSP will wait 20.5 μs for responses from a slave. After this time period, the I/O-Interrupt handler will be called.

SR
The internal SR line is set true by setting bit 5. and set false by setting bit 21. If the internal SR line is true, the SSP will assert TP for broadcast case 5.

The internal SR line can be asserted on either the cable segment or crate segment or both. Setting bit 9 enables the internal SR onto the cable segment, bit 25 disables it. Setting bit 4 enables the internal SR onto the crate segment, bit 20 disables it.

BUS Mastership
The SSP can be a master on either the cable segment or the crate segment. Setting bit 8 will enable mastership on the crate segment. Setting bit 24 will enable mastership on the cable segment.

Hold GK
The Hold GK feature enables the SSP to maintain GK asserted after dropping AS/AK lock. This allows the SSP to scan from module to module without having to regain bus mastership. If the Hold GK feature is disabled, the SSP will drop GK upon receiving AK. Setting bit 7 will enable the Hold GK feature, bit 23 will disable it.

Single cycle
The SSP program can be stepped through by setting bit 6 and then using the front panel button. This feature is useful for debugging hardware. Setting bit 22 will disable the single step feature. A jumper on the control board controls whether the SSP executes a single clock cycle or a single instruction on each push of the front panel button.

Reset BUS
Setting bit 3 will set the internal RB line true. Setting bit 19 will set the internal RB line false. Setting bit 1 will direct the RB to the crate segment, bit 17 will direct the RB to the cable segment.

This allows the cable segment to issue a RB into the crate segment through the SSP. Setting bits 1 and 3 from the cable segment will set RB in the crate. Setting bit 19 will drop the RB in the crate. A jumper on the control board controls whether setting bit 3 generates a pulse on RB or a level.

RUN
Setting bit 2 will enable the SSP to RUN. The SSP must be triggered by either setting bit 11, trigger bit, or a front panel start. Setting bit 18 will halt the SSP.

Case 2 Broadcasts
The value of the class for case 2 broadcast is written into bits 12 through 15. This register is not of the J-K type. In order to load the class register, bit 16 must also be set. If bit 16 is not set, bits 12 through 15 are not loaded into the class register. Case 2 broadcast response is enabled by writing bit 27.

TRIGGER
bit 11 is the Trigger bit. This bit will start the SSP if the RUN bit (bit 2) was also set.

Power Fault
bit 0 indicates a power fault condition. Voltages on each fused section of both the SSP CPU and Control boards are monitored and a power fail signal is asserted if any voltage falls below tolerance. Indication is made by LEDs fitted to each board, optionally by a front-panel LED, and by setting bit 0. If a power fault exists on the CPU board the FASTBUS slave interface will still function. However a fault on the Control board will probably render the slave interface inoperative.

CSR#8:
CSR#8 is a write only register which contains arbitration level and the assured access bit.

I/O INTERRUPT AND ERROR HANDLER

The I/O-Error Handler is a program that is executed when the SSP is externally interrupted or encounters an error. Hardware-wise, entering the I/O-Error Handler does the following:

1. Stores the Processor Status Word (PSW) into DM 0.
2. Stores the Memory Address Register (MAR) into DM 4.
3. Sets the System Mask and Overflow Mask OFF; Disables Interrupts
4. Jumps to PM 0 and starts executing the I/O-Error Handler.

It is up to the I/O-Error Handler to determine what action to take in response to the I/O-Error. The reason that the I/O-Error Handler was entered can be decoded from the PSW and the MAR.

I/O Interrupts:
In order for the SSP to be interruptible, the System Mask must be ON. Turning ON the System Mask is done by setting bit 15 ON in the Initial PSW (if EPM is disabled) or by using the Set System Mask (SSM) instruction. If the System Mask is ON, trying to attach to the SSP from either the Crate or Cable Segment will interrupt the SSP. If the SSP is executing a sequence of I/O instructions, the interrupt will not be acknowledged until a non-I/O instruction or an instruction having a Primary address is executed. If the SSP is not executing I/O instructions, the interrupt will not be acknowledged until the end of the current instruction.

Bit 31 of the PSW indicates that the I/O-Error Handler was entered because of an I/O Interrupt. It is the responsibility of the I/O-Error Handler to put the SSP to 'sleep' by writing to CSR0 and resetting the RUN bit.

If the System Mask is ON and a Master attempts to attach to the SSP, the SSP will return AK and WT during the Primary Address cycle. WT will be released only after the I/O-Error Handler has reset the RUN bit in CSR0.

If the System Mask is OFF and a Master attempts to attach to the SSP, the SSP will return AK during the Primary Address cycle. The SSP will return an SS=1, BUSY, for any subsequent address or data cycles.

To continue executing the program from the point it was interrupted, DM 0 must be loaded with the correct PSW. This PSW is either the same PSW that was loaded into DM 0 when the interrupt occurred, or one less than it. The NPA Bit in the PSW indicates whether to subtract 1 or not. If the NPA Bit is 1, subtract 1, otherwise subtract 0. After the correct PSW is loaded into DM 0, set the RUN Bit and the Start Bit in CSR#0 to continue execution.

To start the SSP running in a different location than where the interrupt occurred, load the desired PSW into DM 0. Then set the RUN and Start Bit in CSR#0 to start the SSP.

ERRORS:
Besides handling I/O Interrupts, the I/O-Error Handler must also handle the following error condition:

1. Unsupported Op-Code
2. Divide by Zero
3. Overflow, if the Overflow Mask is ON
4. An unexpected non-zero SS code
5. A word-count overflow during block transfers using the word count as a guard.

EXTENDED PROGRAM MEMORY

Extended Program Memory (EPM) refers to a PM size of 64kW instead of 4kW. This is achieved by using a 16-bit PC instead of a 12-bit PC. This extended PC is too large to be saved in the PSW, so two instructions must be used instead of one.

In order to remain compatible with existing SSP software, EPM is initially disabled on power-up. When EPM is disabled, only the lower 4k words of program memory may be accessed, and the Program Counter (PC) is 12 bits long. When EPM is enabled, the full 64k words of program memory is available, and the PC is 16 bits long.

Initial Program Load (IPL) loads the PC with the PM address found in Data Memory address zero (DM 0), and also sets the system masks (LSMSK, LPMSK). Programs compiled for EPM must have an initial DM 0 of 0080, not 8080, and the system mask must be set with an SSM instruction if required. A DM 0 of 8080 would cause the SSP to start executing at address 8080 in addition to setting the mask.

When EPM is enabled, the instructions BAL, BALR, BALR0 save a 16-bit PC and do not save the program masks. Thus the link address is saved in the register, but the original values of the condition codes and system masks are lost. If these are required, they must be explicitly saved beforehand using a BALR0M instruction.

When EPM is enabled, the interrupt microcode (IOERR) clears the system mask (disabling interrupts) and saves a 16-bit PC. The error handler code must perform a BALR0M instruction to save the condition codes. The previous value of the system mask is lost, but if a consistent procedure is used to set and clear the mask then a copy may be kept in data memory. On exit from the error handler, an LMSK instruction should be executed to restore the condition codes and masks, followed by an LPC instruction to restore the PC. Executing an LPSW instruction while EPM is enabled would cause the upper PC bits to be set to the CC and masks values.

With EPM enabled, a base register must be used (as in the IBM 370) to access the upper 60kW of Program Memory.

PSW FORMAT

The format of the PSW is shown in Table 2.

Table 2: Format of the SSP PSW

Bit	Hex	Name	Meaning
31	80000000	INTRP	I/O INTERRUPT
30	40000000	EXCP	EXCEPTION
29	20000000	T.O.	I/O TIME-OUT
28	10000000	SS2	SS2, SS1 and SS0
27	08000000	SS1	indicate the FASTBUS SS code
26	04000000	SS0	
25	02000000	BRQ	BUS request by the SSP
24	01000000	EG	Indicates a Primary Address cycle
23	00800000	AS/AK	AS/AK lock was established
22	00400000	MS1	MS1 and MS0 indicate the FB MS code
21	00200000	MS0	
20	00100000	RDOT	Indicates a FASTBUS Read cycle.
19	00080000	WTOT	Indicates that SSP was issuing Wait
18	00040000	EXCP2	EXCP2 and EXCP1
17	00020000	EXCP1	indicate the type of EXCEPTION
16	00010000	/NPA	Next Prom Address
15	00008000	SMSK	System Mask
14	00004000	OVMSK	Overflow Mask
13	00002000	ECC2	ECC2 and ECC1
12	00001000	ECC1	Encoded Condition Codes
11			If EPM is disabled, bits 15:12
10			are the values of masks and
9			condition codes.
8			
7	00000XXX		If EPM is enabled, bits 15:12
6			indicate the upper 4 bits of the PC
5			
4			
3			Bits 11:0 indicate the PC
2			at the time the
1			I/O-Error handler was entered
0			

INTRP
Indicates that the I/O-Error handler was entered due to an I/O interrupt.

EXCP
Indicates that the I/O-Error handler was entered due to an EXCEPTION. EXCP2 and EXCP1 (bits

18 and 17 of the PSW), and A17 (the MSB of the MAR, for a given memory size), indicate the type of exception.

The exception is determined from Table 3.

Table 3: Exceptions

EXCP2	EXCP1	Explanation
0	0	Divide by zero if A17 is low, Data Memory wrap-around if A17 is high
0	1	Arithmetic Overflow
1	0	Unsupported Opcode
1	1	FASTBUS block transfer terminated due to a word count overflow

T.O.
Indicates that the I/O-Error handler was entered due to a FASTBUS Time-Out.

SS2-SS0
Indicate the SS code at the time the I/O-Error Handler was entered.

BRQ
Indicates that the SSP was requesting the BUS, Crate or Cable.

EG
Internal Enable Geographic, indicates that a Primary Address was in progress.

AS/AK
Indicates that the AS/AK lock was established when the I/O-Error Handler was entered.

MS1-MS0
Indicate the MS code at the time when the I/O-Error Handler was entered.

RDOT
Indicates that the SSP was issuing a RD to the BUS, Crate or Cable.

WTOT
Indicates that the SSP was issuing a WT to the BUS, Crate or Cable.

EXCP2, EXCP1
Indicates the type of exception encountered as described above.

NPA
This bit is used to determine whether or not to subtract 1 from the PSW to resume execution of the MAIN program. If this bit is HIGH, the PSW must be decremented by 1. If this bit is LOW, the PSW is left alone.

SMSK
System Mask bit. If this bit is ON, I/O Interrupts are enabled. If this bit is OFF, the SSP will not acknowledge I/O Interrupts.

OVMSK
Overflow Mask bit. If this bit is ON, an Overflow will generate an Exception. If this bit is OFF, an Overflow will be ignored.

ECC2, ECC1
Encoded condition code bits. Program Counter Bits 11 - 0 indicate the P.C. at the time the I/O-Error Handler was entered. Because of the instruction pipelining in the SSP, this P.C. may point to the instruction that would have been executed had an error not been encountered, or 1 past it. The NPA bit is used to determine whether to subtract one or not.

SSP INSTRUCTION SET

The SSP emulates IBM 370 integer machine language instructions as well as several SSP specific instructions. A further explanation of the IBM Instructions can be found in the IBM 370/360 assembly language manual. The new instructions for the SSP are detailed in this section.

IBM Instructions:
Table 4 lists the IBM instructions and Table 5 lists the SSP specific instructions.

Table 4: IBM Instructions

Op Code	Mnemonic	Name	Type
5	BALR	BRANCH AND LINK	RR
6	BCTR	BRANCH ON COUNT	RR
7	BCR	BRANCH ON CONDITION	RR
1A	AR	ADD	RR
1B	SR	SUBTRACT	RR
1C	MR	MULTIPLY	RR
1D	DR	DIVIDE	RR
1E	ALR	ADD LOGICAL	RR
1F	SLR	SUBTRACT LOGICAL	RR
10	LPR	LOAD POSITIVE	RR
11	LNR	LOAD NEGATIVE	RR
12	LTR	LOAD AND TEST	RR
13	LCR	LOAD COMPLEMENT	RR
14	NR	AND	RR
15	CLR	COMPARE LOGICAL	RR
16	OR	OR	RR
17	XR	EXCLUSIVE OR	RR
18	LR	LOAD	RR
19	CR	COMPARE	RR
4A	AH	ADD HALFWORD	RX
4B	SH	SUBTRACT HALFWORD	RX
4C	MH	MULTIPLY HALFWORD	RX
40	STH	STORE HALFWORD	RX
41	LA	LOAD ADDRESS	RX
42	STC	STORE CHARACTER	RX
43	IC	INSERT CHARACTER	RX
45	BAL	BRANCH AND LINK	RX
46	BCT	BRANCH ON COUNT	RX
47	BC	BRANCH ON CONDITION	RX
48	LH	LOAD HALFWORD	RX
49	CH	COMPARE HALFWORD	RX

Table 4: IBM Instructions (continued)

Op Code	Mnemonic	Name	Type
5A	A	ADD	RX
5B	S	SUBTRACT	RX
5C	M	MULTIPLY	RX
5D	D	DIVIDE	RX
5E	AL	ADD LOGICAL	RX
5F	SL	SUBTRACT LOGICAL	RX
50	ST	STORE	RX
54	N	AND	RX
55	CL	COMPARE LOGICAL	RX
56	O	OR	RX
57	X	EXCLUSIVE OR	RX
58	L	LOAD	RX
59	C	COMPARE	RX
8A	SRA	SHIFT RIGHT SINGLE	RS
8B	SLA	SHIFT LEFT SINGLE	RS
8C	SRDL	SHIFT RIGHT DOUBLE LOGICAL	RS
8D	SLDL	SHIFT LEFT DOUBLE LOGICAL	RS
8E	SRDA	SHIFT RIGHT DOUBLE	RS
8F	SLDA	SHIFT LEFT DOUBLE	RS
86	BXH	BRANCH ON INDEX HIGH	RS
87	BXLE	BRANCH ON INDEX LOW OR EQUAL	RS
88	SRL	SHIFT RIGHT SINGLE LOGICAL	RS
89	SLL	SHIFT LEFT SINGLE LOGICAL	RS
90	STM	STORE MULTIPLE	RS
91	TM	TEST UNDER MASK	SI
92	MVI	MOVE IMMEDIATE	SI
94	NI	AND IMMEDIATE	SI
95	CLI	COMPARE LOGICAL IMMEDIATE	SI
96	OI	OR IMMEDIATE	SI
97	XI	EXCLUSIVE OR IMMEDIATE	SI
98	LM	LOAD MULTIPLE	RS

Program Execution

Normally , operation of the CPU is controlled by instructions taken in sequence. This sequence is governed by the program status word (PSW) , which contains the primary information required for proper program execution. A change in the sequential operation may be caused by branching, LOAD PSW, interruptions, or manual intervention.

Instructions
Each instruction consists of two major parts : (1) an operation code , which specifies the operation

to be performed, and (2) the designation of the operands that participate.

Operands
Operands can be grouped in three classes : operands located in registers, immediate operands, and operands in main storage. Operands may be either explicitly or implicitly designated.

Register operands are located in one of the general registers R0 - R15. The register containing the operand is specified by identifying the register in a four-bit field, called the R field, in the instruction. For some instructions an operand is located in an implicitly designated register, the register being implied by the operation code.

Immediate operands are contained within the instruction, and the eight-bit field containing the immediate operand is called the I field.

Operands in main storage may either have an implied length, be specified by a bit mask, or, in other cases, be specified by a four-bit or eight-bit length specification, called the L field, in the instruction. The addresses of operands in main storage are specified by means of a format that uses the contents of a general register as part of the address. This makes it possible to:

- Specify a complete address by using an abbreviated notation.
- Perform address manipulation using instructions which employ general registers for operands
- Modify addresses by program means without alteration of the instruction stream.
- Operate independently of the location of data areas by directly using addresses received from other programs.

The address used to refer to main storage either is contained in a register designated by the **R** field in the instruction or is calculated from a base address, index, and displacement, designated by the **B**, **X**, and **D** fields. respectively, in the instruction.

For purposes of describing the execution of instructions, operands are designated as first and second operands and. in some cases, third operands.

In general, two operands participate in an instruction execution, and the result replaces the first operand. An exception is instructions with “store” in the name, where the result replaces the second operand. Except for storing the final result, the contents of all registers and storage locations participating in the addressing or execution part of an operation remain unchanged.

Instruction Format
An instruction is one, two, or three halfwords in length and must be located in main storage on an integral halfword boundary. Each instruction is in one of five basic formats: RR, RX, RS, SI, and S.

Some instructions contain fields that vary slightly from the basic format, and in some instructions the operation performed does not follow the general rules stated in this section. All such exceptions are explicitly identified in the individual instruction descriptions.

The format names express, in general terms, the classes of operands which participate in the operation: **RR** denotes a register-to-register operation; **RX**. a register-and-indexed-storage operation; **RS**, a register-and-storage operation; **SI**, a storage-and- immediate operation. The **S** format denotes an operation using all implied operand and storage.

The first byte and, in the S format, the first two bytes of an instruction contain the operation code (op code). For some instructions in the S format, all or a portion of the second byte is ignored. The first two bits of the operation code specify the length and format of an instruction, as follows:

Bit Positions 0-1	Instruction Length	Instruction Format
00	One halfword	RR
01	Two halfwords	RX
10	Two halfwords	RS/SI/S/RX

In the format illustration for each individual instruction description. the opcode field shows the opcode in hexadecimal representation. The hexadecimal representation uses one graphic for a four-bit code, and therefore two graphics for an eight-bit code. The graphics 0-9 are used for the codes 0000 - 1001 ; the graphics A-F are used for codes 1010 - 1111.

The remaining fields in the format illustration for each instruction are designated by code names, consisting of a letter and possibly a subscript number. The subscript number denotes the operand to which the field applies.

Register Operation
In the RR, RX, and RS formats, the contents of the register designated by the R1 field are called the first operand. In the RR format, the R2 field designates the register containing the second operand, and the same register may be designated for the first and second operand. In the RS format, the use of the Rx field depends on the instruction.

The R field designates a general register in the general instructions.

Unless otherwise indicated in the individual instruction description, the register operand is one register in length (32 bits for a general register), and the second operand is the same length as the first.

Immediate Operands
In the SI format, the contents of the eight-bit immediate-data field, the I2 field of the instruction, are used directly as the second operand. The B1 and D1 fields designate the first operand, which is one byte in length.

Storage Operands
In the SI format, the contents of the general register designated by the B1 field are added to the contents of the D1 field to form the first-operand address. In the S and RS format, the contents of the general register designated by the B2 field are added to the contents of the D2 field to form the second-operand address. In the RX format, the contents of the general registers designated by the X2 and B2 fields are added to the contents of the D2 field to form the second-operand address.

Op Code	R_1	X_2	B_2	D_2	
31	23	19	15	11	0

RX Format

Op Code	R_1	R_2		
31	23	19	15	0

RR Format

Op Code	R_1	R_3	B_2	D_2	
31	23	19	15	11	0

RS Format

Op Code	I_2	B_1	D_1	
31	19	15	11	0

SI Format

Op Code		B_2	D_2	
31	23	15	11	0

S Format

Five Basic Instruction Formats

Address Generation

The address used to refer to main storage either is contained in a register designated by the R field in the instruction or is calculated from the following three binary numbers:

Base Address is a 24-bit number contained in a general register specified by the program in a four-bit field, called the B field, in the instruction. Base addresses can be used as a means of independently addressing each program and data area. in array type calculations, it can specify the location of an array, and, in record-type processing, it can identify the record. The base address provides for addressing the entire main storage. The base address may also be used for indexing purposes.

Index is a 24-bit number contained in a general register designated by the program in a four-bit field, called the X field, in the instruction. It is included only in the address specified by the RX instruction format. The RX format instructions permit double indexing; that is, the index can be used to provide the address of an element within an array.

Displacement is a 12-bit number contained in a field, called the D field, in the instruction. The displacement provides for relative addressing of up to 4,095 bytes beyond the location designated by the base address. in array-type calculations, the displacement can be used to specify one of many items associated with an element. In the processing of records, the displacement can be used to identify items within a record.

In forming the address, the base address and index are treated as unsigned 24-bit positive binary integers. The displacement is similarly treated as a 12-bit positive binary integer, and 12 high-order zeros are appended. The three are added as 24-bit binary numbers, ignoring overflow. The sum is always 24 bits long. The bits of the generated address are numbered 8-31 , corresponding to the numbering of the base-address and index bits in the general register.

A zero in any of the X2, B1, or B2 fields indicates the absence of the corresponding address component. For the absent component, a zero is used in forming the address, regardless of the contents of General register 0. A displacement of zero has no special significance.

An instruction can designate the same general register both for address computation and as the location of an operand. Address computation is completed prior to the execution of the operation.

Unless otherwise indicated in the individual instruction definition, the computed operand address designates an operand in main storage. When a main-storage operand is designated, the address designates the leftmost byte of the operand. For branching instructions, the second-operand address is used as the branch address. For shifting instructions, the second-operand address is not used as an address but specifies the shift amount.

Program Status Word

The program status word (PSW) is 32 bits in length and contains the information required for proper program execution. The PSW includes the instruction address, condition code, and other fields. In general, the PSW is used to control instruction sequencing and to hold and indicate the status of the system in relation to the program currently being executed. The active or controlling PSW is called the current PSW. By storing the current PSW during an interruption, the status of the CPU can be preserved for subsequent inspection. By loading a new PSW or part of a PSW, the state of the CPU can be initialized or changed.

Instruction Execution

In program execution, the instruction is fetched from the location designated by the instruction address in the current PSW. The instruction address is then increased by one in order to address the next instruction in sequence. The instruction is then executed, and the same steps are repeated using the new value of the instruction address.

Branching

The normal sequential execution of instructions may be changed by the use of the branching instructions in order to perform subroutine linkage , decision making, and loop control.

Subroutine linkage is provided by the BRANCH AND LINK instructions, which permit not only the introduction of a new instruction address but also the preservation of the return address and associated information.

Facilities for decision making are provided by the BRANCH ON CONDITION instruction. This instruction inspects a two-bit condition code that reflects the result of a majority of the arithmetic, logical, and I/0 operations. Each of these operations can set the code in any one of four states, and the instruction BRANCH ON CONDITION can specify any selection of these four states as the criterion for branching. For example, the condition code reflects such conditions as nonzero, first operand high, equal, overflow, and zero. Once set, the condition code remains unchanged until modified by an instruction that causes a different condition code to be set.

The two bits of the condition code provide for four possible condition code settings: 0, 1 , 2, and 3. The specific meaning of any setting depends on the operation that sets the condition code.

Loop control can be performed by the use of BRANCH ON CONDITION to test the outcome of address arithmetic and counting operations. For some particularly frequent combinations of arithmetic and tests, the instructions BRANCH ON count and BRANCH ON INDEX are provided. These branches, being specialized, provide increased performance for these tasks.

The interruption system permits the CPU to change state as a result of conditions external to the system, in input/output (I/0) units, or in the CPU itself. Four classes of interruption conditions are possible : machine check (divide by zero), program, external (FASTBUS access), and FASTBUS error.

Data Format

The general instructions treat data as being of three types: signed fixed-point numbers, unsigned fixed-point numbers, and unstructured logical quantities.

Data resides in general registers or in storage or is introduced from the instruction stream.

Number Representation

Fixed-point numbers are treated as signed or unsigned integers.

In an unsigned fixed-point number, all bits are used to express the absolute value of the number. When two unsigned fixed-point numbers are added, the shorter number is considered to be extended with high-order zeros.

For signed fixed-point numbers, the leftmost bit represents the sign, which is followed by the integer field. Positive numbers are represented in true binary notation with the sign bit set to zero. Negative numbers are represented in two's-complement binary notation with a one in the sign-bit position.

Specifically, a negative number is represented by the two's complement of the positive number. The two's complement of a number is obtained by inverting each bit of the number and adding a one in the low-order bit position.

This type of number representation can be considered the low-order portion of an infinitely long representation of the number. When the number is positive, all bits to the left of the most significant bit of the number are zeros. the number is negative, all these bits are ones. Therefore, when an operand must be extended with high-order bits, the expansion is achieved by setting the bits equal to the high-order bit of the operand.

The notation for signed fixed-point numbers does not include a negative zero. It has a number range in which the set of negative numbers is one larger than the set of positive numbers. The maximum positive number consists of an all-one integer field with a sign bit of zero, whereas the maximum negative number (the negative number with the greatest absolute value) consists of an all-zero integer field with a sign bit of one.

The complement of the maximum negative number cannot be represented in the same number of bits. when an operation, such as a subtraction of the maximum negative number from zero, attempts to produce the complement of the maximum negative number, a fixed-point overflow exception is recognized. An overflow does not result, however, when the maximum negative number is complemented and the final result is within the representable range. An example of this case is a subtraction of the maximum negative number from minus one. The product of two maximum negative numbers is representable as a double-length positive number.

In discussions of signed fixed-point numbers in this publication, the expression "32-bit signed integer" denotes a 31-bit integer with a sign bit, and the expression "64-bit signed integer" denotes a 63-bit integer with a sign bit.

In some operations, the result is achieved by the use of the one's complement of the number. The one's complement of a number is obtained by inverting each bit of the number.

In an arithmetic operation, a carry out of the integer field changes the sign. However, in algebraic left-shifting the sign bit does not change even if significant high-order bits are shifted out.

Programming Note
The integer part of a signed fixed-point number may be considered to represent a positive value , with the sign representing a value of either zero or the maximum negative number,

Instructions
The general instructions and their mnemonics, formats, and operation codes are listed in the following table. The table also indicates when the condition code is set and the exceptional conditions in operand designations, data, or results that cause a program interruption. Note: in the detailed descriptions of the individual instructions , the mnemonic and the symbolic operand designations for the IBM System/370 assembly language are shown with each instruction. For LOAD AND TEST, for example, LTR is the mnemonic and RI, R2 the operand designation.

Add

AR **R_1,R_2** **[RR]**

1A	R_1	R_2	
31	23	19	15 0

A **R_1,D_2(X_2,B_2)** **[RX]**

5A	R_1	X_2	B_2	D_2
31	23	19	15	11 0

The second operand is added to the first operand, and the sum is placed in the first-operand location.

Addition is performed by adding all 32 bits of both operands. If the carry out of the sign-bit position and the carry out of the high-order numeric bit position agree, the sum is satisfactory; if they disagree, an overflow occurs. The sign bit is not changed after the overflow. A positive overflow yields a negative final sum, and a negative overflow results in ; positive sum. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:
0 Sum is zero
1 Sum is less than zero
2 Sum is greater than zero
3 Overflow

Program Exceptions:
Access (fetch, operand 2 of A only)
Fixed-Point Overflow

Programming Note:
In two's-complement notation a zero result is always positive.

Relevant Extended Mnemonic Instructions:

Mnemonic	Meaning	Op Code
BO or BOR	Branch on Overflow	BC or BCR 1
BP or BPR	Branch on Plus	BC or BCR 2
BM or BMR	Branch on Minus	BC or BCR 4
BNP or BNPR	Branch on Not Plus	BC or BCR 13
BNM or BNMR	Branch on Not Minus	BC or BCR 11
BNZ or BNZR	Branch on Not Zero	BC or BCR 7
BZ or BZR	Branch on Zero	BC or BCR 8

Add Halfword

AH **R_1,D_2(X_2,B_2)** **[RX]**

4A	R_1	X_2	B_2	D_2	
31	23	19	15	11	0

The second operand is added to the first operand, and the sum is placed in the first-operand location, The second operand is two bytes in length and is considered to be a 16-bit signed integer.

The second operand is expanded to 32 bits before the addition by propagating the sign-bit value through the 16 high-order bit positions. The contents of the second operand in main storage remain unchanged.

Addition is performed by adding all 32 bits of both operands. If the carry out of the sign-bit position and the carry out of the high-order numeric bit position agree, the sum is satisfactory; if they disagree, an overflow occurs. The sign bit is not changed after the overflow. A positive overflow yields a negative final sum, and a negative overflow results in a positive sum. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:
0 Sum is zero
1 Sum is less than zero
2 Sum is greater than zero
3 Overflow

Program Exceptions:
Access (fetch, operand 2)
Fixed-Point Overflow

Programming Note
See Extended Mnemonic Instructions on page 24

Add Logical

ALR **R_1,R_2** **[RR]**

1E	R_1	R_2	
31	23	19	15 … 0

AL **R_1,D_2(X_2,B_2)** **[RX]**

5E	R_1	X_2	B_2	D_2
31	23	19	15	11 … 0

The second operand is added to the first operand, and the sum is placed in the first-operand location. The occurrence of a carry out of the sign position is recorded in the condition code.

Logical addition is performed by adding all 32 bits of both operands without further change to the resulting sign bit. The instruction differs from ADD in the meaning of the condition code and in the absence of the interruption for overflow.

If a carry out of the sign position occurs, the leftmost bit of the condition code is made one. In the absence of a carry, the bit is made zero. When the sum is zero, the rightmost bit of the condition code is made zero. For a nonzero sum, the bit is made one.

Resulting Condition Codes
0 Sum is zero, with no carry
1 Sum is not zero, with no carry
2 Sum is zero, with carry
3 Sum is not zero, with carry

Program Exceptions:
Access (fetch, operand 2 of AL only)

And

NR **R_1,R_2** **[RR]**

14	R_1	R_2	
31	23	19	15 … 0

N **R_1,D_2(X_2,B_2)** **[RX]**

54	R_1	X_2	B_2	D_2
31	23	19	15	11 … 0

NI **D_1(B_1),I_2** **[SI]**

12	I_2	B_1	D_1
31	19	15	11 … 0

The AND of the first and second operands is placed in the first-operand location.

Operands are treated as unstructured logical quantities, and the connective AND is applied bit by bit. A bit position in the result is set to one if the corresponding bit positions in both operands contain a one; otherwise, the result bit is set to zero.

Resulting Condition Code:
0 Result is zero
1 Result not zero
2 -
3 -

Program Exceptions:
Access (fetch, operand 2, N)
fetch and store, operand 1, NI)

Programming Note
The instruction AND may be used to set a bit to zero.

The immediate byte's position in the operand is reversed. Given a word-aligned address D, bytes will be accessed as follows:

Address	Bits
D	D(31:24)
D+1	D(23:16)
D+2	D(15:8)
D+3	D(7:0)

Branch and link

BALR $\mathbf{R_1,R_2}$ **[RR]**

05	R_1	R_2	
31	23	19	15 … 0

BAL $\mathbf{R_1,D_2(X_2,B_2)}$ **[RX]**

45	R_1	X_2	B_2	D_2
31	23	19	15	11 … 0

Information from the current PSW, including the updated instruction address, is loaded as link information in the general register designated by R1. Subsequently, the instruction address is replaced by the branch address.

In the RX format, the second-operand address is used as the branch address. In the RR format, the contents of bit positions 23-0 of the general register designated by R2 are used as the branch address. In the IBM 370, when the R2 field contains zeros, the operation is performed without branching. In the SSP, the assembler translates such an instruction to a BAL0 or BALR0 opcode.

The branch address is computed before the link information is loaded. The link information, in both the BC and EC modes, consists of the condition code , the program mask bits, and the updated instruction address, arranged in the following format:

Resulting Condition Code:
The code remains unchanged.

Program Exceptions:
None

Branch on Condition

BCR $\mathbf{M_1,R_2}$ **[RR]**

07	M_1	R_2	
31	23	19	15 … 0

BC $\mathbf{M_1,D_2(X_2,B_2)}$ **[RX]**

47	M_1	X_2	B_2	D_2
31	23	19	15	11 … 0

The updated instruction address in the current PSW is replaced by the branch address if the state of the condition code is as specified by M1; otherwise, normal instruction sequencing proceeds with the updated instruction address.

In the RX format the second-operand address is used as the branch address. In the RR format the contents of bit positions 23-0 of the general register specified by R2 are used as the branch address. However , when the R2 field contains zeros, the operation is performed without branching.

The M1 field is used as a four-bit mask. The four bits of the mask correspond, left to right, with the four condition codes (0, 1, 2, and 3), as follows:

Instruction Bit	Mask Position Value	Condition Code
8	8	0
9	4	1
10	2	2
11	1	3

The branch is successful whenever the condition code has a corresponding mask bit of one.

When the M1 and R2 fields of BCR are 15 and 0, respectively, a serialization function is performed. CPU operation is delayed until all previous storage accesses by this CPU to main storage have been completed, as observed by channels and other CPUs. No subsequent instructions or their operands are accessed by this CPU until the execution of this instruction is completed.

Resulting Condition Code:
The code remains unchanged.

Program Exceptions:
None

Programming Notes
When a branch is to be made on more than one condition code, the pertinent condition codes are specified in the mask as the stun of their mask position values. A mask of 12, for example, specifies that a branch is to be made on condition codes 0 and 1.

When all four mask bits are zero or when the R2 field in the RR format contains zero, the branch instruction is equivalent to a no-operation. When all four mask bits are ones, that is, the mask value is 15, the branch is unconditional unless the R_2 field in the RX format is zero.

Note that the relation between the RR and RX formats in branch-address specification is not the same as in operand-address specification. For branch instructions in the RX format, the branch address is the address specified by X2, B2, and D2; in the RR format, the branch address is in the low-order 24 bits of the register specified by R2. For operands, the address specified by X2, B2, and D2 is the operand address, but the register specified by R2 contains the operand itself.

Branch on Count

BCTR **R_1,R_2** **[RR]**

06	R_1	R_2	
31	23	19	15 … 0

BCT **R_1,D_2(X_2,B_2)** **[RX]**

46	R_1	X_2	B_2	D_2
31	23	19	15	11 … 0

The contents of the general register specified by R1 are algebraically reduced by one. When the result is zero, normal instruction sequencing proceeds with the updated instruction address. When the result is not zero, the instruction address in the current PSW is replaced by the branch address.

In the RX format, the second-operand address is used as the branch address. In the RR format, the contents of bit positions 23-0 of the general register specified by R2 are used as the branch address. However, when the R2 field contains zeros, the operation is performed without branching.

The branch address is computed before the counting operation. Counting does not change the condition code. The overflow occurring on transition from the maximum negative number to the maximum positive number is ignored. Otherwise, the subtraction proceeds as in fixed-point arithmetic, and all 32 bits of the general register participate in the operation.

Resulting Condition Code:
The code remains unchanged.

Program Exceptions
None

Programming Notes
An initial count of one results in zero, and no branching takes place; an initial count of zero results in minus one and causes branching to be executed; an initial count of minus one results in minus 2 and causes branching to be executed; and so on. In a loop, branching takes place each time the instruction

is executed until the result is again zero. Note that, because of the number range, an initial count of minus 231 results in the positive value of 211.

Counting is performed without branching when the R2 field in the RR format contains zero.

Branch on Index High

BXH **R_1,R_3,D_2(B_2)** [**RS**]

86	R_1	R_3	B_2	D_2
31	23	19	15	11 0

An increment is added to the first operand, and the sum is compared algebraically with a comparand. Subsequently, the sum is placed in the first-operand location, regardless of whether the branch is taken. The second-operand address is used as the branch address. When the sum is high, the instruction address in the current PSW is replaced by the branch address. When the sum is low or equal, instruction sequencing proceeds with the updated instruction address.

The first operand and the increment are in the registers specified by R1 and R3. The comparand register address is odd and is either one larger than R3 or equal to R3- The branch address is computed before the addition and comparison.

Overflow caused by the addition is ignored and does not affect the comparison. Otherwise, the addition and comparison proceed as in fixed-point arithmetic. All 32 bits of the general registers participate in the operations, and negative quantities are expressed in two's-complement notation. When the first operand and comparand locations coincide , the original register contents are used as the comparand.

Resulting Condition Code:
The code remains unchanged.

Program Exceptions:
None

Programming Note
The name "branch on index high" indicates that one of the major purposes of this instruction is the incrementing and testing of an index value. The increment may be algebraic and of any magnitude.

Branch on Index Low or Equal

BXLE **R_1,R_3,D_2(B_2)** [**RS**]

87	R_1	R_3	B_2	D_2
31	23	19	15	11 0

An increment is added to the first operand, and the sum is compared algebraically with a comparand.

Subsequently, the sum is placed in the first-operand location, regardless of whether the branch is taken. The second-operand address is used as the branch address.

When the sum is low or equal, the instruction address in the current PSW is replaced by the branch address. When the sum is high, normal instruction sequencing proceeds with the updated instruction address.

The first operand and the increment are in the registers specified by R1 and R3. The comparand register address is odd and is either one larger than R3 or equal to Rx. The branch address is computed before the addition and comparison.

This instruction is similar to BRANCH ON INDEX HIGH, except that the branch is successful when the sum is low or equal compared to the comparand.

Resulting Condition Code:
The code remains unchanged.

Program Exceptions:
None

Compare

CR **R_1,R_2** **[RR]**

19	R_1	R_2	
31	23	19	15 … 0

C **R_1,R_2** **[RR]**

59	R_1	R_2	
31	23	19	15 … 0

The first operand is compared with the second operand, and the result determines the setting of the condition code.

Comparison is algebraic, treating both comparands as 32-bit signed integers. Operands in registers or storage are not changed.

Resulting Condition Code:
0 Operands are equal
1 First operand is low
2 First operand is high
3 -

Program Exceptions:
Access (fetch, operand 2 of C only)

The first operand is compared with the second operand, and the result determines the setting of the condition code.

Comparison is algebraic, treating both comparands as 32-bit signed integers. Operands in registers or storage are not changed.

Resulting Condition Code:
0 Operands are equal
1 First operand is low
2 First operand is high
3 -

Program Exceptions:
Access (fetch, operand 2 of C only)

Programming Note

Relevant Extended Mnemonic Instructions:

Mnemonic	Meaning	Op Code
BH or BHR	Branch on A High	BC or BCR 2
BL or BLR	Branch on A Low	BC or BCR 4
BE or BER	Branch on A Equal B	BC or BCR 8
BNH or BNHR	Branch on A Not High	BC or BCR 13
BNL or BNLR	Branch on A Not Low	BC or BCR 11
BNE or BNER	Branch on A Not Equal B	BC or BCR 7

Compare Halfword

CH **R_1,D_2(X_2,B_2)** **[RX]**

49	R_1	X_2	B_2	D_2
31	23	19	15	11 … 0

The first operand is compared with the second operand, and the result determines the setting of the condition code. The second operand is two bytes in length and is considered to be a 16-bit signed integer.

The second operand is expanded to 32 bits before the comparison by propagating the sign-bit value through the 16 high-order bit positions.

Comparison is algebraic, treating both comparands as 32-bit signed integers. Operands in registers or storage are not changed.

Resulting Condition Code:
0 Operands are equal
1 First operand is low

2 First operand is high
3 -

Program Exceptions:
Access (fetch, operand 2)

Programming Note
See Extended Mnemonic Instructions on page 32

Compare Logical

CLR **R_1,R_2** **[RR]**

15	R_1	R_2	
31	23	19	15 … 0

CL **R_1,D_2(X_2,B_2)** **[RX]**

55	R_1	X_2	B_2	D_2
31	23	19	15	11 … 0

CLI **D_1(B_1),I_2** **[SI]**

31	I_2	B_1	D_1
31		19	15	11 … 0

The first operand is compared with the second operand, and the result is indicated in the condition code.

The comparison is performed with the operands considered as binary unsigned quantities, with all codes valid. The operation proceeds left to right and ends as soon as an inequality is found or an end of the fields is reached.

When part of an operand in CL or CLC is designated in an inaccessible location but the operation can be completed by using the accessible operand parts, it is unpredictable whether the access exception for the inaccessible part is recognized.

Resulting Condition Code:
0 Operands are equal
1 First operand is low
2 First operand is high
3 -

Program Exceptions:
Access (fetch, operand 2, CL and CLC; fetch, operand 1, CLI and CLC)

Programming Note
The COMPARE LOGICAL instruction treats all bits alike as part of an unsigned binary quantity.

The immediate byte's position in the operand is reversed. Given a word-aligned address D, bytes will be accessed as follows:

Address	Bits
D	D(31:24)
D+1	D(23:16)
D+2	D(15:8)
D+3	D(7:0)

See Extended Mnemonic Instructions on page 32

Divide

DR **R_1,R_2** **[RR]**

1D	R_1	R_2	//////////
31	23	19	15 … 0

D **R_1,D_2(X_2,B_2)** **[RX]**

5D	R_1	X_2	B_2	D_2
31	23	19	15	11 … 0

The dividend (first operand) is divided by the divisor (second operand) and replaced by the remainder and the quotient.

The dividend is a 64-bit signed integer and occupies the even-odd pair of registers specified by the R1 field of the instruction. A specification exception occurs when R1 is odd. A 32-bit signed remainder and a 32-bit signed quotient replace the dividend in the even-numbered and odd-numbered registers, respectively. The divisor is a 32-bit signed integer.

The sign of the quotient is determined by the rules of algebra. The remainder has the same six as the dividend, except that a zero quotient or a zero remainder is always positive. When the relative magnitude of dividend and divisor is such that the quotient cannot be expressed by a 32-bit signed integer, a fixed-point divide exception is recognized (a program interruption occurs, no division takes place, and the dividend remains unchanged in the general registers).

Condition Code:
The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2 of D only) Specification
Fixed-Point Divide

exclusive OR

XR **R_1,R_2** **[RR]**

17	R_1	R_2	
31	23	19	15 … 0

X **R_1,D_2(X_2,B_2)** **[RX]**

57	R_1	X_2	B_2	D_2
31	23	19	15	11 … 0

XI **D_1(B_1),I_2** **[SI]**

97	I_2	B_1	D_1
31	19	15	11 … 0

The EXCLUSIVE OR of the first and second operands is placed in the first-operand location.

Operands are treated as unstructured logical quantities, and the connective EXCLUSIVE OR is applied bit by bit. A bit position in the result is set to one if the corresponding bit positions in the two operands are unlike ; otherwise , the result bit is set to zero.

Resulting Condition Code:
0 Result is zero
1 Result not zero
2 -
3 -

Program Exceptions:
Access (fetch, operand 2, X ; fetch and store, operand 1, XI) Programming Note
The instruction EXCLUSIVE OR may be used to invert a bit, an operation particularly useful in testing and setting programmed binary bit switches.

A field EXCLUSIVE-ORed with itself becomes all zeros.

The sequence A EXCLUSIVE-ORed B, B EXCLUSIVE-ORed A, A EXCLUSIVE-ORed B results in the exchange of the contents of A and B without the use of an auxiliary buffer area.

The immediate byte's position in the operand is reversed. Given a word-aligned address D, bytes will be accessed as follows:

Address	Bits
D	D(31:24)
D+1	D(23:16)
D+2	D(15:8)
D+3	D(7:0)

Insert Character

IC $\quad R_1,D_2(X_2,B_2)$ $\quad$ [RX]

43	R_1	X_2	B_2	D_2
31	23	19	15	11 $\quad$ 0

The byte at the second-operand location is inserted into bit positions 11-0 of the general register designated by the R1 field. The remaining bits in the register remain unchanged.

Condition Code:
The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2)

Load

LR $\quad R_1,R_2$ $\quad$ [RR]

18	R_1	R_2	
31	23	19	15 $\quad$ 0

L $\quad R_1,D_2(X_2,B_2)$ $\quad$ [RX]

58	R_1	X_2	B_2	D_2
31	23	19	15	11 $\quad$ 0

The second operand is placed unchanged in the first operand location.

Condition Code:
The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2 of L only)

Load Address

LA $\mathbf{R_1,D_2(X_2,B_2)}$ **[RX]**

41	R_1	X_2	B_2	D_2
31	23	19	15	11 … 0

The address specified by the X2, B2, and D2 fields is inserted in bit positions 23-0 of the general register specified by the R1 field. Bits 24-31 of the register are set to zeros. The address computation follows the rules for address arithmetic.

No storage references for operands take place, and the address is not inspected for access exception

Resulting Condition Code:
The code remains unchanged.

Program Exceptions:
None.

programming Note
The same general register may be specified by the RI, X2, and B2 instruction field, except that general register 0 can be specified only by the R1 field. ln this manner it is possible to increment the low-order 24 bits of a general register, other than 0, by the contents of the D2 field of the instruction. The register to be incremented should be specified by R1 and by either X2 (with B2 set to zero) or B2 (with X2 set to zero).

Load and test

LTR $\mathbf{R_1,R_2}$ **[RR]**

12	R_1	R_2	//////////
31	23	19	15 … 0

The second operand is placed unchanged in the first operand location, and the sign and magnitude of the second operand determine the condition code.

Condition Code:
0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 -

Program Exceptions:
None.

Programming Note
When the R1 and R2 fields designate the same register, the operation is equivalent to a test without

data movement.

Load Complement

LCR **R_1,R_2** **[RR]**

13	R_1	R_2	
31	23	19	15 ... 0

The two's complement of the second operand is placed in the first-operand location.

An overflow condition occurs when the maximum negative number is complemented; the number remains unchanged. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:
0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Exceptions:
Fixed-Point Overflow

Programming Note
Zero remains unchanged by complementation.

See Extended Mnemonic Instructions on page 24

Load Halfword

LH **R_1,D_2(X_2,B_2)** **[RX]**

48	R_1	X_2	B_2	D_2
31	23	19	15	11 ... 0

The second operand is placed in the first-operand location. The second operand is two bytes in length and is considered to be a 16-bit signed integer.

The second operand is expanded to 32 bits by propagating the sign-bit value through the 16 high order bit positions. Expansion occurs after the operand is obtained from storage and before insertion in the register.

Condition Code:
The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2)

Programming Note
The second operand is a 16-bit integer. Given a word-aligned address D, the instruction LH R1,D will move bits 31:16 of D to bits 15:0 of R1.

Load Multiple

LM **R_1,R_3,D_2(B_2)** **[RS]**

98	R_1	R_3	B_2	D_2
31	23	19	15	11 … 0

The set of general registers starting with the register specified by R1 and ending with the register specified by R3 is loaded from the locations designated by the second-operand address.

The storage area from which the contents of the general registers are obtained starts at the location designated by the second-operand address and continues through as many locations as needed. The general registers are loaded in the ascending order of their addresses, starting with the register specified by R1 and continuing up to and including the register specified by R3, with register 0 following register 15.

Condition Code:
The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2)

Programming Note
All combinations of register addresses specified by R1 and R3 are valid. When the register addresses are equal, only one word is transmitted. When the address specified by R3 is less than the address specified by R1, the register addresses wrap around from 15 to 0.

Load Negative

LNR **R_1,R_2** **[RR]**

11	R_1	R_2	
31	23	19	15 … 0

The two's complement of the absolute value of the second operand is placed in the first-operand location.

The operation complements positive numbers; negative numbers remain unchanged. The number zero remains unchanged with positive sign.

Condition Code:
0 Result is zero
1 Result is less than zero
2 -
3 -

Program Exceptions:
None.

Load Positive

LPR $\mathbf{R_1,R_2}$ **[RR]**

xx	R_1	R_2	
31	23	19	15 0

The absolute value of the second operand is placed in the first-operand location. The operation includes complementation of negative numbers; positive numbers remain unchanged. An overflow condition occurs when the maximum negative number is complemented; the number remains unchanged. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:
0 Result is zero
1 -
2 Result is greater than zero
3 Overflow

Program Exceptions
Fixed-Point Overflow

Move

MVI $\mathbf{D_1(B_1),I_2}$ **[SI]**

92	I_2	B_1	D_1
31	19	15	11 0

I The second operand is placed in the first-operand location.

Condition Code:
The code remains unchanged.

Program Exceptions:
Access (store, operand 1, MVI)

Programming Note
The immediate byte's position in the operand is reversed. Given a word-aligned address D, bytes will be accessed as follows:

Address	Bits
D	D(31:24)
D+1	D(23:16)
D+2	D(15:8)
D+3	D(7:0)

Multiply

MR $\mathbf{R_1,R_2}$ **[RR]**

1C	R_1	R_2	
31	23	19	15 0

M $\mathbf{R_1,D_2(X_2,B_2)}$ **[RX]**

5C	R_1	X_2	B_2	D_2
31	23	19	15	11 0

The product of the multiplier (the second operand) and the multiplicand (the first operand) replaces the multiplicand.

Both multiplier and multiplicand are 32-bit signed integers. The product is always a 64-bit signed integer and occupies an even-odd register pair. Because the multiplicand is replaced by the product, the R1 field of the instruction must refer to an even- numbered register. A specification exception occurs when R1 is odd. The multiplicand is taken from the odd register of the pair. The contents of the even numbered register replaced by the product are ignored, unless the register contains the multiplier. An overflow cannot occur.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand sign, except that a zero result is always positive.

Condition Code:
The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2 of M only)
Specification

Programming Note

The significant part of the product usually occupies 62 bits or fewer. Only when two maximum negative numbers are multiplied are 63 significant product bits formed. Since two's-complement notation is used, the sign bit is extended right until the first significant product digit is encountered.

Multiply Halfword

MH $\mathbf{R_1,D_2(X_2,B_2)}$ **[RX]**

4C	R_1	X_2	B_2	D_2
31	23	19	15	11 0

The product of the multiplier (second operand) and multiplicand (first operand) replaces the multiplicand. The second operand is two bytes in length and is considered to be a 16-bit signed integer.

Both multiplicand and product are 32-bit signed integers and may be located in any general register. The 16-bit multiplier is expanded to 32 bits before multiplication by propagating the six-bit value through the 16 high-order bit positions. The multiplicand is replaced by the low-order part of the product. The bits to the left of the 32 low-order bits are not tested for significance; no overflow indication is given.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand sign, except that a zero result is always positive.

Resulting Condition Code:
The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2)

Programming Note
The significant part of the product usually occupies 46 bits or fewer, the exception being 47 bits when both operands are maximum negative. Since the low-order 32 bits of the product are stored unchanged, ignoring all bits to the left, the sign bit of the result may differ from the true sign of the product in the case of overflow.

OR

OR $\mathbf{R_1,R_2}$ **[RR]**

16	R_1	R_2	
31	23	19	15 0

O **R_1,D_2(X_2,B_2)** **[RX]**

56	R_1	X_2	B_2	D_2	
31	23	19	15	11	0

OI **D_1(B_1),I_2** **[SI]**

96	I_2	B_1	D_1	
31	19	15	11	0

The OR of the first and second operands is placed in the first-operand location.

Operands are treated as unstructured logical quantities, and the connective OR is applied bit by bit. A bit position in the result is set to one if the corresponding bit position in one or both operands contains a one; otherwise, the result bit is set to zero.

Resulting Condition Code:
0 Result is zero
1 Result not zero
2 -
3 -

Program Exceptions:
Access (fetch, operand 2, 0; fetch and store, operand 1, 0I)

Programming Note
The instruction OR may be used to set a bit to one.

The immediate byte's position in the operand is reversed. Given a word-aligned address D, bytes will be accessed as follows:

Address	Bits
D	D(31:24)
D+1	D(23:16)
D+2	D(15:8)
D+3	D(7:0)

Set program Mask

SPM **R_1,R_2** **[RR]**

04	R_1	R_2		
31	23	19	15	0

Bits 12-14 of the general register specified by the R1 field replace the condition code and the program mask bits of the current PSW. Other bits of the instruction are ignored.

The contents of the register specified by the. R1 field remain unchanged.

Resulting Condition Code:
The code is set according to bits 12 and 13 of the register specified by R1.

Program Exceptions:
None

Programming Note
Bits 14-15 of the general register may have been loaded from the PSW by BRANCH AND LINK.

Shift Left Double

SLDA **R_1,D_2(B_2)** **[RS]**

8F	R_1		B_2	D_2
31	23	19	15	11 0

The double-length integer part of the first operand is shifted left the number of bits specified by the second-operand address. Bits 16-19 of the instruction are ignored.

The R1 field of the instruction specifies an even-odd pair of registers and must designate an even numbered register. When R1 is odd, a specification exception is recognized.

The second-operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The first operand is treated as a number with 63 integer bits and a sign in the sign position of the even register. The sign remains unchanged. The low-order position of the odd register contains an integer bit, and the contents of the odd register participate in the shift in the same manner as the other integer bits. Zeros are supplied to the vacated positions of the registers.

If a bit unlike the sign bit is shifted out of bit position 1 of the even register, an overflow occurs. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:
0 Result is zero
I Result is less than zero
2 Result is greater than zero
3 Overflow

Program Exceptions:
Specification
Fixed-Point Overflow

programming Notes

The eight shift instructions provide the following three pairs of alternatives: left or right, single or double, and algebraic or logical. The algebraic shifts differ from the logical shifts in that, in the algebraic shifts, overflow is recognized, the condition code is set, and the high-order bit participates as a sign.

The maximum shift amount which can be specified is 63. For algebraic shifts this is sufficient to shift out the entire integer field. Since 64 bits participate in the double-logical shifts, the entire register contents cannot be shifted out.

A zero shift amount in the two algebraic double-shift operations provides a double-length sign and magnitude test. The base register participating in the generation of the second-operand address permits indirect specification of the shift amount. A zero in the B2 field indicates the absence of indirect shift specification.

See Extended Mnemonic Instructions on page 24

Shift Left Double Logical

SLDL **R_1,D_2(B_2)** **[RS]**

8D	R_1		B_2	D_2
31	23	19	15	11 … 0

The double-length first operand is shifted left the number of bits specified by the second-operand address. Bits 16-19 of the instruction are ignored.

The R1 field of the instruction specifies an even-odd pair of registers and must designate an even-numbered register. When R1 is odd, a specification exception is recognized.

The second-operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 64 bits of the first operand participate in the shift. High-order bits are shifted out of the even-numbered register without inspection and are lost. zeroes are supplied to the vacated positions of the registers.

Resulting Condition Code:
The code remains unchanged.

Program Exceptions:
Specification

Shift Left Single

SLA **R_1,D_2(B_2)** **[RS]**

8B	R_1	/	B_2	D_2
31	23	19	15	11 0

The integer part of the first operand is shifted left the number of bits specified by the second-operand address. Bits 16-19 of the instruction are ignored.

The second-operand address is not used to address data ; ix low-ordered six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The sign of the first operand remains unchanged. All 31 integer bits of the operand participate in the left shift. Zeros are supplied to the vacated low-order register positions.

If a bit unlike the sign bit is shifted out of position 1 , an overflow occurs. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:
0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Exceptions:
Fixed-Point Overflow

Programming Note
For numbers with an absolute value of less than 230 a left shift of one bit position is equivalent to multiplying the number by two.

Shift amounts from 31-63 cause the entire integer to be shifted out of the register. When the entire integer field for a positive number has been shifted out, the register contains a value of zero. For a negative number, the register contains a value of -231

See Extended Mnemonic Instructions on page 24

Shift Left Single Logical

SLL $\mathbf{R_1,D_2(B_2)}$ **[RS]**

89	R_1	/	B_2	D_2

31 23 19 15 11 0

The first operand is shifted left the number of bits specified by the second-operand address. Bits 16-19 of the instruction are ignored.

The second-operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 32 bits of the first operand participate in the shift. High-order bits are shifted out without inspection and are lost. Zeros are supplied to the vacated low-order register positions.

Resulting Condition Code:
The code remains unchanged.

Program Exceptions:
None

Shift Right Double

SRDA $\mathbf{R_1,D_2(B_2)}$ **[RS]**

8E	R_1	/	B_2	D_2

31 23 19 15 11 0

The double-length integer part of the first operand is shifted right the number of places specified by the second-operand address. Bits 16-19 of the instruction are ignored.

The R1 field of the instruction specifies an even-odd pair of registers and must designate an even numbered register. When R1 is odd, a specification exception is recognized.

The second-operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The first operand is treated as a number with 63 integer bits and a sign in the sign position of the even register. The sign remains unchanged. The high-order position of the odd register contains an integer bit, and the contents of the odd register participate in the shift in the same manner as the other integer bits. The low-order bits are shifted out without inspection and are lost. Bits equal to the sign are supplied to the vacated positions of the registers.

Resulting Condition Code:
0 Result is zero

1 Result is less than zero
2 Result is greater than zero
3 -

Program Exceptions:
Specification

Shift Right Double Logical

SRDL $\mathbf{R_1,D_2(B_2)}$ **[RS]**

8C	R_1	/	B_2	D_2
31	23	19	15	11 0

The double-length first operand is shifted right the number of bits specified by the second-operand address. Bits 16-19 of the instruction are ignored.

The R1 field of the instruction specifies an even-odd pair of registers and must designate an even numbered register. When R1 is odd, a specification exception is recognized.

The second-operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 64 bits of the first operand participate in the shift. Low-order bits are shifted out of the odd-numbered register without inspection and are lost. Zeros are supplied to the vacated positions of the registers.

Resulting Condition Code:
The code remains unchanged.

Program Exceptions:
Specification

Shift Right Single

SRA $\mathbf{R_1,D_2(B_2)}$ **[RS]**

8A	R_1	/	B_2	D_2
31	23	19	15	11 0

The integer part of the first operand is shifted right the number of bits specified by the second-operand address. Bits 16-19 of the instruction are ignored. The second-operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored. The sign of the first operand remains unchanged. All 31 integer bits of the operand participate in the right shift. Bits equal to the sign are supplied to the vacated high-order bit positions. Low-order bits are shifted out without inspection and are lost.

Resulting Condition Code:
0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 -

Program Exceptions:
None

Programming Note
A right shift of one bit position is equivalent to division by two with rounding downward. When an even number is shifted right one position, the value of the field is that obtained by dividing the value by 2. When an odd number is shifted right one position, the value of the field is that obtained by dividing the *next lower* number by two. For example, +5 shifted right by one bit position yields +2, whereas -5 yields -3.

Shift amounts from 31-63 cause the entire integer to be shifted out of the register. When the entire integer field of a positive number has been shifted out, the register contains a value of zero. For a negative number, the register contains a value of -1.

Shift Right Single Logical

SRL $\mathbf{R_1,D_2(B_2)}$ [**RS**]

88	R_1	/	B_2	D_2
31	23	19	15	11 … 0

The first operand is shifted right the number of bits specified by the second-operand address. Bits 16-19 of the instruction are ignored.

The second-operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 32 bits of the first operand participate in the shift. Low-order bits are shifted out without inspection and are lost. Zeros are supplied to the vacated high-order register positions.

Resulting Condition Code:
The code remains unchanged.

Program Exceptions:
None

Store

ST $\mathbf{R_1,D_2(X_2,B_2)}$ **[RX]**

50	R_1	X_2	B_2	D_2
31	23	19	15	11 … 0

The first operand is stored at the second-operand location.

The 32 bits in the general register are placed unchanged at the second-operand location.

Resulting condition Code:
The code remains unchanged.

Program Exceptions:
Access (store, operand 2)

Store Character

STC $\mathbf{R_1,D_2(X_2,B_2)}$ **[RX]**

42	R_1	X_2	B_2	D_2
31	23	19	15	11 … 0

The contents of bit positions 7-0 of the general register designated by the R1 field are placed unchanged at the second-operand location. The second operand is one byte in length.

Condition Code:
The code remains unchanged.

Program Exceptions:
Access (store , operand 2)

Store Halfword

STH $\mathbf{R_1,D_2(X_2,B_2)}$ **[RX]**

40	R_1	X_2	B_2	D_2
31	23	19	15	11 … 0

The contents of bit positions 15-0 of the general register designated by the R1 field are placed unchanged at the second-operand location. The second operand is two bytes in length.

Resulting Condition Code:
The code remains unchanged.

Program Exceptions:

Access (store, operand 2)

Store Multiple

STM $\mathbf{R_1,R_3,D_2(B_2)}$ **[RS]**

90	R_1	R_3	B_2	D_2
31	23	19	15	11 … 0

The set of general registers starting with the register specified by R1 and ending with the register specified by R15 is stored at the locations designated by the second-operand address.

The storage area where the contents of the general registers are placed starts at the location designated by the second-operand address and continues through as many locations as needed. The general registers are stored in the ascending order of their addresses, starting with the register specified by R1 and continuing up to and including the register specified by Rx, with register 0 following register 15.

Resulting Condition Code:
The code remains unchanged.

Program Exceptions:
Access (store, operand 2)

Subtract

SR $\mathbf{R_1,R_2}$ **[RR]**

1B	R_1	R_2	
31	23	19	15 … 0

S $\mathbf{R_1,D_2(X_2,B_2)}$ **[RX]**

5B	R_1	X_2	B_2	D_2
31	23	19	15	11 … 0

The second operand is subtracted from the first operand, and the difference is placed in the first operand location. Subtraction is considered to be performed by adding the one's complement of the second operand and a low-order one to the first operand. All 32 bits of both operands participate, as in ADD. If the carry out of the sign-bit position and the carry out of the high-order numeric bit position agree, the difference is satisfactory; if they disagree, an overflow occurs. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:
0 Difference is zero

1 Difference is less than zero
2 Difference is greater than zero
3 Overflow

Program Exceptions:
Access (fetch, operand 2 of S only)
Fixed-Point Overflow

Programming Note
The use of the one's complement and the low-order one instead of the two's complement of the second operand is necessary for proper recognition of overflow when subtracting the maximum negative number.

When, in the RR format, the R1 and R2 fields designate the same register, subtracting is equivalent to clearing the register.

Subtracting a maximum negative number from another maximum negative number gives a zero result and no overflow.

See Extended Mnemonic Instructions on page 24

Subtract halfword

SH **R_1,D_2(X_2,B_2)** **[RX]**

4B	R_1	X_2	B_2	D_2
31	23	19	15	11 … 0

The second operand is subtracted from the first operand, and the difference is placed in the first-operand location. The second operand is two bytes in length and is considered to be a 16-bit signed integer.

The second operand is expanded to 32 bits before the subtraction by propagating the sign-bit value through the 16 high-order bit positions.

Subtraction is considered to be performed by adding the one's complement of the expanded second operand and a low-order one to the first operand. All 32 bits of both operands participate, as in ADD. If the carry out of the sign-bit position and the carry out of the high-order numeric bit position agree, the difference is satisfactory; ir they disagree, an overflow' occurs. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

Resulting Condition Code:
0 Difference is zero
1 Difference is less than zero
2 Difference is greater than zero

3 Overflow

Program Exceptions:
Access (fetch, operand 2)
Fixed-Point Overflow

Programming Note
See Extended Mnemonic Instructions on page 24

Subtract Logical

SLR **R_1,R_2** **[RR]**

1F	R_1	R_2	
31	23	19	15 … 0

SL **R_1,D_2(X_2,B_2)** **[RX]**

5F	R_1	X_2	B_2	D_2
31	23	19	15	11 … 0

The second operand is subtracted from the first operand, and the difference is placed in the first operand location. The occurrence of a carry out of the sign position is recorded in the condition code.

Logical subtraction is considered to be performed by adding the one's complement of the second operand and a low-order one to the first operand. All 32 bits of both operands participate, without further change to the resulting sign bit. The instruction differs from SUBTRACT in the meaning of the condition code and in the absence of the interruption for overflow.

If a carry out of the sign position occurs, the leftmost bit of the condition code is made one. In the absence of a carry, the bit is made zero. When the sum is zero, the rightmost bit of the condition code is made zero. For a nonzero sum, the bit is made one.

Resulting Condition Code:
0 -
1 Difference is not zero, with no carry
2 Difference is zero, with carry
3 Difference is not zero, with carry

Program Exceptions:
Access (fetch, operand 2 of SL only)

Programming Note
the use of the one's complement and the low-order one instead of the two's complement of the second operand results in the recognition of carry when subtracting zero or the maximum negative number.

A zero difference cannot be obtained without a carry out of the sign position.

Test Under Mask

TM **$D_1(B_1),I_2$** **[SI]**

91	I_2	B_1	D_1
31	19	15	11 0

The state of the first-operand bits selected by a mask is used to set the condition code.

The byte of immediate data, I3, is used as an eight-bit mask. The bits of the mask are made to correspond one for one with the bits of the character in storage specified by the first-operand address.

A mask bit of one indicates that the storage bit is to be tested. When the mask bit is zero, the storage bit is ignored. When all storage bits thus selected are zero, the condition code is made 0. The code is also made 0 when the mask is all zeros. When the selected bits are all ones, the code is made 3 ; otherwise , the code is made 1. The character in storage is not changed.

Access Exceptions associated with the storage operand are recognized f or one byte , even when the mask is all zeros.

Resulting Condition Code:
0 Selected bits all zeros ; or the mask is all zeros
1 Selected bits mixed zeros and ones
2 -
3 Selected bits all ones

Program Exceptions:
Access (fetch, operand 1)

TM Test under Mask

Programming Note
The immediate byte's position in the operand is reversed. Given a word-aligned address D, bytes will be accessed as follows:

Address	Bits
D	D(31:24)
D+1	D(23:16)
D+2	D(15:8)
D+3	D(7:0)

Relevant Extended Mnemonic Instructions:

Mnemonic	Meaning	Op Code
BO or BOR	Branch if Ones	BC or BCR 1
BM or BMR	Branch if Mixed	BC or BCR 4
BZ or BZR	Branch if Zeroes	BC or BCR 8
BNO or BNOR	Branch if Not Ones	BC or BCR 14

Table 5: SSP Specific Instructions

Op Code	**Mnemonic**	**Name**	**Type**
04	SPM	SET PROGRAM MASK	RR
0A	BCTR0	SPECIAL BCTR	RR
0B	BCR0	SPECIAL BCR	RR
09	BALR0	SPECIAL BALR	RR
44	BALM	BRANCH AND SAVE MASKS	RR
03	BALRM	BRANCH AND SAVE MASKS	RR
08	BALR0M	SPECIAL BALR (SAVE MASKS)	RR
20	LAF	LOAD ADDRESS, AS UP	RX
21	BCTF	BRANCH ON COUNT, AS UP	RX
22	BCF	BRANCH ON CONDITION, AS UP	RX
23	STF	STORE, AS UP	RX
24	NF	AND, AS UP	RX
25	OF	OR, AS UP	RX
26	XF	EXCLUSIVE OR, AS UP	RX
27	LF	LOAD, AS UP	RX
28	CF	COMPARE, AS UP	RX
29	AF	ADD, AS UP	RX
2A	SF	SUBTRACT, AS UP	RX
30	FPAI	PRIMARY ADDRESS, IGNORE SS	RX
31	FPWNTAI	SECONDARY ADDRESS, IGNORE SS	RX
32	FPWIG	WRITE RANDOM, IGNORE SS	PL
60	FPA	PRIMARY ADDRESS	RX
61	FPWNTA	SECONDARY ADDRESS	RX
62	FPR	READ RANDOM	RX
63	FPRNTA	READ SECONDARY ADDRESS	RX
64	FPRB2	READ BLOCK (SS=2)	RX
65	FPRB2C	READ BLOCK (SS=2, COUNT GUARD)	PL
66	FPRBC	READ BLOCK (COUNT)	PL
67	FPW	WRITE RANDOM	PL
68	FPWI	WRITE RANDOM INDIRECT	PL
69	FPWBC	WRITE BLOCK (COUNT)	PL
6E	WCSR0,HALT	WRITE CSR#0	RX
6F	RCSR0	READ CSR#0	RX
7A	FR	PRIMARY, SECONDARY, READ RANDOM	PL
7B	FRB2	PRIMARY, SECONDARY, READ BLOCK	PL
7C	FRB2C	PRIMARY, SECONDARY, READ BLOCK	PL
7D	FRBC	PRIMARY, SECONDARY, READ BLOCK	PL
7E	FW	PRIMARY, SECONDARY, WRITE RAND.	PL
7F	FWBC	PRIMARY, SECONDARY, WRITE BLOCK	PL

Table 5: SSP Specific Instructions (continued)

Op Code	Mnemonic	Name	Type
72	FSR	SECONDARY, READ RANDOM	PL
73	FSRB2	SECONDARY, READ BLOCK	PL
74	FSRB2C	SECONDARY, READ BLOCK	PL
75	FSRBC	SECONDARY, READ BLOCK	PL
76	FSW	SECONDARY, WRITE RANDOM	PL
77	FSWBC	SECONDARY, WRITE BLOCK	PL
78	FSWI	SECONDARY, WRITE RANDOM INDIRECT	PL
80	SSM	SET SYSTEM MASK	S
81	LMSK	LOAD MASKS	S
82	LPSW	LOAD PSW	S
9A	SLLB	SHIFT LEFT LOGICAL BYTE	RS
9B	SRLB	SHIFT RIGHT LOGICAL BYTE	RS
9C	SLAB	SHIFT LEFT BYTE	RS
9D	SRAB	SHIFT RIGHT BYTE	RS
9E	SR32	SHIFT RIGHT 32	RS
99	WCSR0X	WRITE CSR#0	RS
A2	LPC	LOAD PC (16 BIT)	S

SSP INSTRUCTIONS

FASTBUS I/O:

The following conditions are for all FASTBUS I/O instructions:

1. All FASTBUS cycles having a Primary Address will always perform a BUS Arbitration.

2. All FASTBUS cycles having a Primary Address will drop and re-establish the AS/AK lock.

3. Any non-I/O instruction will break the AS/AK lock, with the exception of the 'F' series instructions; see page 71.

4. All FASTBUS cycles returning an SS code other than 0 will cause an error with the exception of block transfers that normally terminate with an SS=2.

5. A Data-Memory wrap-around during any FASTBUS block transfer will generate an error.

6. The R1 field is ignored for all FASTBUS I/O except during Primary Address cycles. During a Primary Address cycle, the two LSB of R1 are used to select the MS code as shown in Table 6.

Table 6: Address Time MS Codes

MS	Address Mode	
00	Specific device	- Data space
01	Specific device	- CSR space
10	Broadcast	- Data space
11	Broadcast	- CSR space

FASTBUS I/O Instruction Format:

Primary Address

FPA **MS,D_2(X_2,B_2)** [**RX**]

60	MS	X_2	B_2	D_2
31	23	19	15	11 … 0

The second operand is used as the primary address. Bus Arbitration is part of the Primary Address. If the Scanner does not already have mastership it will try to gain mastership. The MS field gives the MS code for the Primary Address.

Secondary Address

FPWNTA $\mathbf{D_2(X_2,B_2)}$ **[RX]**

61		X_2	B_2	D_2	
31	23	19	15	11	0

The second operand is used as the secondary address.

Read Random

FPR $\mathbf{D_2(X_2,B_2)}$ **[RX]**

62		X_2	B_2	D_2	
31	23	19	15	11	0

Data is stored into the second operand location.

Read Secondary Address

FPRNTA $\mathbf{D_2(X_2,B_2)}$ **[RX]**

63		X_2	B_2	D_2	
31	23	19	15	11	0

The Next Transfer Address is stored into the second operand location.

Read Block

FPRB2 $\mathbf{D_2(X_2,B_2)}$ **[RX]**

64		X_2	B_2	D_2	
31	23	19	15	11	0

Data is stored starting at the second operand location. The block is terminated normally upon receipt of an SS=2.

Read Block with a Word Count as a guard.

FPRB2C $\mathbf{D_2(X_2,B_2)}$ **[PL]**

65		X_2	B_2	D_2
31	23	19	15	11 … 0

The second operand address points to a parameter list containing the following 2 parameter(s):

1. Word Count

2. Starting address of the block

The block is terminated normally upon receipt of an SS=2. Exhausting the word count will generate an error.

Read Block with a Word Count.

FPRBC $\mathbf{D_2(X_2,B_2)}$ **[PL]**

66		X_2	B_2	D_2
31	23	19	15	11 … 0

The second operand address points to a parameter list containing the following 2 parameter(s):

1. Word Count

2. Starting address of the block

The block is terminated normally upon exhausting the word count.

Write Random

FPW $\mathbf{D_2(X_2,B_2)}$ **[PL]**

67		X_2	B_2	D_2
31	23	19	15	11 … 0

The second operand is used as the data.

Write Random Indirect

FPWI $\mathbf{D_2(X_2,B_2)}$ **[PL]**

68	/	X_2	B_2	D_2
31	23	19	15	11 0

The second operand is used as the address of the data.

Write Block with a Word Count

FPWBC $\mathbf{D_2(X_2,B_2)}$ **[PL]**

69	/	X_2	B_2	D_2
31	23	19	15	11 0

The second operand address points to a parameter list containing the following 2 parameter(s):

1. Word Count
2. Starting address of the block

The block is terminated normally by exhausting the word count.

Write Block with a Word Count, Non-Handshake

FPWPB $\mathbf{D_2(X_2,B_2)}$ **[PL]**

6A	/	X_2	B_2	D_2
31	23	19	15	11 0

The second operand address points to a parameter list containing the following 2 parameter(s):

1. Word Count
2. Starting address of the block

The block is terminated normally by exhausting the word count. The timing of the pipeline transfer is controlled by a DIPswitch on the control board. The cycle time may be selected in units of 20ns. Following the last data cycle, the SSP waits a fixed time (determined by microcode) before terminating the operation (dropping AS).

Secondary Address, Read Random

FSR $\mathbf{D_2(X_2,B_2)}$ **[PL]**

72		X_2	B_2	D_2
31	23	19	15	11 0

The second operand address points to a parameter list containing the following 2 parameter(s):

1. Secondary Address
2. Address to store data into

Secondary Address, Read Block.

FSRB2 $\mathbf{D_2(X_2,B_2)}$ **[PL]**

73		X_2	B_2	D_2
31	23	19	15	11 0

The second operand address points to a parameter list containing the following 2 parameter(s):

1. Secondary Address
2. Starting address of the block

The block is terminated normally upon receipt of an SS=2.

Secondary Address, Read Block with a word count guard.

FSRB2C $\mathbf{D_2(X_2,B_2)}$ **[PL]**

74		X_2	B_2	D_2
31	23	19	15	11 0

The second operand address points to a parameter list containing the following 3 parameter(s):

1. Secondary Address
2. Word count
3. Starting address of the block

The block is terminated normally upon receipt of an SS=2. Exhausting the word count will generate an error.

Secondary Address, Read Block with a word count.

FSRBC $\mathbf{D_2(X_2,B_2)}$ [**PL**]

75		X_2	B_2	D_2
31	23	19	15	11 0

The second operand address points to a parameter list containing the following 3 parameter(s):

1. Secondary Address
2. Word Count
3. Starting address of the block

The block is terminated normally upon exhausting the word count.

Secondary Address, Write Random.

FSW $\mathbf{D_2(X_2,B_2)}$ [**PL**]

76		X_2	B_2	D_2
31	23	19	15	11 0

The second operand address points to a parameter list containing the following 2 parameter(s):

1. Secondary Address
2. Data

Secondary Address, Write Block with a word count.

FSWBC $\mathbf{D_2(X_2,B_2)}$ [**PL**]

77		X_2	B_2	D_2
31	23	19	15	11 0

The second operand address points to a parameter list containing the following 3 parameter(s):

1. Secondary Address

2. Word Count
3. Starting Address of the block

The block is terminated normally by exhausting the word count.

Secondary Address, Write Random indirect.

FSWI $\mathbf{D_2(X_2,B_2)}$ **[PL]**

78		X_2	B_2	D_2	
31	23	19	15	11	0

The second operand address points to a parameter list containing the following 2 parameter(s):

1. Secondary address
2. Address of data

Primary, Secondary Address, Read Random.

FR $\mathbf{MS,D_2(X_2,B_2)}$ **[PL]**

7A	MS	X_2	B_2	D_2	
31	23	19	15	11	0

The second operand address points to a parameter list containing the following 3 parameter(s):

1. Primary Address
2. Secondary Address
3. Address to store data

The MS field gives the MS code for the Primary Address. Bus Arbitration is part of the Primary Address. If the SSP does not already have mastership it will try to gain mastership.

Primary, Secondary Address, Read Block

FRB2 $\mathbf{MS,D_2(X_2,B_2)}$ **[PL]**

7B	MS	X_2	B_2	D_2	
31	23	19	15	11	0

The second operand address points to a parameter list containing the following 3 parameter(s):

1. Primary Address
2. Secondary Address
3. Starting Address of data

The block is terminated normally by an SS=2. The MS field gives the MS code for the Primary Address.

Primary, Secondary Address, Read Block with a word count guard.

FRB2C **MS,D$_2$(X$_2$,B$_2$)** [**PL**]

7C	MS	X_2	B_2	D_2
31	23	19	15	11 0

The second operand address points to a parameter list containing the following 4 parameter(s):

1. Primary Address
2. Secondary Address
3. Word Count
4. Starting Address of data

The block is terminated normally upon receipt of an SS=2. Exhausting the word count will generate an error. The MS field gives the MS code for the Primary Address.

Primary, Secondary Address, Read Block with a word count.

FRBC **MS,D$_2$(X$_2$,B$_2$)** [**PL**]

7D	MS	X_2	B_2	D_2
31	23	19	15	11 0

The second operand address points to a parameter list containing the following 4 parameter(s):

1. Primary Address
2. Secondary Address
3. Word Count
4. Starting Address of data

The block is terminated normally upon exhausting the word count. The MS field gives the MS code for the Primary Address.

Primary, Secondary Address, Write Random.

FW **MS,D$_2$(X$_2$,B$_2$)** [**PL**]

7E	MS	X_2	B_2	D_2
31	23	19	15	11 0

The second operand address points to a parameter list containing the following 3 parameter(s):

1. Primary Address
2. Secondary Address
3. Data

The MS field gives the MS code for the Primary Address.

Primary, Secondary Address, Write Block.

FWBC **MS,D$_2$(X$_2$,B$_2$)** [**PL**]

7F	MS	X_2	B_2	D_2
31	23	19	15	11 0

The second operand address points to a parameter list containing the following 4 parameter(s):

1. Primary Address
2. Secondary Address
3. Word Count
4. Starting Address of data

The block is terminated normally by exhausting the word count. The MS field gives the MS code for the Primary Address.

BLOCK TRANSFERS

The SSP is capable of block transfers as a SLAVE to and from both DATA space and CSR space. In DATA space, the SSP will generate SS=6 when writing more than 512K (or 1M, 2M) words is attempted. In CSR space, the SSP will generate SS=6 when writing more than 4K (64K with EPM) words into program memory or 512K to data memory is attempted. The SSP will respond to block (MS=1) or pipeline (MS=3) transfers

As a MASTER, the SSP is capable of block transfers to and from its DATA memory only. There are 3 basic block read instructions and 1 block write instruction. The block instructions can be preceded by a secondary address or a primary and secondary address. See section on FASTBUS Instructions. The 3 basic block read instructions are:

1. FPRB2 - Normal termination on SS=2
2. FPRB2C - Normal termination on SS=2 with a word count guard
3. FPRBC - Normal termination on exhausting the word count

The basic block write instruction is:

1. FPWBC - Normal termination on exhausting the word count

When the I/O-Error handler is called because of an I/O error, the PC indicated by the PSW, saved in DM 0, will point to the instruction following the instruction where the error occurred.

FPRB2, FSRB2, FRB2:
These instructions terminate normally on SS=2. Upon receipt of SS=2, the SSP stores the updated NTA at DM 4 and continues on to the next instruction. The address contained in the MAR (DM 4) points to the location where the last data would have been stored had SS=0.

Any SS response besides 0 or 2 will generate an I/O-Error. The NTA will be stored into DM 4 as above. The PSW will also be stored into DM 0 indicating the PC and the SS code that caused the error.

FPRB2C, FSRB2C, FRB2C:
These instructions terminate normally on SS=2 as the B2 instructions. However, the RBC instructions have a word count associated with them. The word count is a guard in case a module sends back more data than buffer space was allowed for.

As with the B2 instructions, an SS=2 response will cause the updated NTA to be stored in DM 4. An SS response other than 0 or 2 will cause an I/O-Error.

If the word count is exhausted before receipt of an SS=2, an I/O-Error will be generated. The address contained in DM4 will point to the location where the data would have been stored had the word-count not been exhausted. The PSW will indicate an EXCEPTION=3, word count overflow.

FPRBC, FSRBC, FRBC:
These instructions terminate normally when the word count is exhausted. The PSW and updated NTA are not stored anywhere on normal termination.

If the block transfer terminates because of an SS!=0 response, the PSW and updated NTA will be stored in DM 0 and DM 4. The address contained in DM 4 points to the location where the data would have been stored had SS=0.

FPWBC, FSWBC, FWBC:
These instructions terminate normally when the word count is exhausted. The PSW and updated NTA are not stored anywhere on normal termination.

If the block transfer terminates because of an SS!=0 response, the PSW and updated NTA will be stored in DM 0 and DM 4. The NTA will be 8 greater than the address of the last data written. The number of words transferred will be the NTA minus the starting address divided by 4 minus 2.

SSP-SPECIFIC NON-FASTBUS INSTRUCTIONS

Shift Right Logical Bytes

SRLB **R_1,D_2(B_2)** **[RS]**

9B	R_1		B_2	D_2
31	23	19	15	11 ... 0

The first operand is shifted right the number of bytes specified by the second operand address. Example: 9B300002 will shift R3 right 2 bytes.

Also

9A	SLLB	SHIFT LEFT LOGICAL BYTE	RS
9C	SLAB	SHIFT LEFT BYTE	RS
9D	SRAB	SHIFT RIGHT BYTE	RS

Shift Right 32 bits

SR32 **[RS]**

9E	R_1	
31	23	19 ... 0

The first operand is shifted right by 32 bits into register (R1+1). The sign bit is propagated in R1. Given initial contents R6 = 8110111, R7 = 1234, then following 'SR32,6' R6 = FFFFFFFF, R7 = 8110111

Set System Mask

SSM **D_2(B_2)** **[S]**

80		B_2	D_2
31	23	15	11 ... 0

bit 15 of the operand is loaded into the System Mask. Other bits are ignored.

Load Processor Status Word

LPSW **D_2(B_2)** **[S]**

82		B_2	D_2
31	23	15	11 ... 0

Bits 15-0 of the operand is loaded into the PSW (System Mask, Overflow mask, Condition codes and

PC) Other bits are ignored. Should not be used with EPM enabled.

Write CSR#0

WCSR0X $\mathbf{D_2(B_2)}$ **[S]**

99		B_2	D_2
31	23	15	11 0

The operand is written into CSR#0

Write CSR#0

WCSR0 $\mathbf{D_2(X_2,B_2)}$ **[RX]**

6E		X_2	B_2	D_2
31	23	19	15	11 0

The operand is written into CSR#0

Read CSR#0

RCSR0 $\mathbf{D_2(X_2,B_2)}$ **[RX]**

6F		X_2	B_2	D_2
31	23	19	15	11 0

The contents of CSR#0 are written to the operand

Set Program Mask

SPM $\mathbf{R_1}$ **[RR]**

04	R_1	
31	23	19 0

Bits 12-14 of the register specified by R1 replace the condition code and program mask bits of the current PSW. The contents of the register are unchanged. The remaining bits are ignored.

Load Masks

LMSK $\mathbf{D_2(B_2)}$ **[S]**

81		B_2	D_2
31	23	15	11 0

Bits 15-12 of the operand are loaded into the PSW (System Mask, Overflow mask, Condition codes) Other bits are ignored.

Load Program Counter

LPC $\mathbf{D_2(B_2)}$ **[S]**

A2		B_2	D_2
31	23	15	11 0

Bits 15-0 of the operand is loaded into the PC (16-bit program counter for Extended PM) Other bits are ignored.

Branch and Link Masks

BALR0M $\mathbf{R_1,R_2}$ **[RR]**

08	R_1	R_2	
31	23	19	15 0

The program masks are saved in the register R1. No branch is performed. The PC is not saved.

Also

03	BALRM	RR	BALR masks
44	BALM	RX	BAL masks

Notes on some IBM SSP instructions:

BCTR0, BCR0, BALR0 (0A,0B,09) are special versions of the IBM opcodes BCTR, BCR, BALR which do not perform a branch. Thus to achieve the function of the IBM code 'BCR M1,0' the code 'BCR0 M1,0' must be used. The BNL cross-assembler makes this substitution automatically.

Read-Modify-Write FASTBUS Instructions

The following opcodes may be used to construct Read-Modify-Write FASTBUS operations. AS remains up until the next primary address or normal non-FASTBUS instruction.

Code	Mnemonic	Type	Description
20	LAF	RX	Load address, AS up
21	BCTF	RX	Branch on Count, AS up
22	BCF	RX	Branch conditional, AS up
23	STF	RX	Store, AS up
24	NF	RX	And, AS up
25	OF	RX	OR, AS up
26	XF	RX	X-OR, AS up
27	LF	RX	Load, AS up
28	CF	RX	Compare, AS up
29	AF	RX	Add, AS up
2A	SF	RX	Subtract, AS up

The following instructions may be used to construct a FASTBUS instruction to ignore SS codes at Primary or Secondary address and perform a subsequent data cycle

30	FPAI	RS	Primary address, ignore SS codes
31	FPWNTAI	RS	Secondary address, ignore SS codes
32	FPWIG	RS	FASTBUS write, ignore SS code

** FPWNTAI does not work correctly

**

HARDWARE CONFIGUREABLE ITEMS

CPU Board Jumpers

The CPU board may be fitted with either 2MB of Data Memory RAM using 256Kbit devices, or either 4Mb or 8MB using 1Mbit devices. To accommodate these different sizes of Data Memory, two jumper areas are used. The MSB jumper (near I3J3) selects the Most Significant Bit for RAM Bank enable. The OVF (near I3I5) jumper selects the overflowed address bit and is used to generate the address wraparound error. The jumpers are set as follows:

MSB	OVF	DM Size	Configuration
20	21	2MB	$64 \times$ 256Kbit
21	22	4MB	$32 \times$ 1Mbit
22	23	8MB	$64 \times$ 1Mbit

Control Board Jumpers

To generate an illegal address error (SS=6) during a block transfer operation to Data Memory, a jumper area (JX09,10,11 near J1B2) is used as follows:

DM Size	Configuration
2MB	$64 \times$ 256Kbit
4MB	$32 \times$ 1Mbit
8MB	$64 \times$ 1Mbit

Jumpers JX01,2 (near J9J3) select either single cycle or single instruction stepping under control of the Step button on the front panel, when enabled by CSR#0¡6¿.

Jumpers JX04,5 (near G4C5) select either a DC level or a pulse on the RB line when CSR#0¡3¿ is written.

Jumpers JX12,13,14 selects the Vcc supply voltage for the FASTBUS Cable Segment Receivers. Either 10114 ECL line receivers or F6901 ASICs may be fitted. The F6901 receivers allow more modules to be connected on one cable segment than the 10114s.

Programmable Logic Devices

The Data Memory Size must be programmed into the ADRDEC decoder chip (M2A6) in addition to setting the jumpers.

If it is desired that the SSP respond to any of the Manufacturer's defined Broadcasts (Case 8, segment address 8D - FD), the two BCPROMs D5C1 and D9J1 (10149) must be programmed accordingly.

Other Components

When the Cable Segment Drivers are fitted, but no Cable Segment is connected, the segment lines may sometimes float high. If the Cable RB, WT or AS lines are high the SSP will hang. Socketed 10K resistors R021, R022, R042 (near F5L8) will pull these lines low; they may be removed if necessary when a cable segment is connected.

Sockets are provided for Cable Segment drivers. Either CERN hybrid drivers or KEK-CSD hybrids may be fitted. Pin 10 of the CERN hybrids must be set at -1.4V. This voltages ($V_B B$) is supplied by the Cable Segment receivers. Pin 10 of the KEK hybrids is grounded internally; accordingly pin 10 of KEK hybrids must be cut for correct operation of the receivers.

A DIPswitch (J9L9) controls the data cycle length for the FPWBP pipeline write block transfer operation. The delay is 2's complement binary coded in units of 20ns. All switches open corresponds to a maximum DS - DS period of around 300ns.

Table 7: SSP Instructions (alphabetic order)

Op Code	Mnemonic	Name	Type
5A	A	ADD	RX
29	AF	ADD, AS UP	RX
4A	AH	ADD HALFWORD	RX
5E	AL	ADD LOGICAL	RX
1E	ALR	ADD LOGICAL	RR
1A	AR	ADD	RR
45	BAL	BRANCH AND LINK	RX
44	BALM	BRANCH AND SAVE MASKS	RR
05	BALR	BRANCH AND LINK	RR
09	BALR0	SPECIAL BALR	RR
03	BALRM	BRANCH AND SAVE MASKS	RR
08	BALR0M	SPECIAL BALR (SAVE MASKS)	RR
47	BC	BRANCH ON CONDITION	RX
22	BCF	BRANCH ON CONDITION, AS UP	RX
07	BCR	BRANCH ON CONDITION	RR
0B	BCR0	SPECIAL BCR	RR
46	BCT	BRANCH ON COUNT	RX
21	BCTF	BRANCH ON COUNT, AS UP	RX
06	BCTR	BRANCH ON COUNT	RR
0A	BCTR0	SPECIAL BCTR	RR
86	BXH	BRANCH ON INDEX HIGH	RS
87	BXLE	BRANCH ON INDEX LOW OR EQUAL	RS
59	C	COMPARE	RX
28	CF	COMPARE, AS UP	RX
49	CH	COMPARE HALFWORD	RX
55	CL	COMPARE LOGICAL	RX
95	CLI	COMPARE LOGICAL IMMEDIATE	SI
15	CLR	COMPARE LOGICAL	RR
19	CR	COMPARE	RR
5D	D	DIVIDE	RX
1D	DR	DIVIDE	RR
60	FPA	PRIMARY ADDRESS	RX
30	FPAI	PRIMARY ADDRESS, IGNORE SS	RX
62	FPR	READ RANDOM	RX
64	FPRB2	READ BLOCK (SS=2)	RX
65	FPRB2C	READ BLOCK (SS=2, COUNT GUARD)	PL
66	FPRBC	READ BLOCK (COUNT)	PL
63	FPRNTA	READ SECONDARY ADDRESS	RX
67	FPW	WRITE RANDOM	PL

Table 7: SSP Instructions (alphabetic order cont.)

Op Code	Mnemonic	Name	Type
69	FPWBC	WRITE BLOCK (COUNT)	PL
68	FPWI	WRITE RANDOM INDIRECT	PL
32	FPWIG	WRITE RANDOM, IGNORE SS	PL
6A	FPWBP	WRITE PIPELINED (COUNT)	PL
61	FPWNTA	SECONDARY ADDRESS	RX
31	FPWNTAI	SECONDARY ADDRESS, IGNORE SS	RX
7A	FR	PRIMARY, SECONDARY, READ RANDOM	PL
7B	FRB2	PRIMARY, SECONDARY, READ BLOCK	PL
7C	FRB2C	PRIMARY, SECONDARY, READ BLOCK	PL
7D	FRBC	PRIMARY, SECONDARY, READ BLOCK	PL
72	FSR	SECONDARY, READ RANDOM	PL
73	FSRB2	SECONDARY, READ BLOCK	PL
74	FSRB2C	SECONDARY, READ BLOCK	PL
75	FSRBC	SECONDARY, READ BLOCK	PL
76	FSW	SECONDARY, WRITE RANDOM	PL
77	FSWBC	SECONDARY, WRITE BLOCK	PL
78	FSWI	SECONDARY, WRITE RANDOM INDIRECT	PL
7E	FW	PRIMARY, SECONDARY, WRITE RAND.	PL
7F	FWBC	PRIMARY, SECONDARY, WRITE BLOCK	PL
6E	HALT	WRITE CSR#0	RX
43	IC	INSERT CHARACTER	RX
58	L	LOAD	RX
27	LF	LOAD, AS UP	RX
41	LA	LOAD ADDRESS	RX
20	LAF	LOAD ADDRESS, AS UP	RX
13	LCR	LOAD COMPLEMENT	RR
48	LH	LOAD HALFWORD	RX
98	LM	LOAD MULTIPLE	RS
81	LMSK	LOAD MASKS	S
11	LNR	LOAD NEGATIVE	RR
A2	LPC	LOAD PC (16 BIT)	S
10	LPR	LOAD POSITIVE	RR
82	LPSW	LOAD PSW	S
18	LR	LOAD	RR
12	LTR	LOAD AND TEST	RR
5C	M	MULTIPLY	RX
4C	MH	MULTIPLY HALFWORD	RX
1C	MR	MULTIPLY	RR
92	MVI	MOVE IMMEDIATE	SI
54	N	AND	RX

Table 7: SSP Instructions (alphabetic order cont.)

Op Code	Mnemonic	Name	Type
24	NF	AND, AS UP	RX
94	NI	AND IMMEDIATE	SI
14	NR	AND	RR
56	O	OR	RX
25	OF	OR, AS UP	RX
96	OI	OR IMMEDIATE	SI
16	OR	OR	RR
6F	RCSR0	READ CSR#0	RX
5B	S	SUBTRACT	RX
2A	SF	SUBTRACT, AS UP	RX
4B	SH	SUBTRACT HALFWORD	RX
5F	SL	SUBTRACT LOGICAL	RX
8B	SLA	SHIFT LEFT SINGLE	RS
9C	SLAB	SHIFT LEFT BYTE	RS
8F	SLDA	SHIFT LEFT DOUBLE	RS
8D	SLDL	SHIFT LEFT DOUBLE LOGICAL	RS
89	SLL	SHIFT LEFT SINGLE LOGICAL	RS
9A	SLLB	SHIFT LEFT LOGICAL BYTE	RS
1F	SLR	SUBTRACT LOGICAL	RR
04	SPM	SET PROGRAM MASK	RR
1B	SR	SUBTRACT	RR
9E	SR32	SHIFT RIGHT 32	RS
8A	SRA	SHIFT RIGHT SINGLE	RS
9D	SRAB	SHIFT RIGHT BYTE	RS
8E	SRDA	SHIFT RIGHT DOUBLE	RS
8C	SRDL	SHIFT RIGHT DOUBLE LOGICAL	RS
88	SRL	SHIFT RIGHT SINGLE LOGICAL	RS
9B	SRLB	SHIFT RIGHT LOGICAL BYTE	RS
80	SSM	SET SYSTEM MASK	S
50	ST	STORE	RX
23	STF	STORE, AS UP	RX
42	STC	STORE CHARACTER	RX
40	STH	STORE HALFWORD	RX
90	STM	STORE MULTIPLE	RS
91	TM	TEST UNDER MASK	SI
6E	WCSR0	WRITE CSR#0	RX
99	WCSR0X	WRITE CSR#0	RS
57	X	EXCLUSIVE OR	RX
26	XF	EXCLUSIVE OR, AS UP	RX
97	XI	EXCLUSIVE OR IMMEDIATE	SI
17	XR	EXCLUSIVE OR	RR

Table 8: SSP Instructions (numeric order)

Op Code	Mnemonic	Name	Type
03	BALRM	BRANCH AND SAVE MASKS	RR
04	SPM	SET PROGRAM MASK	RR
05	BALR	BRANCH AND LINK	RR
06	BCTR	BRANCH ON COUNT	RR
07	BCR	BRANCH ON CONDITION	RR
08	BALR0M	SPECIAL BALR (SAVE MASKS)	RR
09	BALR0	SPECIAL BALR	RR
0A	BCTR0	SPECIAL BCTR	RR
0B	BCR0	SPECIAL BCR	RR
10	LPR	LOAD POSITIVE	RR
11	LNR	LOAD NEGATIVE	RR
12	LTR	LOAD AND TEST	RR
13	LCR	LOAD COMPLEMENT	RR
14	NR	AND	RR
15	CLR	COMPARE LOGICAL	RR
16	OR	OR	RR
17	XR	EXCLUSIVE OR	RR
18	LR	LOAD	RR
19	CR	COMPARE	RR
1A	AR	ADD	RR
1B	SR	SUBTRACT	RR
1C	MR	MULTIPLY	RR
1D	DR	DIVIDE	RR
1E	ALR	ADD LOGICAL	RR
1F	SLR	SUBTRACT LOGICAL	RR
20	LAF	LOAD ADDRESS, AS UP	RX
21	BCTF	BRANCH ON COUNT, AS UP	RX
22	BCF	BRANCH ON CONDITION, AS UP	RX
23	STF	STORE, AS UP	RX
24	NF	AND, AS UP	RX
25	OF	OR, AS UP	RX
26	XF	EXCLUSIVE OR, AS UP	RX
27	LF	LOAD, AS UP	RX
28	CF	COMPARE, AS UP	RX
29	AF	ADD, AS UP	RX
2A	SF	SUBTRACT, AS UP	RX
30	FPAI	PRIMARY ADDRESS, IGNORE SS	RX
31	FPWNTAI	SECONDARY ADDRESS, IGNORE SS	RX
32	FPWIG	WRITE RANDOM, IGNORE SS	PL

Table 8: SSP Instructions (numeric order cont.)

Op Code	Mnemonic	Name	Type
40	STH	STORE HALFWORD	RX
41	LA	LOAD ADDRESS	RX
42	STC	STORE CHARACTER	RX
43	IC	INSERT CHARACTER	RX
44	BALM	BRANCH AND SAVE MASKS	RR
45	BAL	BRANCH AND LINK	RX
46	BCT	BRANCH ON COUNT	RX
47	BC	BRANCH ON CONDITION	RX
48	LH	LOAD HALFWORD	RX
49	CH	COMPARE HALFWORD	RX
4A	AH	ADD HALFWORD	RX
4B	SH	SUBTRACT HALFWORD	RX
4C	MH	MULTIPLY HALFWORD	RX
50	ST	STORE	RX
54	N	AND	RX
55	CL	COMPARE LOGICAL	RX
56	O	OR	RX
57	X	EXCLUSIVE OR	RX
58	L	LOAD	RX
59	C	COMPARE	RX
5A	A	ADD	RX
5B	S	SUBTRACT	RX
5C	M	MULTIPLY	RX
5D	D	DIVIDE	RX
5E	AL	ADD LOGICAL	RX
5F	SL	SUBTRACT LOGICAL	RX
60	FPA	PRIMARY ADDRESS	RX
61	FPWNTA	SECONDARY ADDRESS	RX
62	FPR	READ RANDOM	RX
63	FPRNTA	READ SECONDARY ADDRESS	RX
64	FPRB2	READ BLOCK (SS=2)	RX
65	FPRB2C	READ BLOCK (SS=2, COUNT GUARD)	PL
66	FPRBC	READ BLOCK (COUNT)	PL
67	FPW	WRITE RANDOM	PL
68	FPWI	WRITE RANDOM INDIRECT	PL
69	FPWBC	WRITE BLOCK (COUNT)	PL
6A	FPWBP	WRITE PIPELINED (COUNT)	PL
6E	WCSR0,HALT	WRITE CSR#0	RX
6F	RCSR0	READ CSR#0	RX
72	FSR	SECONDARY, READ RANDOM	PL

Table 8: SSP Instructions (numeric order cont.)

Op Code	Mnemonic	Name	Type
73	FSRB2	SECONDARY, READ BLOCK	PL
74	FSRB2C	SECONDARY, READ BLOCK	PL
75	FSRBC	SECONDARY, READ BLOCK	PL
76	FSW	SECONDARY, WRITE RANDOM	PL
77	FSWBC	SECONDARY, WRITE BLOCK	PL
78	FSWI	SECONDARY, WRITE RANDOM INDIRECT	PL
7A	FR	PRIMARY, SECONDARY, READ RANDOM	PL
7B	FRB2	PRIMARY, SECONDARY, READ BLOCK	PL
7C	FRB2C	PRIMARY, SECONDARY, READ BLOCK	PL
7D	FRBC	PRIMARY, SECONDARY, READ BLOCK	PL
7E	FW	PRIMARY, SECONDARY, WRITE RAND.	PL
7F	FWBC	PRIMARY, SECONDARY, WRITE BLOCK	PL
80	SSM	SET SYSTEM MASK	S
81	LMSK	LOAD MASKS	S
82	LPSW	LOAD PSW	S
86	BXH	BRANCH ON INDEX HIGH	RS
87	BXLE	BRANCH ON INDEX LOW OR EQUAL	RS
88	SRL	SHIFT RIGHT SINGLE LOGICAL	RS
89	SLL	SHIFT LEFT SINGLE LOGICAL	RS
8A	SRA	SHIFT RIGHT SINGLE	RS
8B	SLA	SHIFT LEFT SINGLE	RS
8C	SRDL	SHIFT RIGHT DOUBLE LOGICAL	RS
8D	SLDL	SHIFT LEFT DOUBLE LOGICAL	RS
8E	SRDA	SHIFT RIGHT DOUBLE	RS
8F	SLDA	SHIFT LEFT DOUBLE	RS
90	STM	STORE MULTIPLE	RS
91	TM	TEST UNDER MASK	SI
92	MVI	MOVE IMMEDIATE	SI
94	NI	AND IMMEDIATE	SI
95	CLI	COMPARE LOGICAL IMMEDIATE	SI
96	OI	OR IMMEDIATE	SI
97	XI	EXCLUSIVE OR IMMEDIATE	SI
98	LM	LOAD MULTIPLE	RS
99	WCSR0X	WRITE CSR#0	RS
9A	SLLB	SHIFT LEFT LOGICAL BYTE	RS
9B	SRLB	SHIFT RIGHT LOGICAL BYTE	RS
9C	SLAB	SHIFT LEFT BYTE	RS
9D	SRAB	SHIFT RIGHT BYTE	RS
9E	SR32	SHIFT RIGHT 32	RS
A2	LPC	LOAD PC (16 BIT)	S

Index