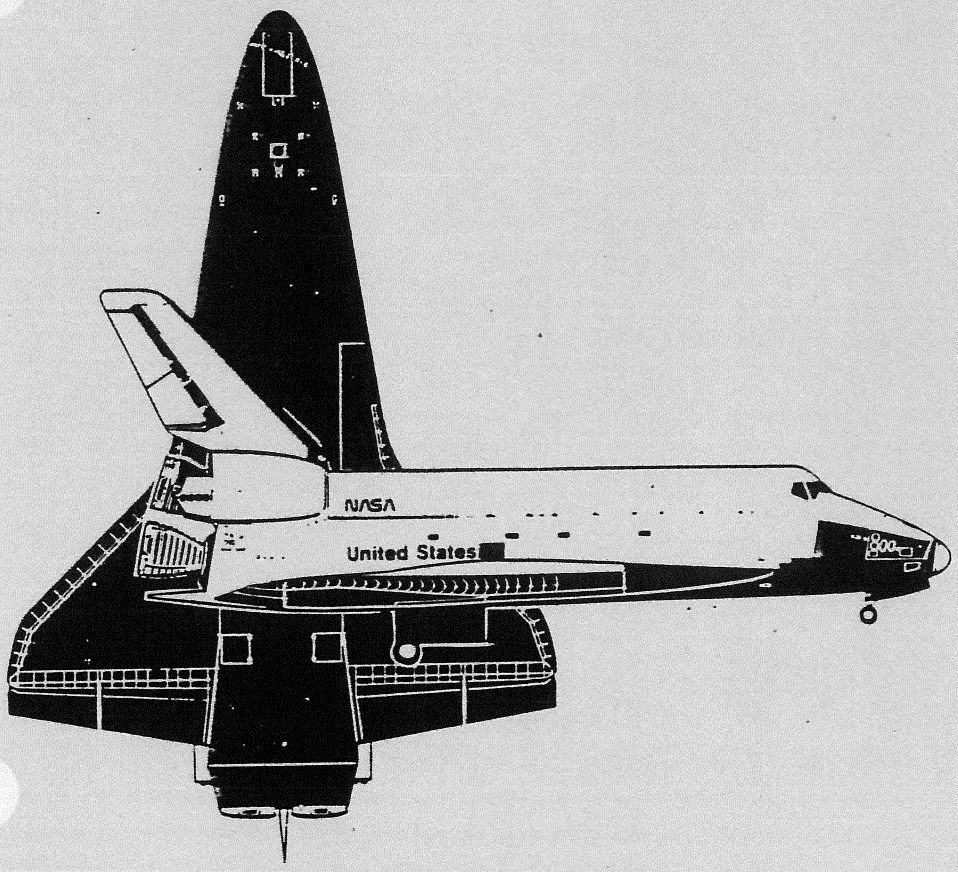
Space Shuttle Model AP-101S
Principles of Operation
with Shuttle Instruction Set



IBM

REV	DESCRIPTION	DATE	APPROVED
-	Release	01-16-85	
A	Changes from AP-101B & OASCB	07-09-85	
B	Update: Add IOP POO as appendix	04-30-86	
C	Update: Add clarifications & EDCP changes	05-01-87	
D	Update: Add clarifications	02-01-91	
E	Update: Add clarifications	05-03-91	
F	Update: Incorporated C.R. 90669A & 90954C	07-12-94	

TITLE

Space Shuttle AP-101S Principles of Operation

Document Control Number: 85-C67-001

IBM Federal Sector Division

EDCP	PAGE	DESCRIPTION OF CHANGE
	iii thru v	Single Margin Bars Incorporated To Indicate Changes To The AP-101B P00.
	2-1 thru 2-4, 2-11, 2-13 thru 2-22, 2-24, 2-25, 2-27, 2-29	
	3-1, 3-2	
	4-2 thru 4-5, 4-10, 4-11, 4-14, 4-24, 4-25, 4-31 thru 4-33	
	5-2, 5-10	
	6-7, 6-8, 6-11	
	7-17	
	8-1 thru 8-9, 8-12 thru 8-22, 8-24, 8-26 thru 8-34	
	9-1 thru 9-17, 9-19 thru 9-22	
	10-2	
	12-3	
	13-1, 13-2	
	15-1 thru 15-34	
	16-1 thru 16-10	
	17-1, 17-2 17-3	

EDCP	PAGE	DESCRIPTION OF CHANGE
	2-1, 2-3, 2-10, 2-13, 2-16, 2-20, 2-23, 2-24, 2-29 7-13 8-5, 8-7, thru 8-9, 8-11, 8-17, 8-19, 8-29, 8-31, 8-32, 8-34 9-4, 9-5, 9-8, 9-15, 9-19, 9-21 10-3 15-10 15-12 15-31 15-33 16-9	The following indicate changes from the Orbiter Avionics Software Change Board (OASCB) Baseline meeting held 02/28/85.

EDCP	PAGE	DESCRIPTION OF CHANGE
86-101S-011	Title	Add note containing margin bar definition.
86-101S-011	iv	Change Appendices A, B, C, to Sections 15, 16, 17, respectively. Add Appendices I, II, III (IOP P00).
86-101S-011	1-1	Change Appendix B reference to Section 16.
86-101S-008	2-21	Instruction Monitor Interrupt handling change.
86-101S-011	2-25	Rewrote Interrupt Priority description in an attempt to clarify Figure 2-20 on page 2-21. The length of this change necessitated an additional page such that material formerly on pages 2-25 thru 2-29 is now on pages 2-26 thru 2-30.
86-101S-011	8-2, 8-8, 8-10	Changed from one to multiple guard digits.
86-101S-011	10-3	Added a new paragraph to clarify ICR AGE command word format.
86-101S-006	10-3	Changed bit format to include the second soft error counter.
86-101S-011	10-3	Corrected typo in table: XFER to MFER.
86-101S-011	15-1 thru 15-34	Changed from A-1 thru A-34.
86-101S-011	15-19	Added statement explaining the effect on scrubbing for the Reset ECC Bits assist command.
86-101S-011	16-1 thru 16-10	Changed from B-1 thru B-10.
86-101S-011	17-1 thru 17-4	Changed from C-1 thru C-4.
86-101S-005	17-3 17-4	Updated instruction times for LXA and LDM. Updated instruction time for STDM.
86-101S-011	17-4	Updated instruction times for SCAL and SRET. Reduced because DSE Registers are no longer stored.

EDCP	PAGE	DESCRIPTION OF CHANGE
		IOP POO 85-C67-004 is being incorporated into this document as Appendices I, II, III.
86-101S-011	Appendix I	
86-101S-011	I-4, I-5, I-7 thru I-7D, I-8, I-10, I-13, I-14, I-17, I-20, I-21, I-29 thru I-31, I-34, I-36 thru I-40, I-46 thru I-52	The following pages contain single margin bars which indicate changes from the original IOP POO and are being carried over into this document from 85-C67-004.
	Appendix II	
	II-18, II-51, II-71, II-94, II-97	
	Appendix III	
	III-14, III-17, III-18, III-19, III-21, III-30, III-39, III-74	

EDCP	PAGE	DESCRIPTION OF CHANGE
86-101S-011	Appendix I	
	I-7A	Clarification of data to be used for forcing errors.
	I-7D	Correct typing errors.
	I-8	Correct three typing errors. Clarify parity error symptoms and recovery.
	I-17	Nomenclature change.
	I-19	Clarification that all other interrupts are reset by this PCO.
	I-20	Removed applicable termination control latches information.
	I-21	Explanation that terminate output driver is permanently inhibited by the hardware.
	I-22	Same as I-21.
	I-24	Added note that Test and Enable do not force the new parity interrupts.
	I-36	Specified priority of interrupt error condition.
	I-39	Flagged Bits I-31 as unavailable.
	I-40	Added note that those errors do not generate interrupts. Clarified PS overtemperature. Flagged Bits 1, 2 as unavailable.
	I-43	Same as I-21.
	I-47	Changed HISAM dump from Bit 14 to Bit 15.

EDCP	PAGE	DESCRIPTION OF CHANGE
86-101S-011	Appendix II	
	II-13	Changed reference to AP-101S Design Workbook.
	II-14	Same as II-13. Corrected typing error.
	II-20	Removed fail latch reference in two places.
	II-24	Removed DMA error and DMA channel reference. Changed to parity error "during" instruction and data read.
	II-96	Added OPX field for diagnostic data flow error test.
	Appendix III	
	III-17	Corrected typing error.
	III-20	Removed fail latch reference in two places.
	III-25	Removed DMA error and channel logic reference. Segregated termination latch detected by IOP hardware.

EDCP	PAGE	DESCRIPTION OF CHANGE
86-101S-024	1-1	Added reference for AP-101S/AP-101B comparisons document.
86-101S-024	2-3	Removed comment about 128K or less programs using the DSEs.
86-101S-024	2-6	Added reference to Effective Address Generation Chart on page 11-1.
86-101S-024	2-16	Removed typo "Z" from "16-Bit Branch Address." Added explanation for IC relative expansion.
86-101S-024	2-18, 26	Move instruction monitor paragraph from 2-26 to 2-18.
86-101S-024	2-19	Added missing information to Bits 48-63 of Figure 2-19.
86-101S-024	2-20	Correct Instruction Address Bit designation.
86-101S-024	2-21	Corrections to Interrupt Structure and Priority Table.
86-101S-009	2-23	Added reserved area for BCE 25.
86-101S-024	2-29	Changed Decimal designations to Hex. Added Memory Store Protected note.
86-101S-024	3-1	Clarified IOP as I/O Device. Added note clarifying DMA can occur between CPU memory cycles.
86-101S-024	3-2	Clarified IOP as I/O device.
86-101S-024	4-5, 22, 32, 34 7-4, 5, 8, 9, 12, 15, 20, 9-5	Added warning note for DMA being enabled during instruction execution.
86-101S-024	9-2	Correct typesetting.
86-101S-024	9-17, 18	Clarified DMA is not allowed during fetch and storeback. Add counter execution times.
86-101S-024	10-3	Correct Soft Error Counter Bit designations. Added Counter Execution times and figure 10-1 reference.
86-101S-024	10-4	New Page: Added definition of MFER/MMU bits.
86-101S-024	13-1	Remove SRS, BROV and CRY from 1100 OP code.

EDCP	PAGE	DESCRIPTION OF CHANGE
86-101S-024	15-4, 10, 13 thru 16, 18, 19, 24, 26, 27, 28	Added missing execution times.
87-101S-047	2-22, 23; 9-8, 19, 20, 22, 17-3, 4	DSE instruction enhancement; text on page 2-23 moved back to page 2-22 to allow room for expanded Figure 2-21.
86-101S-024	2-1, 7-13, 8-27, 10-2 I-2, 3, 5, 7A, 7D, 8, 17, 35, 40, 45, 52, II-24, 25, III-5, 19 thru 22, 24 thru 26, 29, 30, 43, 54, 55, 59, 74, 75, 81	Clarifications and typo corrections.
87-101S-047	17-3, 4	LDM, LXA, STDM, STXA execution changed.
86-101S-009	I-7D, II-25, III-27	Added details of IOP shutdown when an IOP Data Flow Parity Error occurs.
86-101S-024	I-13 thru I-15, I-45 thru I-49	Changed DO/DI Bit designations to correspond to ALD's and Specification. Flagged High Speed Discretes. Clarified Sync Discrete numbering.
86-101S-009	I-8, 17	Processor 25 update.
86-101S-024	I-22	Clarified Note.
86-101S-024	I-25	Removed Bit 25 as self test.
86-101S-024	I-36	Rephrased for multiple error occurrence.
86-101S-024	II-15	Added Bit Alignment note to IL description.
86-101S-024	II-70	Changed & NIX to @ NIX.
86-101S-024	II-97	Added explanation of OP code 011.

EDCP	PAGE	DESCRIPTION OF CHANGE
86-101S-024	III-6	Added "Common IOP Addr" in IUA field.
86-101S-024	III-7	Corrected half word numbers. Added note for CPU and IOP Memory Addressing.
86-101S-024	III-10	Changed "I" field to "M". PC clarification for "DISP" field.
87-101S-049	III-14, 15, 76	Changed for #MIN instruction update.
86-101S-024	III-14	Added reference paragraphs.
86-101S-024	III-15, 20, 48, 52	Corrected gap time.
86-101S-024	III-17	Added paragraph for BCE programmable registers.
86-101S-024	III-27	Added section for MIA Busy when asked to transmit.
86-101S-024	III-32	Removed nonapplicable programming note detailing indexing.
86-101S-024	III-42	Added reference to listen mode/command mode differences.
86-101S-024	III-44	Added gap time details.
86-101S-024	III-50	Added paragraph for IUAR loading during #CMD and #CMDI.
86-101S-024	III-52	Removed nonapplicable programming note.
86-101S-024	III-53, 56	Added paragraph for GPC to GPC word transfers.
86-101S-024	III-65	Added #CMDI reference for Listen Mode.
86-101S-024	III-70	Added typical time out detection time.
86-101S-024	III-76	BCE IUAR in listen mode reference.
86-101S-010	III-80	Added details for real time MIA parity checking. Changed "I" field to "M".

EDCP	PAGE	DESCRIPTION OF CHANGE
86-101S-024	III-81	Added note for additional listen mode implementation.
86-101S-024	III-86	Removed nonapplicable paragraph.
86-101S-024	III-87	Separated #CMDI from #CMD.

EDCP	PAGE	DESCRIPTION OF CHANGE
NA	1-1	Deleted paragraph referencing the "AP-101B/AP-101S Comparisons" document.
NA	2-15	Missing word added. Added word "to".
NA	2-16	Text added to clarify the action of the second stage addressing when the high order address bit = 0.
NA	2-18	Deleted text concerning PSW bit 45.
		Added a line at the top of the page that was left out from the old GPC P00.
NA	2-21	Mask for CPU store protect revised from bit 45 to "--". (CPU store protect not maskable).
NA	2-21	Made a note on the old PSW designation for CPU multibit error.
NA	2-21	Interrupt priorities changed to put the EXT 1 INT (AGE) ahead of the other EXT INTs.
		Footnote added about CPU multibit error as referenced on page 2-25.
NA	2-22	Reference to bit 45 mask deleted.
NA	2-25	Note added on CPU multibit error.
NA	2-25	Group 0 interrupts section clarified.
NA	2-26	Text concerning CPU store protect mask bit 45 deleted.
NA	2-27	BCE 25 processor storage (00A4-00A5) added to list of PSA locations to not be store protected.
NA	4-24	Typographical error corrected. Added an "s" to replace.
NA	5-2	Programming note changed.
NA	7-13	Corrected typographical error.
NA	7-14	"Do not exceed eight" changed to "do not exceed sixteen".
NA	8-2	Sign corrected on exponents.
NA	8-15	CVFL diagram replaced with copy from AP-101B P00.

EDCP	PAGE	DESCRIPTION OF CHANGE
NA	8-26	Corrected typographical error.
NA	9-2	Changed reference from "Appendix A" to "Section 15."
NA	9-8	Last sentence under programming note - corrected from STM to STDM. Flow chart corrected. Programming note clarified for instruction main store addresses crossing 32K boundaries.
NA	9-20	The R1 designator was deleted and a note was added at the bottom of the page.
NA	9-21	Corrected wording in description section for bit 20 and 27.
NA	9-22	The R1 designator was deleted and a note was added at the bottom of the page.
NA	10-3	The ICR instruction operation was clarified by expanding the code column to 32 bits.
NA	13-1	SRS branch on count deleted.
NA	15-1	Changed word "Reference" to "Section."
NA	15-19	Added I/O delay times.
NA	15-29	External 4 interrupt added. Note added to bottom of page.
NA	15-33	An error code of 70 "EDAC error during reset" added.
NA	I-2	Changed description for bits 17 through 31 from "NOT USED" to "IGNORED."
NA	I-4	Information pertaining to AP-101B crew trainers and prototypes deleted.
NA	I-4	Four occurrences of "spare not used in flight IOP" deleted.
NA	I-10	Bits 25 - 31 bracketed.
NA	I-22	Removed note.
NA	I-24	The effect of the Test Interrupts PCO on the interrupt registers clarified.
NA	I-37	Bits 6 - 31 bracketed.
NA	I-43	Changed wording for bit 19 of RM status register.

EDCP	PAGE	DESCRIPTION OF CHANGE
NA	II-iii	Typographical error corrected. Spelling of "general" corrected.
NA	II-13	Removed reference to design workbook.
NA	II-101	Added Appendix number to page numbers in MSC Instruction Summary Chart.
NA	II-102	Added Appendix number to page numbers in MSC Instruction Summary Chart.
NA	III-4	"Power surge" changed to "power".
NA	III-8	Corrected typographical error.
NA	III-11	Added note regarding direct addressing mnemonics.
NA	III-14	Text aligned.
NA	III-15	Changed page spacing.
NA	III-20	Changed wording in section defining signals that set BCE Halt bit to 0.
NA	III-25	Eliminated termination latch from Error Summary Chart.
NA	III-26	"Invalid Manchester, or bit count error" added under the parity error on input data.
NA	III-27	The "wrong bit encoding bit count error" section deleted.
NA	III-43	Added note for MIA Busy.
NA	III-48	Typographical error corrected. Last paragraph corrected.
NA	III-50	Clarified descriptions for #CMD and #CMDI.
87-101S-051	III-52	Removed program note for microcode error (microcode corrected).
NA	III-58	Added note for MIA Busy.
87-101S-049	III-75	Removed description of microcode anomaly (microcode corrected).
87-101S-049	III-80	Note on limited assembler support for this OP code. Corrected typographical errors.

EDCP	PAGE	DESCRIPTION OF CHANGE
NA	III-88	Added Appendix number to page numbers in BCE Instruction Summary Chart.
NA	III-97	Restored paragraph inadvertently deleted.

EDCP	PAGE	DESCRIPTION OF CHANGE
NA	2-21	Remove statement "Maskable Only in Problem State, PSW 47=1" from Interrupt priority C2. Add an X in Not Maskable column for Store Protect Interrupt.
NA	8-22	Correct condition code for Load Complement (Short Operands) for positive results.
NA	II-24, 25	Remove "Terminal Control plus" from last IOP error description on page II-24. Move paragraph extension on top of II-25 back to II-24.
NA	III-8	Change I to M for Short Format 1 Index Specification.
NA	III-15	Remove duplicated sentence under bit 23 description.
NA	III-48	Add an "or" between #MIN and #MIN@.

EDCP	PAGE	DESCRIPTION OF CHANGE
NA	vi	Replaced "(This page intentionally left blank)" with "NOTE: Use of fields marked as reserved can result in unpredictable machine operation."
NA	2-14	Replaced "1" in bit position 0 of Fig. 2-17 with MSB and added explanation.
NA	2-15	Deleted page and replaced with flow chart.
NA	2-21	Added Anomaly Notes to Fig. 2-20.
NA	2-23	Clarification of "Reserved" locations in Fig. 2-21.
NA	8-12	Deleted statement in description.
NA	8-12a	Added Anomaly Note for CEDR/CED instruction.
NA	8-12b	New Page: Added "(This page intentionally left blank)".
NA	8-17	Added Anomaly Note for DEDR/DED instruction.
NA	9-8	Revised statement on MOVE HALFWORD interruptibility.
NA	9-8a	New Page: Added Anomaly Note for MOVE HALFWORD instruction.
NA	9-8b	New Page: Added "(This page intentionally left blank)".
NA	9-19	Corrected instruction bit 31 designation.
NA	15-32	Changed "FOV Fail" to "YOV Fail" for Error Code 5A description.
NA	17-3	Corrected instruction execution time calculation for MVH RR (COUNT ODD).
NA	I-36	Changed "Dev out data error" to "Dev out data parity error" under ERROR CONDITION.
NA	III-14	Corrected bit 26 designation for TO.
NA	III-59	Deleted first paragraph at top of page.

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

NOTE: Use of fields marked as reserved can result in unpredictable machine operation.

1.0 AP-101S WITH SHUTTLE INSTRUCTION SET

The AP-101S is a high-speed, general-purpose computer intended primarily for real-time applications such as guidance, navigation, control, and data processing. The AP-101S is software compatible with the AP-101C/M, described in IBM No. 6246156B, 30 January 1979. This family shares, and is unified by, extensive design experience, proven technology base, and common manufacturing processes.

This Principles of Operation manual provides a direct comprehensive description of the CPU system structure; the arithmetic, logical, branching, and status switching; and the interruption system. This publication defines and describes features common to all AP-101S CPUs including the ground version, the AP-101S/G computers that do not contain an IOP.

Both computers contain a pipeline architecture CPU, and techniques for efficiently programming it are contained in Section 16 of this document.

(This page intentionally left blank)

2.0 AP-101S STRUCTURE

2.1 SHUTTLE INSTRUCTION SET

The AP-101S system structure encompasses the functional operation of main storage, the central processing unit (CPU), and program-controlled I/O facilities.

2.1.1 Information Formats

The system transmits information between main storage and the CPU in units of 16 bits, or in integer multiple of 16 bits. Each 16-bit unit of information is called a halfword. Six error correction bits and three voted storage protection bits are also associated with each halfword for the AP-101S but later references in this manual to the size of data fields exclude these bits. The AP-101S/G has two storage protect bits per halfword.

Halfwords may be handled separately or in pairs. A fullword is a group of two consecutive halfwords. Both halfword and fullword instructions and operands are used. Their location is always specified by the address of the leftmost halfword (leftmost halfword is the numerically smallest address). The instruction length is designated implicitly in every instruction; the operand length is also implicit.

Within any instruction and operand format, the bits making up the format are consecutively numbered from left to right, starting with the number 0, as shown in Figure 2-1.

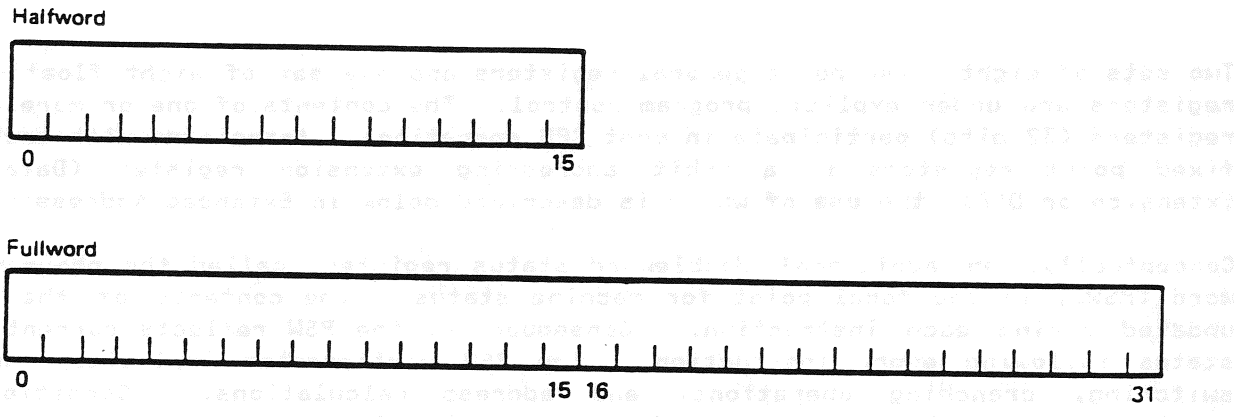


Figure 2-1. Instruction and Operand Bit Numbering

2.1.2 Addressing

Halfword locations in storage are consecutively numbered starting with 0. Each number is considered the address of the corresponding halfword. The addressing technique uses a 19-bit binary address to a maximum of 2^{19} halfword addresses. This set of main storage addresses includes some locations reserved for special purposes, such as program status words; consequently, these special locations should not be used for any purpose not implicitly defined.

2.1.3 Information Positioning

Unlike previous versions of the AP-101 computer, the AP-101S does not require either fullword instructions or fullword/doubleword operands to be located in main storage on even boundaries.

2.2 CENTRAL PROCESSING UNIT

The central processing unit (CPU) contains facilities for addressing main storage, for fetching or storing information, for arithmetic and logical processing of data, for sequencing instructions in the desired order, and for initiating the communication between storage and external devices.

The control section guides the CPU through the functions necessary to execute the program.

2.2.1 Program Addressable Registers

Two sets of eight fixed point general registers and one set of eight floating point registers are under explicit program control. The contents of one or more of these registers (32 bits) participate in most CPU operations. Associated with each of the fixed point registers is a 4-bit addressing extension register (Data Sector Extension or DSE), the use of which is described below in Extended Addressing.

Conceptually, an additional doubleword status register, called the program status word (PSW), is the focal point for machine status. The contents of the PSW are updated during each instruction. Consequently, the PSW reflects current machine status following every instruction. The PSW participates implicitly in status switching, branching operations, and address calculations. Condition codes resulting from an instruction are also part of the PSW.

In addition to the PSW and the general and floating point registers, the CPU also contains working registers used for storage addressing, storage buffering, shift and iteration counting, and operand storage. These registers are of no direct concern to the programmer and are not described herein.

The contents of the PSW specify which of the two sets of general registers is in current use. Only the contents of the selected general register set can participate in arithmetic operations and the contents of unselected sets of general registers cannot be altered by a program. An alternate set of general registers can be selected by changing the PSW. Only one set of the fixed point, general-purpose registers and the floating point registers are available to the program at any one time.

General register contents can be used interchangeably as operands for arithmetic, logical, and shifting operations, or as base and index registers for relative addressing. Each set of general registers is numbered from 0 through 7 and is addressed as shown in Figure 2-2.

General Register Number	Register Function		
	Operand	Base	Index
0	000	00	None
1	001	01	001
2	010	10	010
3	011	11 or None*	011
4	100		100
5	101		101
6	110		110
7	111		111

*11 = Register 3 for SRS; none for RS

Figure 2-2. General Register Addresses

Note that general registers 4 through 7 cannot contain base addresses and that general register 0 cannot contain an index.

For addressing data, general registers 0-3 can be augmented by 4-bit Data Sector Extension (DSE) registers or by the DSR in the PSW to address beyond 16-bit capabilities. There are 16 DSEs, one for each of the eight general-purpose registers in each of the two sets of general registers.

For some operations, a pair of general registers is linked to form a 64-bit doubleword register. The most significant half of a doubleword operand is contained in the specified register; the least significant half of the doubleword is in the next higher-numbered register (determined by Modulo 8 addition of one (1) to the specified register). Note: If Reg 7 is specified, the least significant half of the double word operand is contained in Reg. 0.

2.2.2 Fixed Point Data Representation

Data representation is fractional, with negative numbers represented in twos complement form. A halfword operand is 15 bits plus sign, a fullword operand is 31 bits plus sign, and a doubleword operand is 63 bits plus sign, as shown in Figure 2-3.

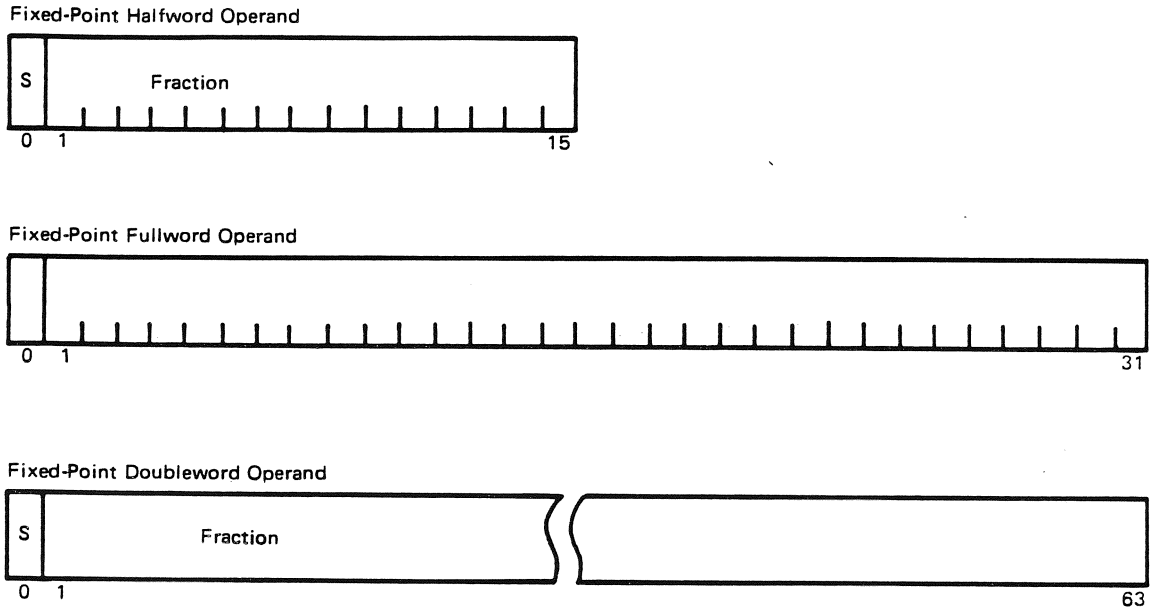


Figure 2-3. Fixed Point Operand Formats

In fractional data representation, the binary point is immediately to the right of the sign.

2.2.3 Instruction Formats

The length of an instruction format can be either one or two halfwords. Long format instructions provide maximum range and extended flexibility for addressing storage operands. Short instructions are used to (1) specify register-to-register operations, and (2) specify storage operands in cases where a small displacement is sufficient and complete address modification capability is not required.

Instruction formats overlap. Programs are written so that, in many instances, any given operation can be coded using either a halfword or a fullword instruction. In such cases, maximum use of halfword instructions results in increased storage efficiency and performance.

The three basic instruction formats are as shown in Figure 2-4. Halfword instructions are automatically selected by the assembler unless otherwise specified by the programmer.

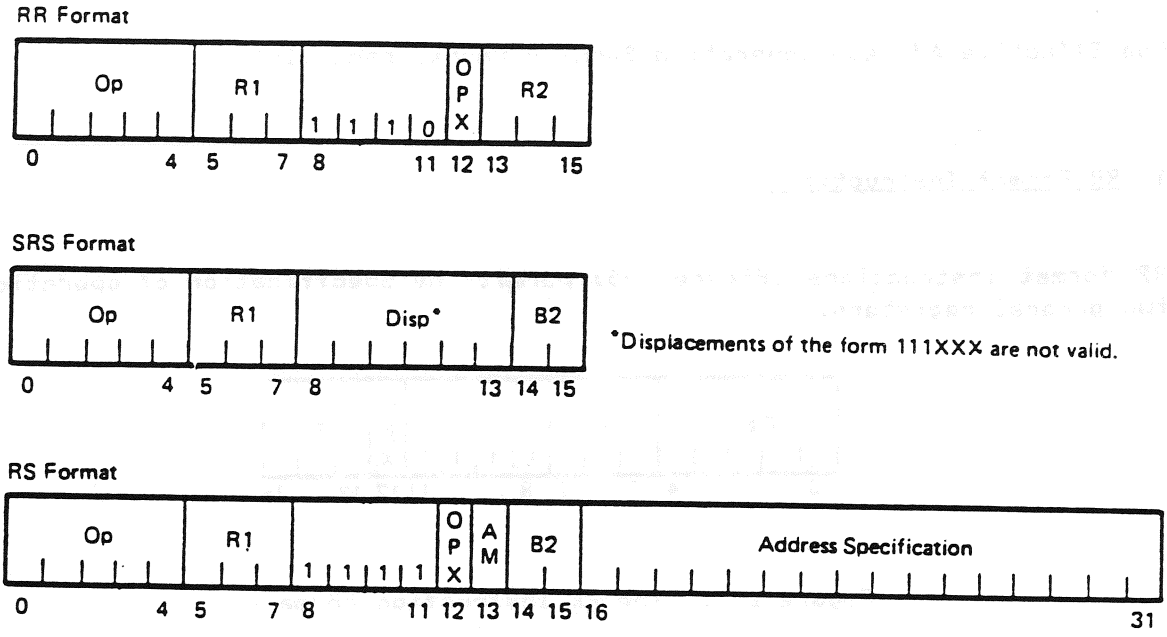


Figure 2-4. Basic Instruction Formats

The fields within the instruction formats usually are used as described below. The exceptions are described in conjunction with the individual formats and instructions.

Op This 5-bit field defines an operation, or the class of operation, to be performed by the CPU.

R1 This 3-bit field designates the register containing the first operand. Except for operations which alter main storage, the result usually replaces the first operand.

R2 This 3-bit field appears only in the RR format. It is used to specify a general register containing either the second operand or the address of the second operand.

B2 This 2-bit field specifies the register containing the base address.

Disp In halfword SRS format instructions, this 6-bit field is called the displacement. For the SRS format, the displacement is added to the base address specified by the B field to obtain a storage address.

OPX This bit is an extension of the OP field.

AM This field designates one of two fullword format addressing options.

Address Specification	The second halfword of a fullword instruction is specified as either extended or indexed addressing.

See the Effective Address Generation Summary Chart, page 11-1.

2.2.4 RR Format Instructions

The RR format instructions (Figure 2-5) permit the specification of operations that use two general registers.

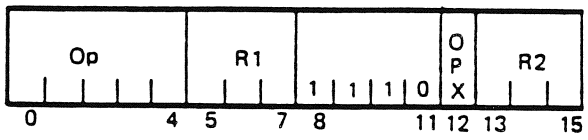
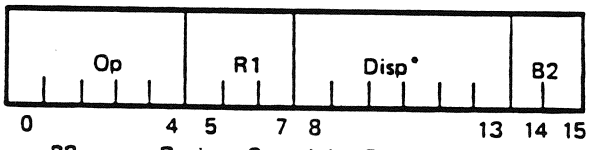


Figure 2-5. The RR Instruction Formats

The operation normally uses as operands the contents of two general registers. The R2 field specifies the second operand while the R1 specifies the first operand. The result of the operation usually replaces the first operand.

2.2.5 SRS Format Instructions

The SRS instruction format (Figure 2-6) is a compression of the RS format. It provides base plus displacement storage addressing.



* Displacements of the form 111XXX are not valid.

B2	Register Containing Base
00	General Register 0
01	General Register 1
10	General Register 2
11	General Register 3

Figure 2-6. SRS Instruction Format

The R1 field specifies the first operand register address. The 19-bit effective address (EA) of the second operand is developed as follows:

Step 1 First the positive integer contained in the displacement field is added to the contents of the base contained in the general register specified by B2.

When addressing halfword operands, the least significant bit of the displacement field (instruction bit 13) is aligned with base register bit 15. The 16-bit result is the sum of the base and the displacement, aligned as shown in Figure 2-7.

When addressing fullword operands using the SRS format, the least significant bit of the displacement field is aligned with base register bit 14 as shown in Figure 2-8.

Unlike previous versions of this architecture, bit 15 of a base register is significant when addressing fullword data. Fullword storage operands may now be located on odd address boundaries. Programs which utilize this feature will not be downward compatible.

Step 2 The 16-bit result of the addition of the base and displacement is expanded (see Expanded Addressing) to a 19-bit effective address (EA), and this is the address of the second operand.

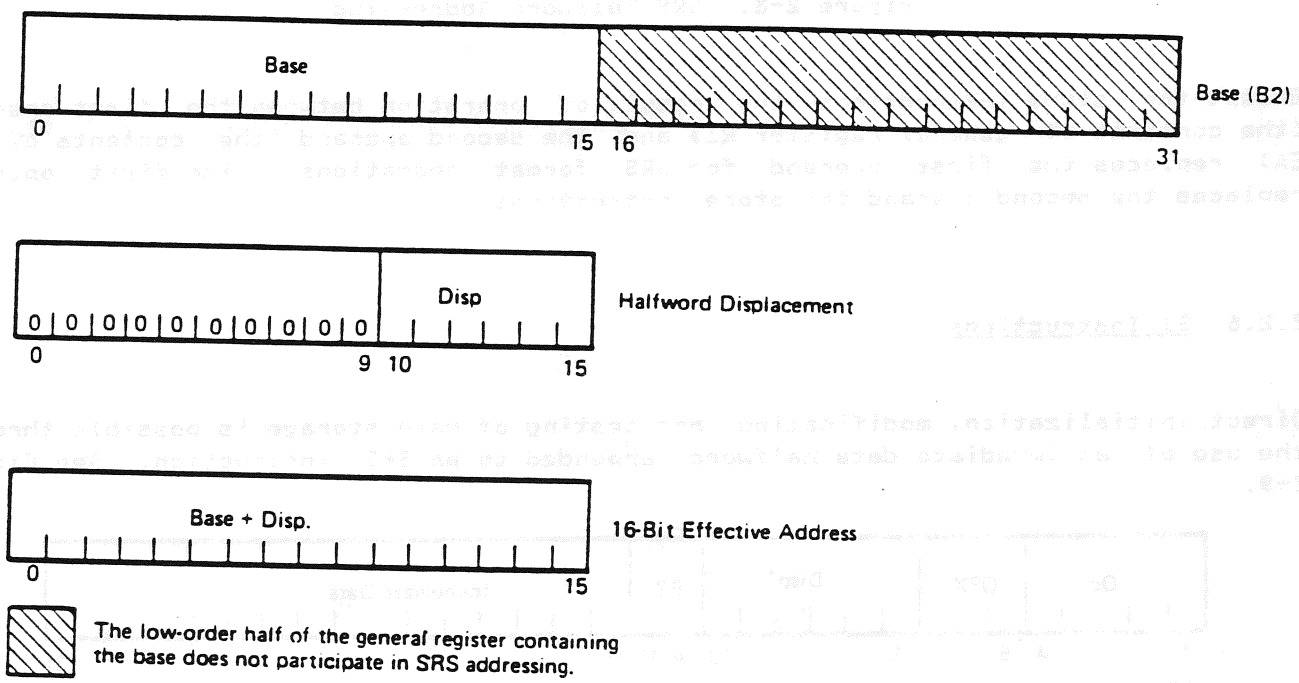


Figure 2-7. SRS Halfword Addressing

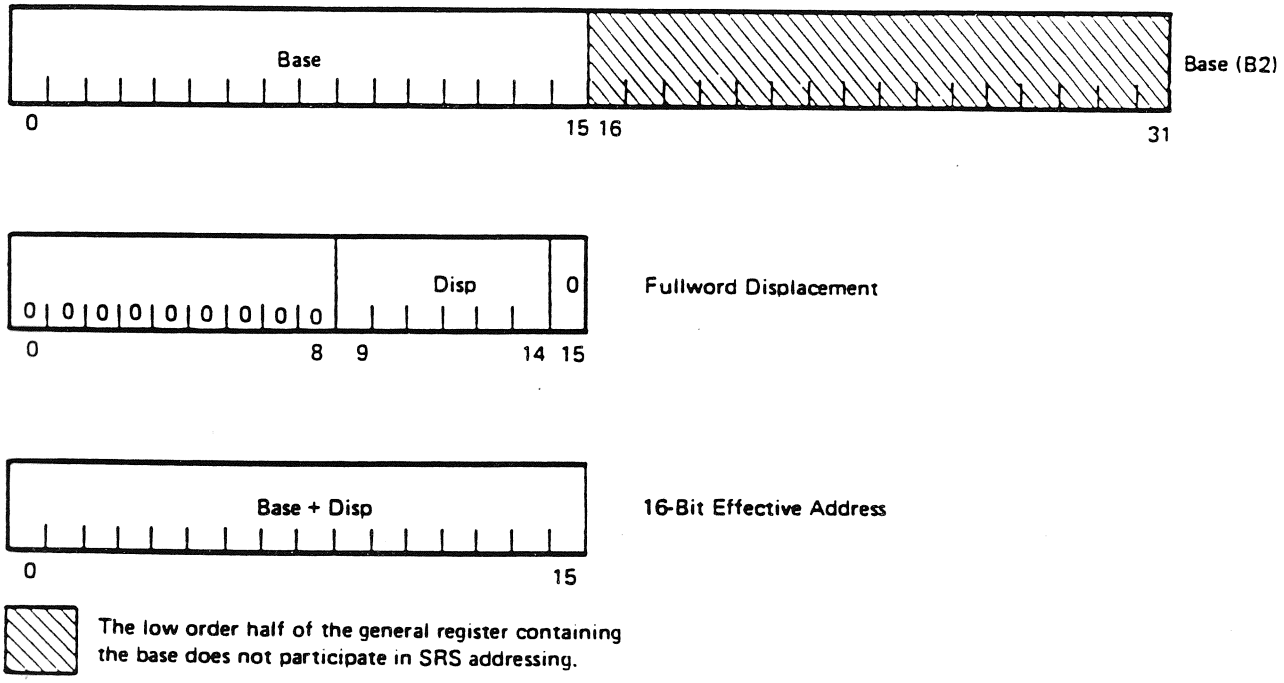


Figure 2-8. SRS Fullword Addressing

Except for store instructions, the result of operation between the first operand (the contents of general register R1) and the second operand (the contents of the EA) replaces the first operand for SRS format operations. The first operand replaces the second operand for store instructions.

2.2.6 SI Instructions

Direct initialization, modification, and testing of main storage is possible through the use of an immediate data halfword appended to an SRS instruction. See Figure 2-9.

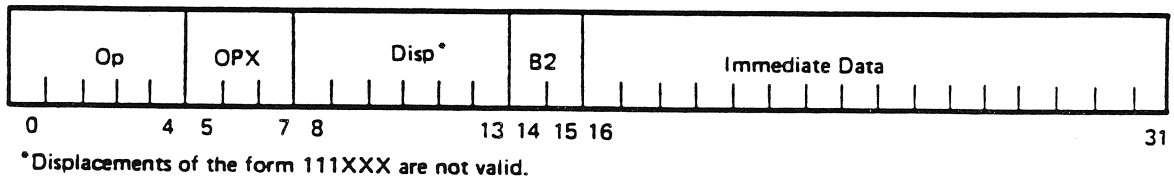


Figure 2-9. SI Instructions

The address of the halfword second operand is developed in the normal manner for SRS instructions using halfword addressing. Except for test instructions, the result of the operation between the halfword second operand and the immediate data replaces

the second operand. The second operand is not altered for test instructions. The first operand is never altered for SI instructions.

2.2.7 RI Instructions

Using an immediate data halfword appended to an RR instruction (Figure 2-10) permits direct initialization, modification, and testing of the most significant 16 bits contained in a general register.

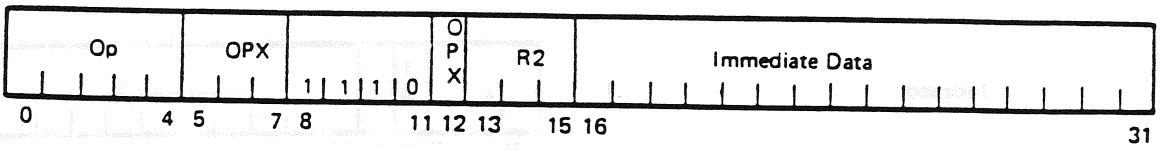


Figure 2-10. RI Instructions

Except for test instructions, the result of the operation between the second operand and the immediate data replaces the second operand. The second operand is not altered for test instructions. The immediate data first operand is never altered for RI instructions.

2.2.8 RS Format Instructions

There are two major classes of RS instructions, extended and indexed addressing modes, differing in the techniques used to specify the second operand. See Figure 2-11.

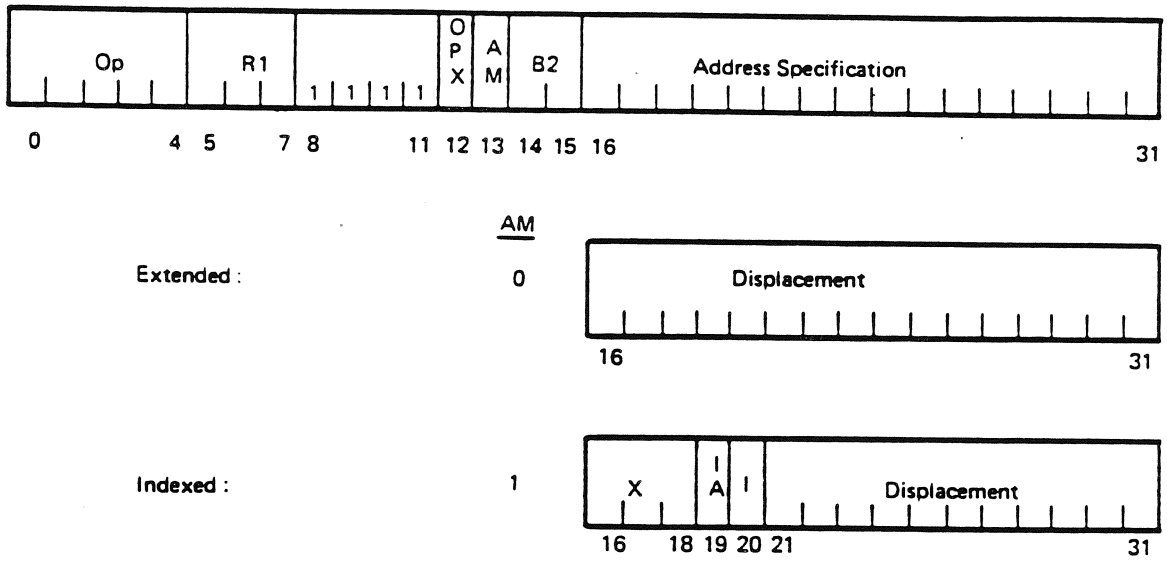


Figure 2-11. RS Instruction Formats

Extended addressing is specified when RS format bit 13 (AM) equals 0. This addressing mode provides a full 16-bit halfword displacement. The base and displacement are aligned as shown in Figure 2-12 when base addressing is performed.

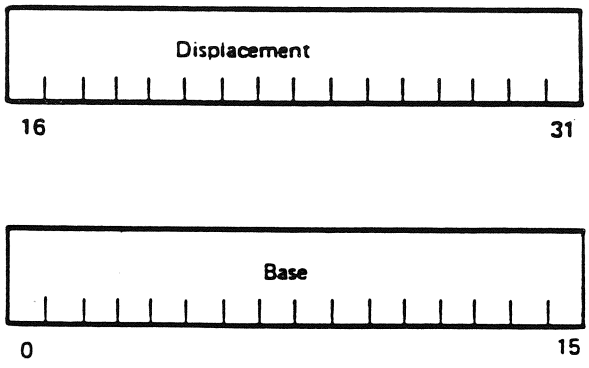


Figure 2-12. Displacement Alignment for Extended Addressing

Aside from the size and alignment of the displacement, RS extended addressing differs from SRS addressing in two other respects:

1. The alignment of the displacement is the same whether addressing doubleword, fullword or halfword operands.

2. When B2 equals 11, base addressing is not performed. In this case, the displacement is instead used directly as the effective address.

Indexed addressing is specified by RS format bit 13 (AM) equal to 1. This addressing mode contains three additional fields. Normally, they contribute to the effective address generation as follows:

X — This 3-bit field specifies one of seven general registers containing the index. Indexing is not performed when X is equal to 000. An index is contained in the upper halfword of a general register. The index is automatically aligned as illustrated in Figure 2-13. For additional information on index alignment, see Section 14. Consistent with the restrictions that apply to register usage and indirect addressing, general register contents can be used interchangeably as either a base or an index or both. When indirect addressing is specified, indexing follows indirect addressing (postindexing).

IA — This format bit, when a one, specifies indirect addressing. Indirect addressing is not performed when this bit is zero. In the instruction descriptions, the symbol @ denotes IA for the assembler.

I — This format bit, in conjunction with X and IA, specifies various address modes which are explained below. In the instruction descriptions, the symbol # denotes I for the assembler.

The development of the EA for the indexed mode (including IC relative) of operand addressing is explained in detail in the subsequent steps:

1. Indexed addressing is specified by RS format bit 13 (AM) equal to 1. This addressing mode provides an 11-bit displacement. The base and displacement are aligned as shown in Figure 2-14 when indexed addressing is performed.

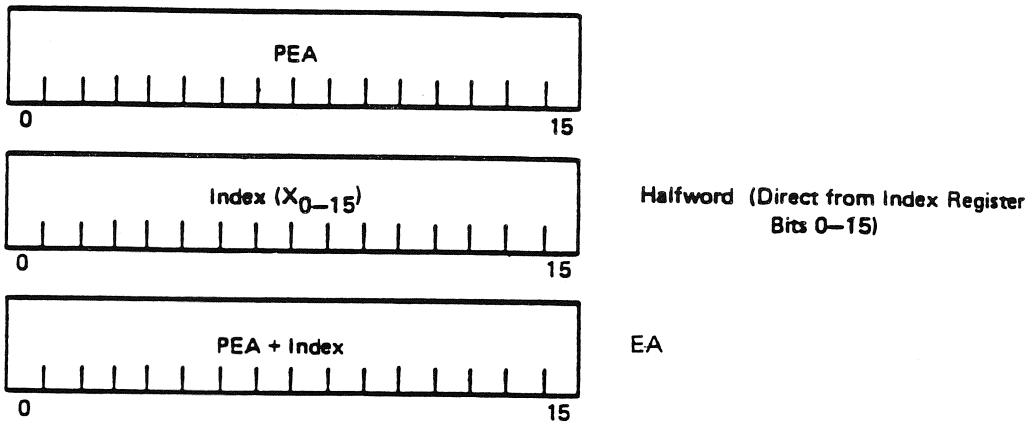
   The displacement is aligned so that bit 31 corresponds to base or index bit 15 and displacement bit 21 corresponds to base or index bit 5. The displacement is expanded to 16 bits by appending five leading zeros.

2. If B2 is not equal to 11, the 16-bit base, contained in the higher order half of the specified register, is added to the aligned displacement. This results in a preliminary effective address (PEA) whereby the PEA = (B) + Displacement.

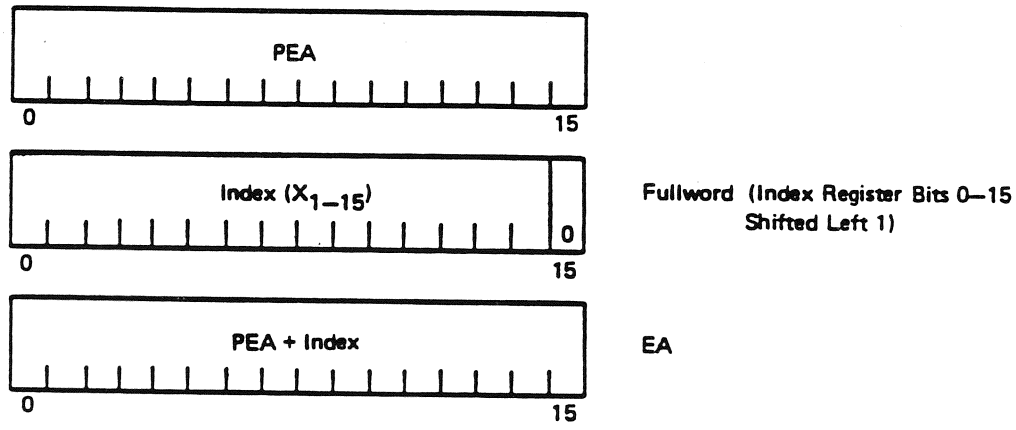
   If B2 is equal to 11, the aligned displacement is added to zero. This result is the preliminary effective address (PEA) whereby the PEA=Displacement.

3. If the X field is all zeros, IA (bit 19) is a zero and I (bit 20) is a zero, then the 16-bit result of Step 2 is added to the contents of the updated instruction counter (IC) to form the 16-bit EA whereby EA=(updated) IC + PEA. (This EA is then expanded to a 19-bit EA, as explained in the Expanded Addressing section, with the exception that the Branch Sector Register (BSR) bits are used instead of the Data Sector Register (DSR bits).

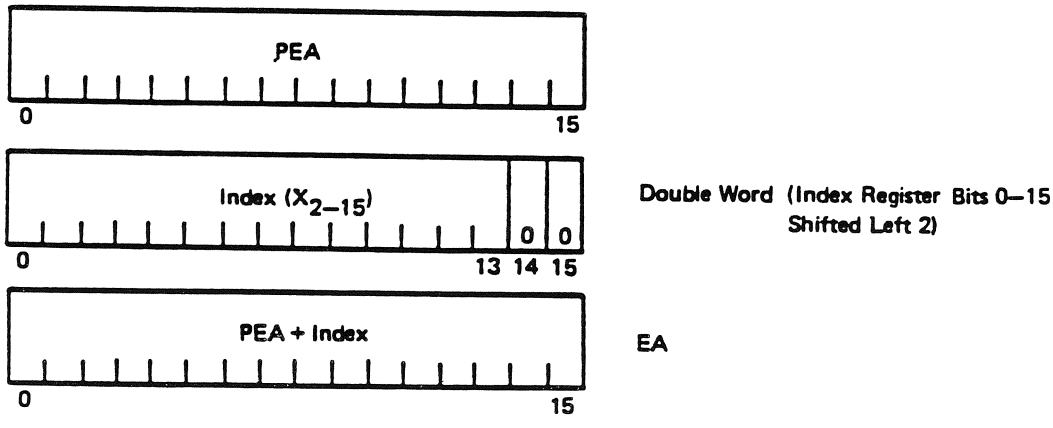
4. If the X field is all zeros, IA (bit 19) is a zero and I (bit 20) is a one, the 16-bit result of Step 2 is subtracted from the contents of the updated IC to form the 16-bit EA whereby EA = (updated) IC - PEA. (This



a. Halfword Index Alignment



b. Fullword Index Alignment



c. Double Word Index Alignment

Figure 2-13. Automatic Index Alignment

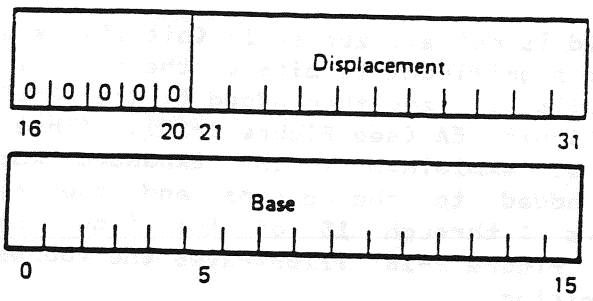


Figure 2-14. Displacement Alignment for Indexed Addressing

EA is then expanded to a 19-bit EA, as explained in the Expanded Addressing section with the exception that the Branch Sector Register (BSR) bits are used instead of the Data Sector Register (DSR) bits.)

5. If the X field is all zeros, IA (bit 19) is a one and I (bit 20) is a zero, then Indirect Addressing is performed. The 16-bit result of Step 2 is expanded to a 19-bit address and is used as the address of a main storage halfword. This halfword is then fetched and expanded to 19 bits by using expanded addressing to form the EA. EA=MS (PEA). Functional equivalency to preindexing capability can be obtained through modification of the base.

6. If the X field is all zeros, IA (bit 19) is a one and I (bit 20) is a one, Indirect Addressing is performed as described in Step 5 with a fullword main storage pointer. Then, after the EA has been formed, storage modification is automatically performed. The indirect address is contained in a fullword. A modifier is contained in bits 16 through 31. An address is contained in bits 0 through 15. The modifier is added to the address and the resulting modified address replaces bits 0 through 15 of the indirect address word (see Figure 2-15).

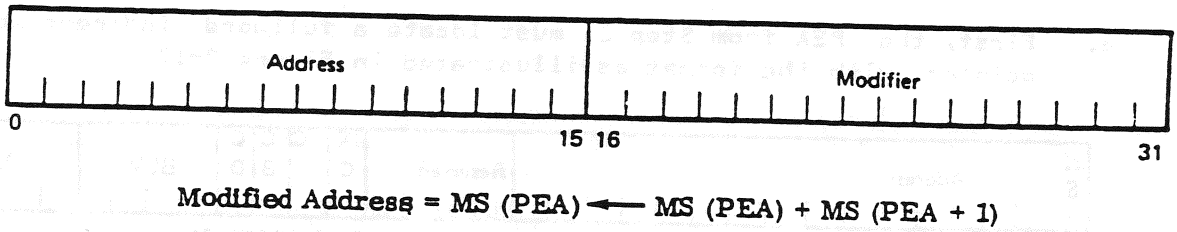
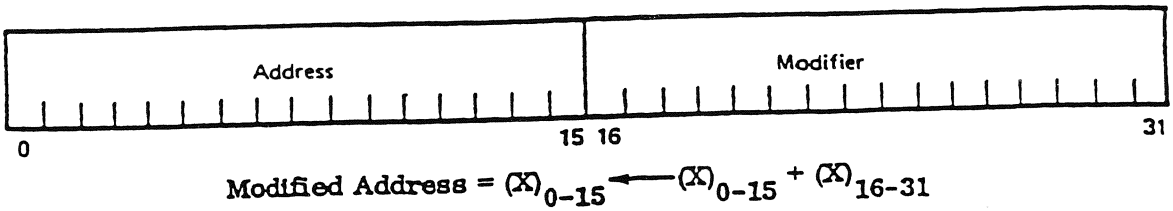


Figure 2-15. The Contents of Indirect Address Storage Modification Word

7. If the X field is not zeros, IA (bit 19) is a zero and I (bit 20) is a zero, the most significant 16 bits of the general register specified by the X field are aligned, and then added to the 16-bit result of Step 2 (PEA) to form the 16-bit EA (See Figure 2-13). (This EA is then expanded to a 19-bit EA, as explained in the Expanded Addressing section.)

8. If the X field is not all zeros, IA (bit 19) is a zero and I (bit 20) is a one, the most significant 16 bits of the general register specified by the X field are aligned, and then added to the 16-bit result of Step 2 (PEA) to form the 16-bit EA (see Figure 2-13). (This EA is then expanded to a 19-bit EA, as explained in the Expanded Addressing section.) (The modifier is added to the address and the resulting modified address replaces bits 0 through 15 of the index register after the EA is determined.) Figure 2-16 illustrates the address and modifier format in the index register.



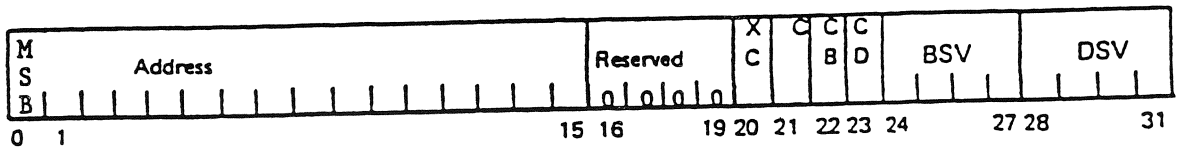
$$\text{Modified Address} = (X)_{0-15} \leftarrow (X)_{0-15} + (X)_{16-31}$$

Figure 2-16. The Contents of Index Register X

9. If the X field is not all zeros, IA (bit 19) is a one and I (bit 20) is a zero, Indirect Addressing (IA) with postindexing is performed. The 16-bit result of Step 2 is expanded to a 19-bit address and is used to fetch a main storage halfword. The index contained in the general register specified by X is aligned and then added to the fetched halfword to form the 16-bit EA (see Figure 2-13). This EA is then expanded to a 19-bit EA by using expanded addressing. Functional equivalency to preindexing capability can be obtained through modification of the base.

10. If the X field is not all zeros, IA (bit 19) is a one and I (bit 20) is a one, an indirect addressing mode is defined using a 32-bit fullword indirect address pointer as follows:

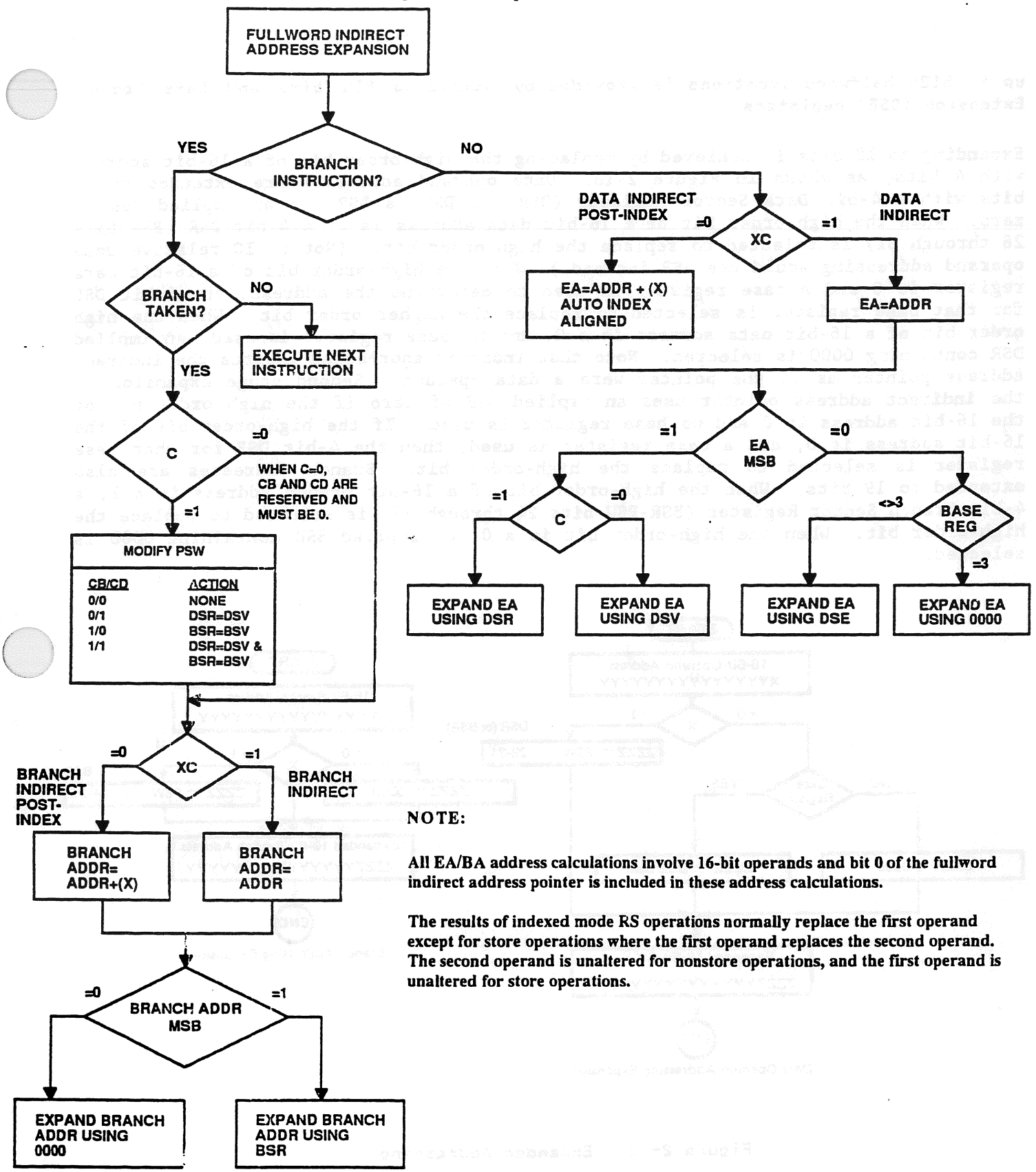
a. First, the PEA from Step 2 must locate a fullword indirect address pointer, with the format as illustrated in Figure 2-17.



Field	Function
X_C	Index Control
C	Control to allow PSW modification
C_B	Control BSV Usage
C_D	Control DSV Usage
BSV (Branch Sector Vector)	Selectively replaces BSR in PSW
DSV (Data Sector Vector)	Selectively replaces DSR in PSW
MSB (Most Significant Bit)	Determines type of address expansion

Figure 2-17. Fullword Indirect Address Pointer

b. Next the fullword indirect address pointer is expanded to a 19 bit address as follows:



NOTE:

All EA/BA address calculations involve 16-bit operands and bit 0 of the fullword indirect address pointer is included in these address calculations.

The results of indexed mode RS operations normally replace the first operand except for store operations where the first operand replaces the second operand. The second operand is unaltered for nonstore operations, and the first operand is unaltered for store operations.

2.9 Expanded Addressing

The addressing philosophy accommodates 64K halfword addresses since a full 16-bit address is provided. Extending the addressing range beyond 64K halfword locations

up to 512K halfword locations is provided by utilizing PSW bits and Data Sector Extension (DSE) registers.

Expanding to 19 bits is achieved by replacing the high-order bit of a 16-bit address with 4 bits, as shown in Figure 2-18. Data operand addresses are extended to 19 bits with a 4-bit Data Sector Register (DSR), a DSE, a BSR, or an implied DSR of zero. When the high-order bit of a 16-bit data address is 1, a 4-bit DSR (PSW bits 28 through 31) is selected to replace the high-order bit. (Note: IC relative data operand addressing would use BSR instead.) When the high-order bit of a 16-bit data register is 0 and a base register is used to determine the address, the 4-bit DSE for that base register is selected to replace the higher order bit. When the high order bit of a 16-bit data address is a 0, and no base register is used, an implied DSR containing 0000 is selected. Note that indirect addressing locates the indirect address pointer as if the pointer were a data operand. Second stage expansion of the indirect address pointer uses an implied DSR of zero if the high order bit of the 16-bit address is 0 and no base register is used. If the high-order bit of the 16-bit address is 0, and a base register is used, then the 4-bit DSE for that base register is selected to replace the high-order bit. Branch addresses are also extended to 19 bits. When the high-order bit of a 16-bit branch address is a 1, a 4-bit Branch Sector Register (BSR-PSW bits 24 through 27) is selected to replace the high-order bit. When the high-order bit is a 0, an implied BSR containing 0000 is selected.

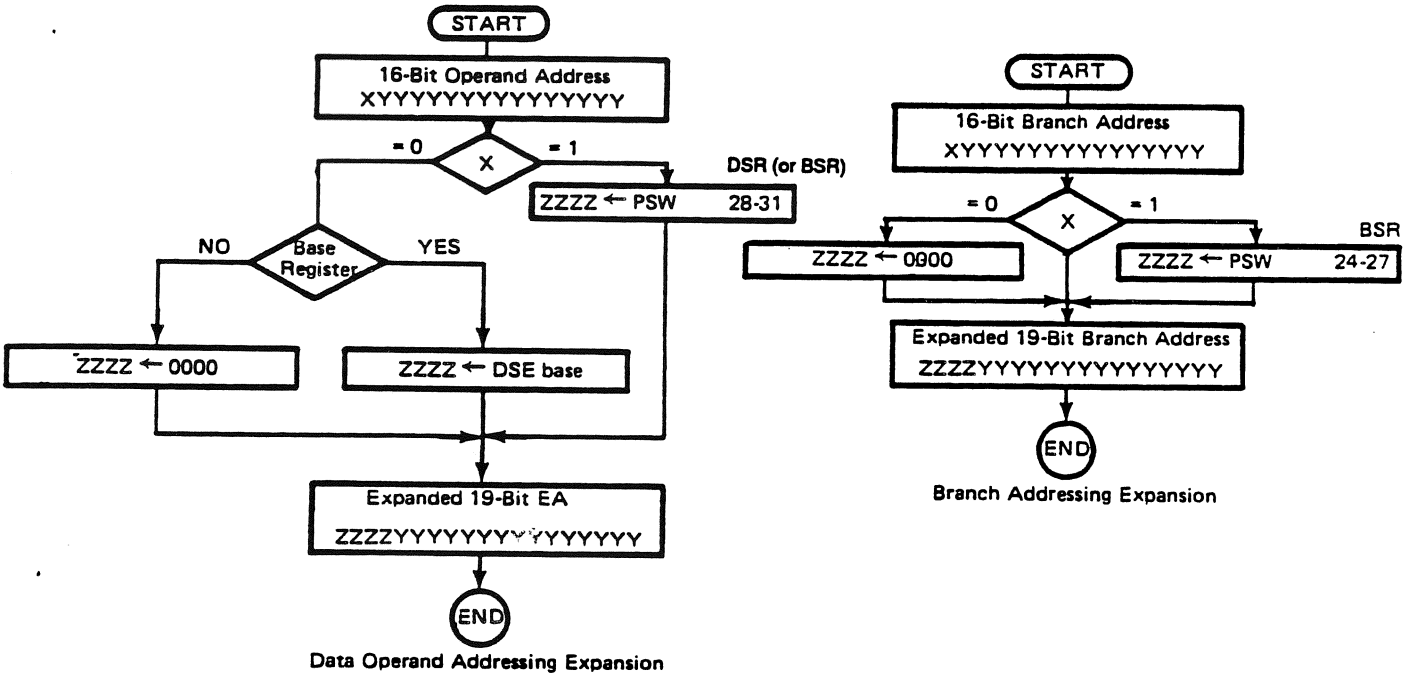
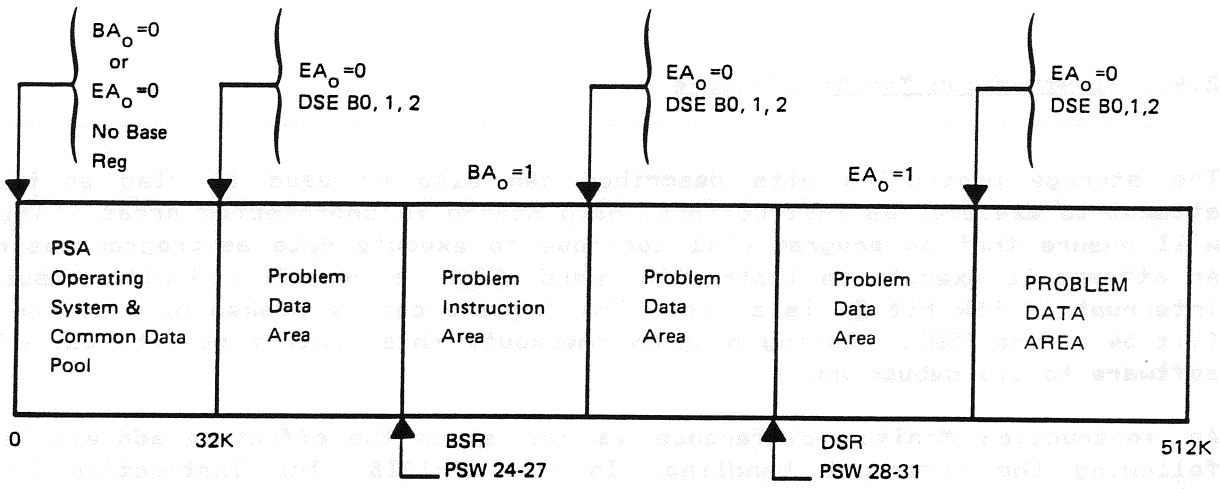


Figure 2-18. Expanded Addressing

Pictorially, main storage can be visualized as follows:



This permits efficient communication from the problem program to the operating system, the preferred storage area, (PSA) or a common data area.

It should be cautioned that instruction address incrementing or address calculations used to form the EA are performed on the low 16 bits only, and will not alter the BSR, DSR, or DSE. The BSR or DSR may be altered only via a PSW swap, special instruction operations (SVC, LPS) or by use of the indirect address pointer described in this section. The DSE registers are loaded by the LXA and LDM instructions.

2.3 PROGRAM EXECUTION

The CPU program consists of instruction and control words specifying the operations to be performed. This information resides in main storage and addressable registers and may be operated on as data. Instruction execution control is as defined under the section on Machine Status and General System Operation. Insert Storage Protect Bits, Load Program Status, Internal Control and Set System Mask instructions are privileged instructions and can only be executed in the Supervisor State. The Program Status Word determines the current state of the CPU and the Supervisor Call instruction can be used by the problem program to enter Supervisor State.

2.4 STORAGE PROTECTION FEATURES

The storage protection feature prevents modification of specific main storage locations. Any location which could, for example, contain constant data or program instructions can be selectively protected from Store operations without restricting the use of other areas. Traps on store operations to specific data words can be inserted during program checkout. A privileged instruction, Insert Storage Protect Bits, is provided to set/reset the protection bits associated with each halfword of

main storage. Attempting to store data in a protected location will result in a program interrupt. In this case, the store operation does not occur.

2.4.1 Instruction Monitor Feature

The storage protection bits described can also be used to flag an inadvertent attempt to execute, as instructions, data stored in unprotected areas. The feature will ensure that no program will continue to execute data as program instructions. An attempt to execute an instruction word which is unprotected will result in an interrupt if PSW bit 34 is a one. The feature can be masked by a System Mask Bit (bit 34 of the PSW). During program checkout, this feature permits use of special software to aid debugging.

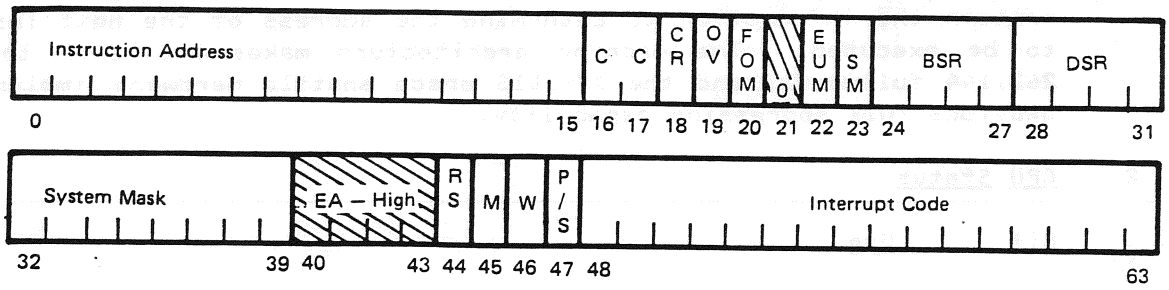
An Instruction Monitor difference is the state the effective address is left in following the interrupt handling. In the AP-101B, the Instruction Counter is incremented to point to the next instruction to be executed. The AP-101S Instruction Counter is not incremented and is left pointing to the offending instruction.

2.5 MACHINE STATUS

System status can be altered by the occurrence of interrupts and by the program. A doubleword register within the CPU contains a program status word (PSW) and is the focal point for CPU and system status conditions.

2.5.1 Program Status Word

The program status word (PSW), contains the basic information required for proper program execution. The 64-bit PSW includes the next instruction address, the current condition code, the carry and overflow indicators, the system mask for interrupts, and other fields significant to CPU operations. 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 use. By loading a new PSW or part of a PSW, the state of the CPU can be initialized or changed. Figure 2-19 shows the PSW format.



0-15	Next Instruction Address	
16-17	Condition Code	
18	Carry Indicator	
19	Overflow Indicator	
20	Fixed-Point Arithmetic Overflow Mask*	
21	Reserved	
22	Floating Point Exponent Underflow Mask*	
23	Significance Mask*	
24-27	Branch Sector Register	
28-31	Data Sector Register	
32	Counter 1 Mask	System* Mask
33	Counter 2 Mask	
34	Instruction Monitor Mask	
35	External Interrupt 0 Mask	
36	External Interrupt 1 Mask	System* Mask
37	External Interrupt 2 Mask	
38	External Interrupt 3 Mask	
39	External Interrupt 4 Mask	
40-43	Reserved for SVC High Order EA Bits	
44	Register Set (GR set 0 or 1)	
45	Machine Check Mask*	
46	Wait State Bit (Wait/Process)***	
47	Problem/Supervisor State Control Bit**	
48-63	Interrupt Code for Program Check, Machine Check, and Special External Interrupts, or 16 Bit Operand PEA for SVC Instruction	

*Mask bit = 0, interrupt inhibited
= 1, interrupt allowed
**0 = supervisor state
1 = problem state
***0 = process state
1 = wait state

Figure 2-19. PSW Fields

The overall status of the CPU is preserved in the current PSW and the contents of the general registers. The PSW is automatically retained upon taking an interrupt. It is the programmer's responsibility to preserve the contents of the general registers when necessary.

Certain other conditions that contribute to an overall system status situation are not automatically preserved when a CPU is interrupted. There conditions involve additional units and include the dynamic state of all other interrupts, the state of real time counters, and I/O system status.

Masking is accomplished by setting the appropriate PSW bit to zero.

2.5.1.1 PSW Fields

The PSW fields (Figure 2-19) are defined as follows:

1. Instruction Address - Bits 0 through 15 and 24 through 27 of the PSW contain the information to determine the address of the next instruction to be executed. The machine architecture makes provision to address 262,144 fullwords, and the AP-101S space shuttle hardware implementation provides full addressing capability.

2. CPU Status

Bit	Use
16, 17	Condition code for certain arithmetic, logical and I/O instructions
18	Carry status bit indicator
19	Overflow status bit indicator (overflow can be reset by testing or by loading the PSW)
20	Fixed Point Arithmetic Overflow Mask
21	Reserved
22	Floating Point Exponent Underflow Mask
23	Significance Mask

3. Branch Sector Register - Bits 24 through 27 replace the high-order bit of a branch address when that bit is a 1. Otherwise, an implied sector register of 0000 replaces the high-order bit.

4. Data Sector Register - Bits 28 through 31 replace the high-order bit of a data address when that bit is a 1. See "Expanded Addressing" for details when bit 0 is a zero.

5. System Mask - Bits 32 through 39 are mask bits. The first two bits of the System Mask are normally assigned to the two counters and the third to the instruction Monitor Feature. The remaining five masks include I/O end conditions, other application dependent items such as a manual interrupt key, and timer overflow conditions. The instruction SET SYSTEM MASK is provided for modifying this field.

6. EA-High - For an SVC instruction, the 4-bit extension to make the 19-bit effective address is saved in the old PSW bits 40-43.

7. Register Select Field - The register select field, bit 44, controls either of two sets of general registers in current use. When this bit is a zero, then register set 0 is used; when this bit is one, then register set 1 is used. The set of general registers in current use can be selected when a new PSW is loaded. This can result from the execution of the PSW load instruction or from an interrupt.

8. Machine Check Mask - Bit 45 is the mask bit which is used to inhibit machine check interrupts (see Figure 2-20). When this bit is a zero, then machine check interrupts detected by the CPU are inhibited.

Interrupt Priority	Class	Old PSW	New PSW	Not Maskable	PSW Mask Bit	Pending	Int. Code	Interrupt Accept Time	CPU/IOP/AGE Generated	Interrupt
00	Power	0010	—	X	—	—	N/A	ENDOP	CPU	Power Off ****** (Microcode Put Away)
01	Power	—	0004	X *	—	—	N/A	MCYCLE	CPU	Power On
02	Power	—	0014	X **	—	—	N/A	MCYCLE	CPU	System Reset
03	Power	—	—	—	—	—	N/A	—	—	N/A to Shuttle ISA
C0	MC	0040***#	0044	—	45	No	0008	MCYCLE	CPU	EA Fault
04	MC	0040***#	0044	—	45	No	0005	MCYCLE	CPU	CPU Microstore Parity
05	MC	0040#	0044	—	45	No	0006	ENDOP	CPU	Interrupt Page Fault
35	MC	0040#	0044	—	45	No	0002	Forced ENDOP	IOP	DMA Memory Multi-bit Error
06	MC	0040†#	0044	—	45	No	0003	Forced ENDOP	CPU	CPU Memory Multi-bit Error
10	MC	—	—	—	—	—	—	—	—	Spare
11	MC	—	—	—	—	—	—	—	—	Spare
12	MC	0040***#	0044	X	—	—	0007	MCYCLE	CPU	ENDOP Timeout
13	MC	—	—	—	—	—	—	—	—	Spare
14	MC	0040***#	0044	X	—	No	0009	MCYCLE	CPU	CPU Cannot Continue
15	MC	—	—	—	—	—	—	—	—	Reserved
16	MC	—	—	X	—	—	—	ENDOP	AGE	AGE Breakpoint (Tester Service)
30	MC	—	—	—	—	—	—	—	—	N/A to Shuttle ISA
36	MC	—	—	—	—	—	—	—	—	IU Memory Error *****
37	MC	—	—	—	—	—	—	—	—	EU Memory Error *****
17	PE	0070	0074	—	34	No	N/A	ENDOP	CPU	CPU Breakpoint (Instruction Monitor)
20	PE	0048	004C	—	20	Note 1	0004	ENDOP	CPU	Fixed Point Overflow
21	PE	0048	004C	X	—	No	000B	Forced ENDOP	CPU	Floating Point Overflow (Exponent)
22	PE	0048	004C	—	22	No	0009	Forced ENDOP	CPU	Floating Point Underflow
23	PE	—	—	—	—	—	—	—	—	Spare
C1, 34	PE	0048	004C	X	—	No	0000	MCYCLE	CPU	Illegal Instruction, or I/O Command
C2	PE	0048	004C	X*****	—	No	0001	ENDOP	CPU	Privileged Instruction
C3	PE	0048	004C	X	—	No	000C	Forced ENDOP	CPU	Divided by Zero (Flt. Pt.)
C4	PE	0048	004C	—	23	No	0005	Forced ENDOP	CPU	Significance
C5	PE	0048	004C	X	—	No	000A	ENDOP	CPU	Convert Overflow
31	PE	0048	004C	X	—	No	0002	Forced ENDOP	CPU	CPU Addr Spec 128K, GB Only
P0	SC	0058	005C	X	—	No	(INST)	ENDOP	CPU	Supervisor Call
31	PE	—	—	—	—	—	—	—	—	Spare
32	PE	—	—	—	—	—	—	—	—	N/A to Shuttle ISA
33	PE	0048#	004C	X	—	—	0007	Forced ENDOP	CPU	Store Protect Violation
07	PE	—	—	—	—	—	—	—	—	N/A to Shuttle ISA
40-43	SYS	—	—	—	—	—	—	—	—	N/A to Shuttle ISA
44	SYS	—	—	—	—	—	—	—	—	Spare
45	SYS	0060	0064	—	32	Yes	—	ENDOP	CPU	Interval Timer No. 1
46	SYS	0068	006C	—	33	Yes	—	ENDOP	CPU	Interval Timer No. 2
47	SYS	—	—	—	—	—	—	—	—	N/A to Shuttle ISA
50	SYS	0078	007C	—	35	Yes	0000	ENDOP	IOP	External 0 (IOP Voter, IOP Reg. A)
50	SYS	0078	007C	—	35	Yes	0000	ENDOP	IOP	External 0 (C/M Idle, IOP Reg. A)
50	SYS	0078	007C	—	35	Yes	0000	ENDOP	IOP	External 0 (IOP ROS Parity, IOP Reg. A)
50	SYS	0078	007C	—	35	Yes	0000	ENDOP	IOP	External 0 (IOP Fault, IOP Reg. A)
50	SYS	0078	007C	—	35	Yes	0000	ENDOP	IOP	External 0 (Watchdog Timer, IOP Reg. A)
51	SYS	0080	0084	—	36	Yes	0000	ENDOP	IOP	Ext 1 IOP Data Flow Error Encoded (see Read Interrupt Reg. B in Appendix I)
51	SYS	0080	0084	—	36	Yes	0000	ENDOP	IOP	Ext 1 Q Overflow (IOP Reg. B)
51	SYS	0080	0084	—	36	Yes	0000	ENDOP	IOP	Ext 1 DMA Timeout (IOP Reg. B)
51	SYS	0080#	0084	—	36	Yes	0004	ENDOP	CPU	Ext 1 DMA Store Protect Violation##
53	SYS	0088	008C	—	37	Yes	—	ENDOP	IOP	Ext 2 IOP Programmed Interrupts (1-12)
54	SYS	0090	0094	—	38	Yes	—	ENDOP	IOP	Spare External 3
55	SYS	0098	009C	—	39	Yes	—	ENDOP	IOP	Spare External 4
56	SYS			—	—	—	—	—		Spare
52	SYS	0080	0084	—	36	Yes	0006	ENDOP	AGE	Shuttle AGE Interrupt

* CPU must not be in the halt mode

** CPU must be in halt mode

*** PSW can vary, maybe updated PC or unupdated PC

**** Only occurs when in problem state

***** Valid only during execution in Diagnose Instruction

****** If power off during long instruction, IC may be backed up

(INST) 16 Bit Operand PEA of SVC Instruction

Note 1 Status held active in PSW 19

† See note in Paragraph 2.5.2.1 on page 2-25.

ANOMALY: When one of these interrupts is taken, the condition code (CC) in the OLD PSW will be set to a binary 10 and the carry and overflow bits in the OLD PSW will be cleared.

ANOMALY: A masked DMA store protect interrupt will set the condition code (CC) to a binary 10 and clear the carry and overflow bits. This can result in erroneous GPC operation if an instruction tries to utilize the CC, carry bit or overflow bit before they are set by another instruction. Additionally, a masked DMA store protect interrupt clears any fixed point overflow, floating point underflow, and floating point overflow interrupts. This can result in a lost arithmetic interrupt if a masked DMA store protect interrupt occurs during an instruction that causes one of these arithmetic interrupts.

Figure 2-20. Interrupt Structure and Priority

9. Wait State - Bit 46 determines the wait or processing (run) states. When this bit is a zero, the CPU is in the processing state. When this bit is a one, the CPU is in the Wait State.

10. Problem/Supervisor - Bit 47 determines the problem or supervisor states. When this bit is a zero, the CPU is in the supervisor state and privileged instructions can be executed. When this bit is a one, the CPU is in the problem state and attempts to execute privileged instructions are inhibited resulting in an interrupt.

11. Bits 48 through 63 are reserved for the interrupt code. Program and machine check interrupt conditions and associated interrupt codes are given in Figure 2-20.

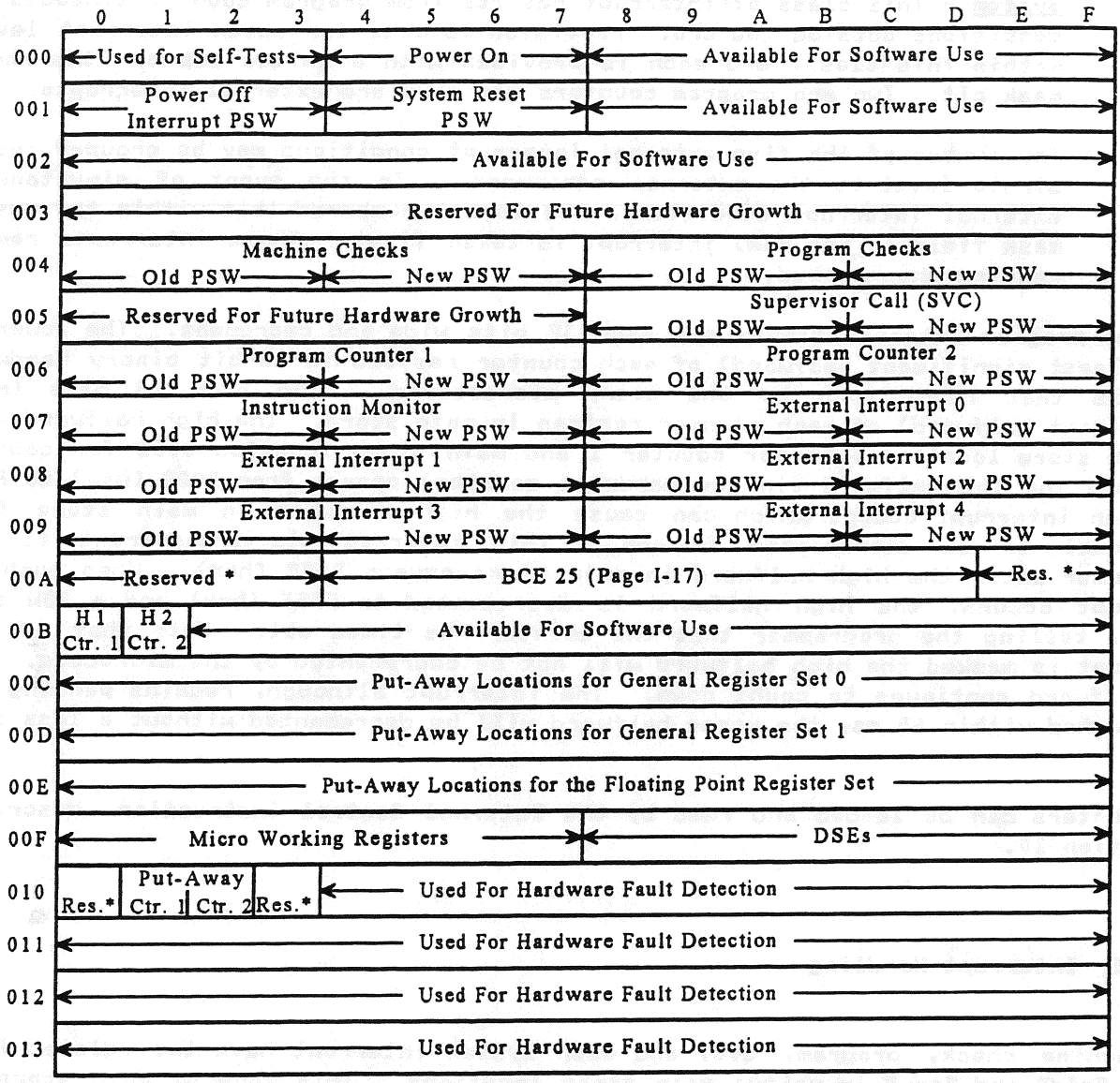
2.5.2 Interrupts

1. Power - This interrupt occurs when primary power is removed from the system for any reason. The current PSW, the general register set 1 and 2, the floating point registers, counters 1 and 2, and the current DSEs are put away (stored) in main storage for future reference. Figure 2-21 shows the PSA assignments including putaway. When primary power is restored, operation is initiated with the "power on PSW" (if the power-up mode is defined as Run). This power-up condition is explained in General System Operation.

2. Machine Check - When not masked, this interrupt class occurs following the detection of a malfunction. The current instruction is then terminated and the interrupt taken. A diagnostic procedure may then be initiated. When masked the interrupt does not remain pending.

3. Program - This class of interrupt arises from improper specification or use of instructions or data. Bits 20, 22, and 23 (1=interrupt enabled, 0=interrupt disabled) in the PSW are provided to permit masking program interrupts due to arithmetic exceptions such as fixed point overflow. Bit 34 in the PSW is provided to permit masking the instruction monitor interrupt. When masked, program interrupts do not remain pending. When invalid instruction or address detection is provided, the resulting program interrupts cannot be masked.

4. Supervisor Call (SVC) - This interrupt results from the execution of the SVC instruction. The four MSBs of the 19-bit extended EA are placed into the EA-high field (bits 40-43) of the old PSW, and the nonextended 16-bit EA is placed into the interrupt code (bits 48-63) of the old PSW. This instruction can be used to switch from the problem to the supervisor state.



* Reserved For Future Hardware Growth

DSE PUTAWAY FORMAT

ADDR	REGISTER SET 0				REGISTER SET 1			
00F8	RESV	DSE0	RESV	DSE1	RESV	DSE0	RESV	DSE1
00FA	RESV	DSE2	RESV	DSE3	RESV	DSE2	RESV	DSE3
00FC	RESV	DSE4	RESV	DSE5	RESV	DSE4	RESV	DSE5
00FE	RESV	DSE6	RESV	DSE7	RESV	DSE6	RESV	DSE7
BITS	0 3	4 7	8 11	12 15	16 19	20 23	24 27	28 31

Figure 2-21. Preferred Storage Area Assignments

5. System - This class of interrupt results from program counter timeouts and conditions outside the CPU. Provision is made for seven interrupt levels within this class, and each is provided with a unique set of PSWs and a mask bit. Two are program counters and five are external interrupts.

   Any number of the five external interrupt conditions may be grouped into a single level by the external equipment. In the event of simultaneous external interrupt conditions, the lowest numbered (bit within the system mask field in the PSW) interrupt is taken first. These interrupts remain pending when masked.

The two program interval timers are each 32 bits wide and decrement. The lower 16 bits (least significant halfword) of each counter resides in 16-bit binary hardware counters that count down by one every microsecond. The high 16 bits (most significant halfword) of each counter resides in main store. The high halfword lies in main store location 00B0 for counter 1 and main store location 00B1 for counter 2. When the low halfword (in the hardware counter) passes from 0000 (hex) to FFFF (hex) an interrupt occurs which can cause the high halfword in main store (via microcode) to be decremented by one. This interrupt is transparent to the programmer until the high halfword in main store equals 0000 (hex). When such an interrupt occurs, the high halfword is decremented to FFFF (hex) and a PSW swap occurs, telling the programmer that the counter has timed out. Note that if the interrupt is masked the high halfword will not be decremented by the microcode. The low halfword continues to count down. The interrupt although, remains pending and if unmasked within 65 ms, the upper halfword will be decremented without a loss of a count.

The counters can be loaded and read by the Internal Control instruction, described in Section 10.

2.5.2.1 Interrupt Handling

The machine check, program, SVC, and each system interrupt have two related PSWs called "old" and "new" in unique main store locations. This zone of main store is referred to as a preferred storage area (PSA), which is illustrated in Figure 2-21.

In all cases, an interruption involves merely storing the current PSW in its old position and making the PSW at the new position the current PSW. The old PSW holds all the necessary status information in the system existing at time of interruption. If, at the conclusion of the interruption routine, there is an instruction to make the old PSW the current PSW, the system is restored to the state prior to the interruption, and the interrupted routine continues. This means the programmer must clear the fixed point overflow indicator before being reloaded. Note that it is possible to switch to the alternate set of general registers when the PSW swap takes place. This set of registers is defined by bit 44 in the new PSW.

Interruptions can only be taken when the CPU is interruptible for a given source. The system mask, machine check mask bit, floating point exponent underflow mask, the significance mask, and the fixed point overflow mask bits in the PSW define the interruptible state of the CPU with respect to those sources. When masked, system interrupts remain pending while machine check and program interrupts are ignored.

The power transient, certain program interrupts, and the SVC interrupt cannot be masked.

Note: The pipeline is the driver for CPU multibit errors (IU and EA). Therefore, the machine check old PSW for CPU multibit error will reflect the updated PC - not the address of the multibit error. The following are ways in which a CPU multibit error may be encountered:

1. The instruction unit (IU) prefetching instructions (up to 23 halfwords ahead of the PC)

2. The effective address unit (EA) prefetching data (anywhere in memory)

3. The EA prefetching a branch target address (anywhere in memory).

In the event of this type of error, the error detection and correction (EDAC) address register may be read for determination of the actual multibit error address.

2.5.2.2 Interrupt Priority

Figure 2-20 presents the repertoire of interrupts with approximate priority levels. Individual interrupts are listed in order by classification, rather than by priority. The priority of each interrupt is represented by a two-digit code, which is interpreted as follows:

First Digit - represents the capture latch number (lower-numbered capture latches are examined first) or, if alphabetic, the fact that the interrupt is generated by the CPU - either a Command Interrupt (C), or a Supervisor Call PSW swap (P).

Second Digit - represents the priority of the interrupt within a grouping (hardware or "other").

Conceptually, the order of processing (in the case of interrupts received simultaneously) is as follows:

1. Group 0 Interrupts - These are the highest priority - the Power/Machine Check type interrupts. The Power, System Reset, and IPL interrupts clear all pending interrupts - the remaining Group 0 interrupts do not. See Page 2-21 for interrupt structure and priority.

2. Command Interrupts - These are usually interrupts which demand direct communication from the CPU to the Interrupt Page Processor. Often, they are included within a CPU microcode procedure. Action taken by the CPU is usually to request the interrupt and then loop at one microword, waiting for the Interrupt Page to reset the Control Store Data Register, thereby forcing a branch to zero.

3. Group 1, 2, or 3 Interrupts - These interrupts differ from the following two groups in that the hardware freezes the CPU microcode at the next ENDOP when one of them is detected.

4. Group 4 or 5 Interrupts - These interrupts are the only types that are held pending until they are unmasked with no additional higher-priority interrupts present. They are only accepted at ENDOP time and generally cause only slight CPU processing delays if they are masked OFF.

When more than one unmasked interrupt requests service, the current (old) PSW is stored into and the new PSW is fetched from two PSA locations assigned to the first interrupt to be processed. Then, the same procedure is followed using the PSA locations of the second interrupt, with the exception that the "old" PSW is the former new PSW as fetched for the first interrupt. This procedure of "passing" the PSW is continued until the last interrupt request is acknowledged. Then, instruction execution is commenced using the PSW last fetched. The order of execution of the interrupt service routines is, consequently, the reverse of the order in which the string of "new" PSWs were fetched. Machine Check and Power Transient interruptions supersede all other interrupts when they are encountered.

The priority scheme as outlined above is used to resolve race conditions due to multiple interrupt conditions. However, since in the case of most normal interrupts (those expected to be encountered during the execution of typical application software) separate mask bits and PSW locations are provided for each external source, the priority of handling these interrupts is further affected by the contents of the PSWs actually fetched during the interrupt service overhead. That is, as each PSW swap occurs, further action with regard to System (and Machine Check) interrupts is determined by the mask fields encountered within the new PSW.

Two major exceptions to the above process involve the Instruction Monitor Interrupt and Supervisor Call. Instruction Monitor conditions are monitored by hardware and cause no processing delays if masked OFF, since the Interrupt Page will not even be notified of the condition in that event. It could be argued that Supervisor Call might not be considered an interrupt at all, since it is not an unexpected condition and is appropriately handled by the CPU microcode, but it is included in the list because its execution necessitates a PSW SWAP and, therefore, cooperation by the Interrupt Page processor in that portion of the instruction implementation.

2.5.2.3 Interrupt Masking

Individual masking of several of the interrupt types is possible. When masked off, the interruption is either ignored or remains pending for later execution. The masking capability for each of the interrupt types is as follows:

1. Power Transient - Cannot be masked off.

2. Machine Check - Can be masked off by setting the machine check mask bit 45 in the PSW equal to zero. When masked off, normal instruction sequencing occurs, and the interrupts do not remain pending.

3. Program - Three of the 11 program interrupts are capable of being masked off; fixed point arithmetic overflow, exponent underflow, and significance, by setting the appropriate mask bits in the PSW equal to zero. When masked off, these interruptions do not remain pending. Note that if a PSW with both Fixed Point Overflow Indicator and mask (bits 19 and 20) set is used, the interrupt will occur.

4. Supervisor Call - Cannot be masked off.

5. System - Each level of external interrupts can individually be masked off by setting the corresponding system mask bit in the PSW equal to zero. Interrupts that are masked remain pending.

2.5.2.4 Preferred Storage Area (PSA) Assignments

The contents of the PSA are shown in Figure 2-21 with the main store address expressed in hexadecimal notation. The following PSA locations must not be store protected:

1. Power off interrupt PSW
2. All old PSW locations
3. BCE 25 processor storage (00A4 - 00A5)
4. Counter 1 and 2, high halfword locations 00B0 and 00B1
5. Putaway locations (00C0 through 0102)
6. Diagnostics (104-13F).

2.5.3 General System Operation

The various states entered by the computer and their relationship to the basic operator controls are shown in Figure 2-22. The basic controls provided for the operator are power-on, initial program load (IPL) and the system reset key. Among the many controls available, these functions have special significance because of their relationship to an unconditional system reset sequence. These functions each produce a system reset sequence which applies to the computer, I/O channels, and peripherals. Further operation within the system differs, however, as explained in the following sections.

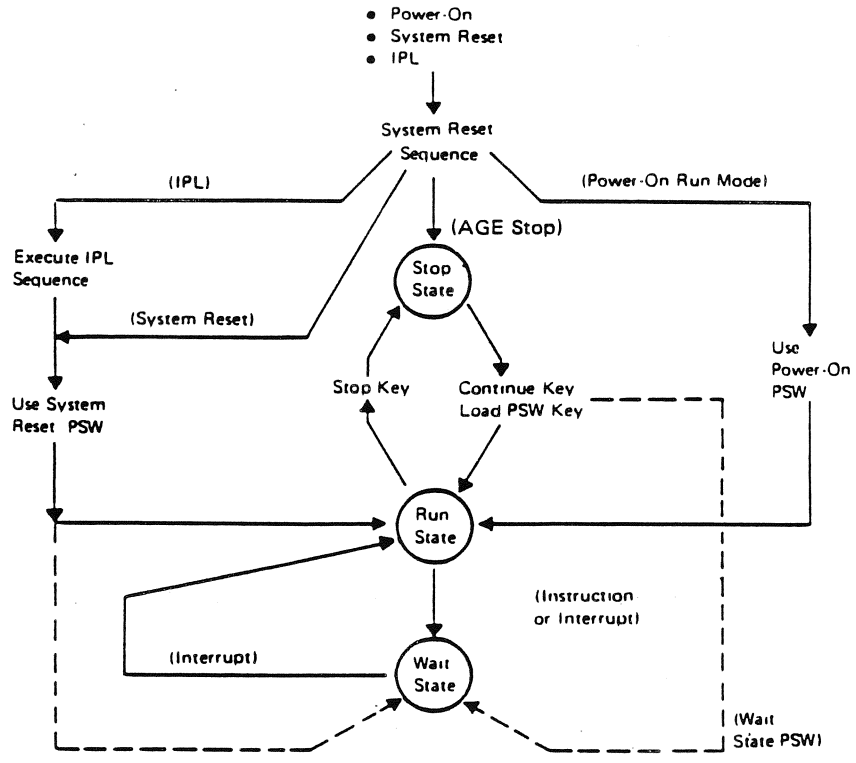


Figure 2-22. CPU Mode Switching

2.5.3.1 Power-On

One of two modes of operation must be specified for the system at power-on. The first results in a system reset followed by the computer entering the stop state. In this state, instructions are not processed, interrupts are not accepted, and system timers are not updated. This system is termed "manual" because further operation must be determined by the operator.

The second mode at power-on enters the run state after the system reset is complete. The instruction stream is initiated and interrupts are processed. The computer can be removed from the run state by certain instructions, interruptions, and by manual intervention.

2.5.3.2 System Reset

The system reset function resets the computer system to a known state such that processing can be initiated without the presence of machine checks, except for those caused by subsequent machine malfunctions. The system reset function causes the following:

- CPU pending interrupts are reset
- Internal timers are reset to all ones (1's)
- Status registers are reset
- DSE registers are set to zero.

2.5.3.3 IPL

The use of the IPL function is independent of the prior state of the system. IPL first causes a system reset function and the writing of C6C6 (hex) by the CPU to all memory locations above and including address 20000 Hex with memory store protected. IOP microcode at IPL writes C9FB (hex) to all locations from 0 to 1FFFF Hex, with memory store protected.

2.5.4 Operating State

The run state and wait state shown in Figure 2-22 are collectively termed the operating state for the system. When the computer is in the run state, instructions are executed in the normal manner. An instruction may be encountered or an interrupt processed that forces the computer into the wait state. The computer does not execute instructions in the wait state, but it is interruptible when not masked. System timers are updated and input/output operations continue in the wait state.

The wait state may also be entered after completing IPL or by special operating intervention via the stop state (dotted lines on Figure 2-22). This action is the result of the wait bit being set in the controlling PSW.

2.5.4.1 Program State Alternatives

Certain other states exist within the CPU that contribute to its overall status. These states are directly related to program operation and are:

1. Masked or Interruptible State - The computer may be masked for certain interrupt conditions at any given time. These conditions generally remain pending within the system until the masked condition is changed by the program. Certain error conditions cannot be masked off, while other error conditions, such as program checks, are ignored when specifically masked.

2. Supervisor or Problem State - In the supervisor state, all instructions are valid. In the problem state, I/O and certain other instructions are invalid, and their use produces an error interrupt. This state is controlled by bit 47 in the PSW. The SVC instruction is provided to switch from problem to supervisor state. The LOAD PSW instruction is used

to switch from supervisor to problem state.

3. General Register Selection - Bit 44 is the current PSW and selects the set of general registers in current use.

2.5.5 Architectural Growth

Throughout this Principles of Operation manual, architecture conventions are defined or facilities are marked "reserved" to retain flexibility for future implementations and extensions. The computer operates in conformance to this manual when architecture definitions are followed consistently. Hardware operation, when these rules are violated, is not defined and is properly outside the scope of this manual to retain flexibility of implementation. "Programmer discovered" operations that violate or go beyond the definitions described herein, but produce "useful" functions, should not be used and should be considered "reserved", because the results obtained may vary from computer to computer, or even release levels for one computer, depending upon options selected or the design release level to which the hardware is manufactured.

3.0 CPU I/O

The transfer of information with input/output occurs in one of two modes:

1. Direct Memory Access (IOP initiated and controlled)
2. Program Controlled (CPU initiated and controlled).

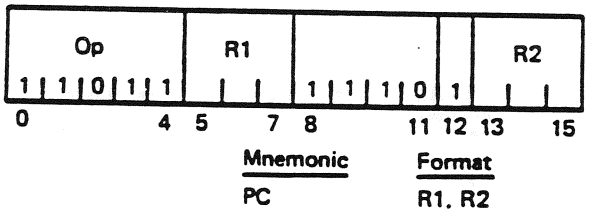
3.1 DIRECT MEMORY ACCESS OPERATION

Direct Memory Access (DMA) operations are IOP initiated. Although the resulting cycle steal memory access preempts CPU accesses, thereby slowing program execution, DMA operations are not under program control and are transparent to the functional operation of the CPU. DMA operations can occur between CPU memory cycles during instruction execution, unless the instruction specifies that DMAs are held off during execution of that instruction.

3.2 PROGRAM-CONTROLLED INPUT/OUTPUT OPERATION

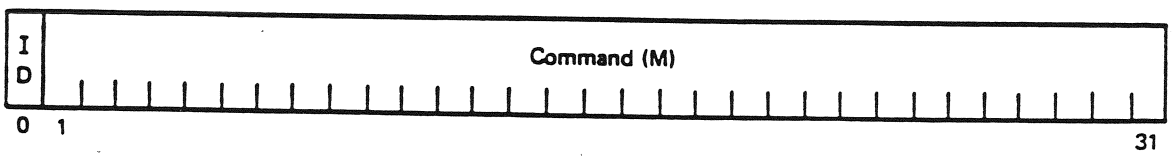
Program-Controlled I/O operations transfer one fullword between a CPU general register and an IOP Subsystem. The operation is initiated by executing the privileged instruction "PC Input/Output". A control word (CW), in a second general register specified by the instruction, defines the specific I/O operation and the specific IOP Subsystem associated with the operation.

3.3 PROGRAM-CONTROLLED I/O INSTRUCTION



DESCRIPTION:

The Input/Output instruction transfers a fullword to or from the general register specified by R1. Direct I/O operations are defined by a control word (CW) contained in the general register specified by R2. The CW format is shown below:



ID: For an input operation, bit 0 must be coded as 0. For an output operation, this bit must be coded as 1.

Command (M): Bits 1-31 specify the particular operation to be performed. In executing an input operation, the channel (1) transmits the 32-bit CW to the IOP Subsystem; and (2) subsequently loads 32 bits of information, transmitted from the IOP Subsystem, into general register R1. In executing an output operation, the channel (1) transmits the CW to the IOP Subsystem, and (2) subsequently transmits bits 0-31 of general register R1 to the IOP Subsystem. The specific definition of the command bits is described in Appendix I, Program Controlled Inputs and Outputs.

Each control unit connected to the channel is required to accept the CW, decode the control unit and device address, and perform the input or output operation defined by the command field.

If the I/O handshaking operation does not complete within 9.5 microseconds for CW and DATA OUT transfers or 6.5 microseconds for DATA IN transfers, the Program-Controlled instruction will terminate and the condition code will be set to reflect the timeout.

RESULTING CONDITION CODE:

00	Operation successful
01	Interface timeout error; operation not successful

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

Program Interrupt - Privileged instruction.

PROGRAMMING NOTE:

This is a privileged instruction and can only be executed when the CPU is in the supervisor state.

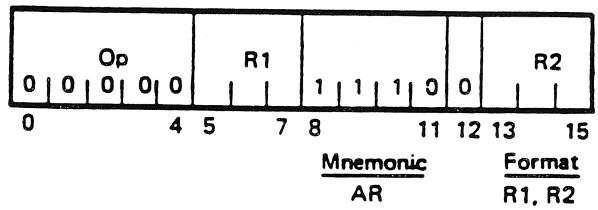
4.0 FIXED POINT ARITHMETIC

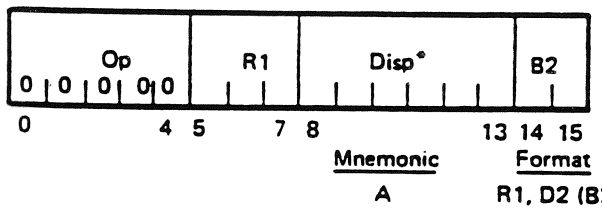
For all of the following sections, [@] [#] indicates that the use of indirect addressing and/or autoindexing is optional. For example, M specifies direct addressing without autoindexing, while M# specifies direct addressing with autoindexing.

The arithmetic instruction set performs binary arithmetic on fixed point, fractional operands. Fullword operands are signed and 32 bits long. Negative quantities are represented in twos complement form.

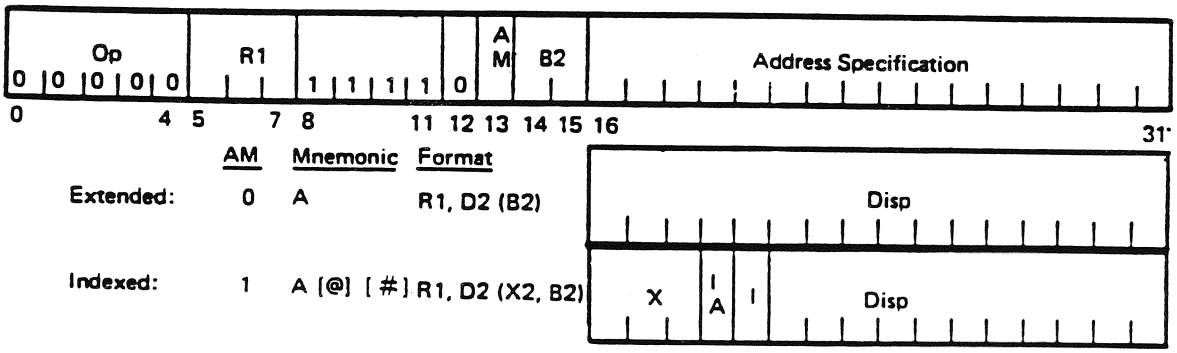
Halfword operands are 16 bits long. Within the CPU, a halfword operand from storage is developed into a fullword operand prior to instruction execution. This is done by using the contents of the halfword second operand location as the most significant 16 operand bits and generating 16 low-order zeros. This result is the second operand.

4.1 ADD





*Displacements of the form 111XXX are not valid.



DESCRIPTION:

The fullword second operand is added to the contents of general register R1. The result replaces the contents of general register R1. The second operand is not changed.

RESULTING CONDITION CODE:

00 The result is zero
11 The result is negative
01 The result is positive (>0)

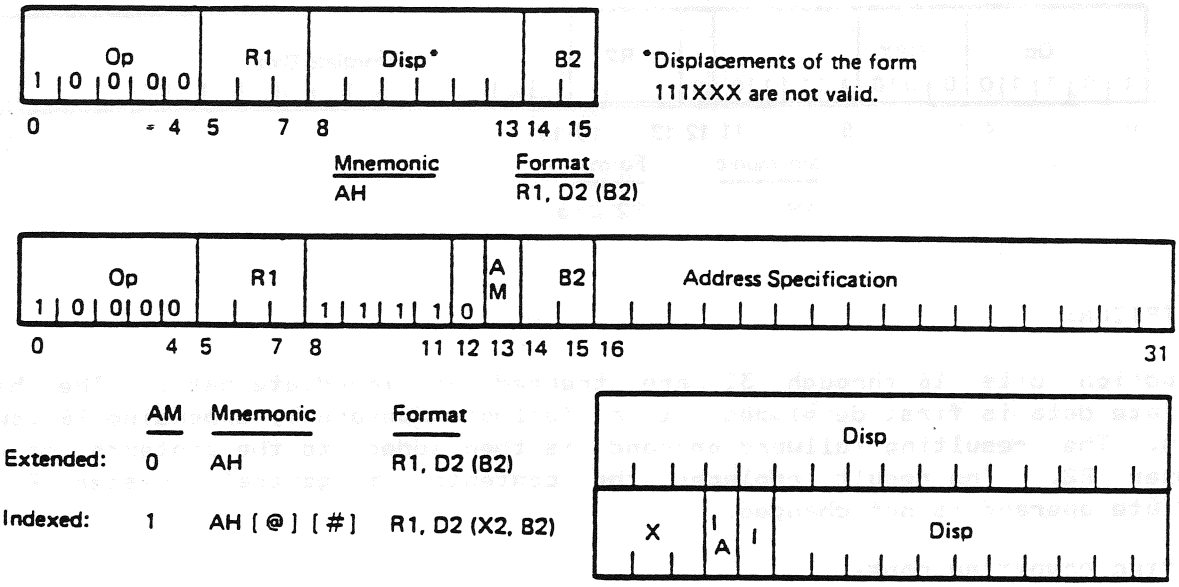
INDICATORS:

The overflow indicator is set to one if the magnitude of the sum is too large to be represented in the general register; that is, greater than $1-2^{-31}$, or less than or = -1. If the overflow indicator already contains a one, it is not altered by this instruction. The carry indicator is set to indicate whether or not there is a carry out of the high-order bit position of the general register.

PROGRAM INTERRUPTS:

Fixed point overflow

4.2 ADD HALFWORD



DESCRIPTION:

The halfword second operand is first developed into a fullword operand by appending 16 low-order zeroes. This fullword operand is then added to the contents of general register R1. The result replaces the contents of general register R1. The second operand is not changed.

RESULTING CONDITION CODE:

00 The result is zero
11 The result is negative
01 The result is positive (>0)

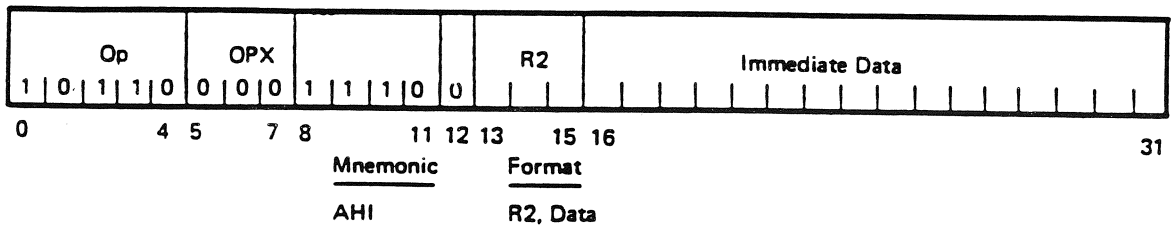
INDICATORS:

The overflow indicator is set to one, if the magnitude of the sum is too large to be represented in the general register; that is, greater than $1-2^{-31}$, or less than or = -1. If the overflow indicator already contains a one, it is not altered by this instruction. The carry indicator is set to indicate whether or not there is a carry out of the high-order bit position of the general register.

PROGRAM INTERRUPTS:

Fixed point overflow

4.3 ADD HALFWORD IMMEDIATE



DESCRIPTION:

Instruction bits 16 through 31 are treated as immediate data. The halfword immediate data is first developed into a fullword operand by appending 16 low-order zeroes. The resulting fullword operand is then added to the contents of general register R2. The result replaces the contents of general register R2. The immediate operand is not changed.

RESULTING CONDITION CODE:

00 The result is zero
11 The result is negative
01 The result is positive (>0)

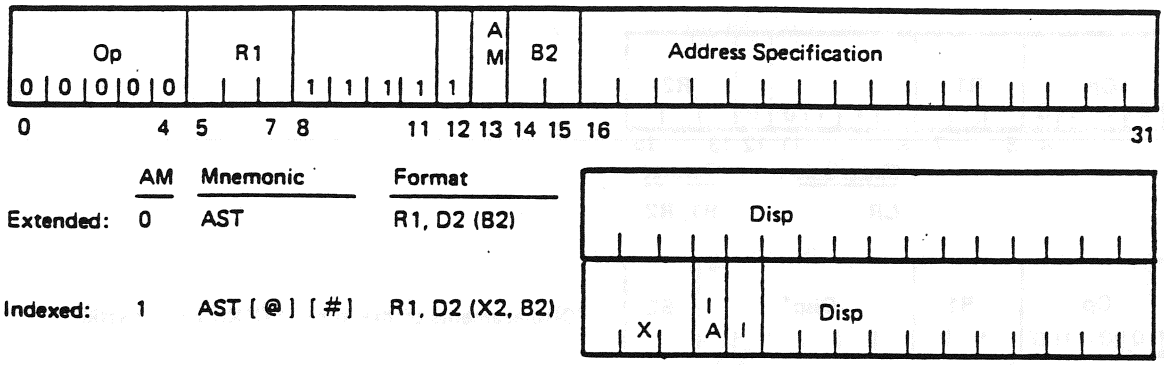
INDICATORS:

The overflow indicator is set to one if the magnitude of the sum is too large to be represented in the general register; that is, greater than $1-2^{-31}$, or less than or = -1. If the overflow indicator already contains a one, it is not altered by this instruction. The carry indicator is set to indicate whether or not there is a carry out of the high-order bit position of the general register.

PROGRAM INTERRUPTS:

Fixed point overflow

4.4 ADD TO STORAGE



DESCRIPTION:

The contents of general register R1 is added to the fullword second operand. The result replaces the contents of the second operand location. The first operand is not changed.

RESULTING CONDITION CODE:

00 The result is zero
11 The result is negative
01 The result is positive (>0)

INDICATORS:

The overflow indicator is set to one if the magnitude of the sum is too large to be represented in the second operand location. That is, greater than $1-2^{-31}$, or less than or = -1. If the overflow indicator already contains a one, it is not altered by this instruction. The carry indicator is set to indicate whether or not there is a carry out of the high-order bit position of the result.

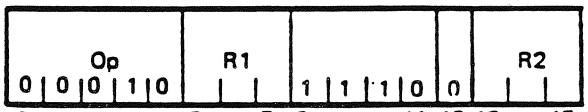
PROGRAM INTERRUPTS:

Fixed point overflow

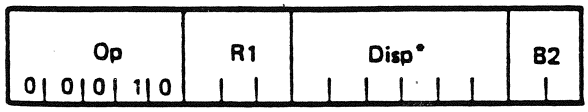
WARNING!

This instruction requires multiple memory accesses. The CPU does not prohibit IOP accesses of the selected main storage location during the time between the fetch of the operand and store of the result. Therefore, this instruction should not be used with any memory locations that might be DMA'd into.

4.5 COMPARE

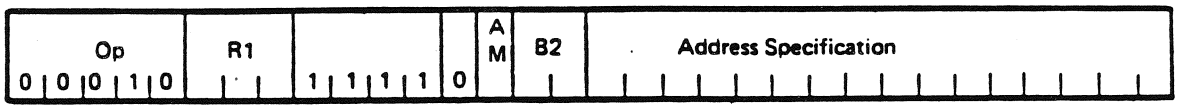


Mnemonic	Format
CR	R1, R2

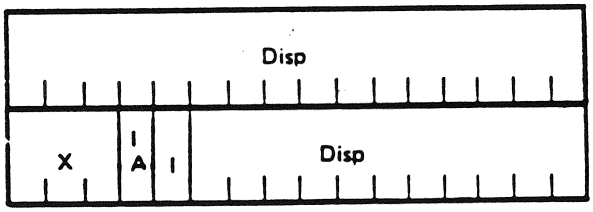


Mnemonic	Format
C	R1, D2 (B2)

*Displacements of the form 111XXX are not valid.



	AM	Mnemonic	Format
Extended:	0	C	R1, D2 (B2)
Indexed:	1	C [@] [#]	R1, D2 (X2, B2)



DESCRIPTION:

The fullword second operand is algebraically compared with the contents of general register R1. The contents of general register R1 and main storage are not changed at the end of instruction execution.

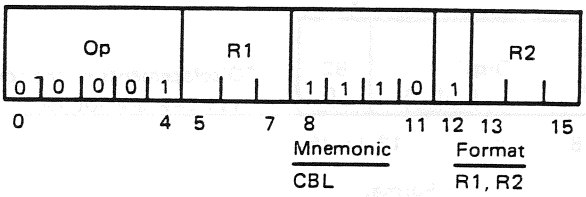
RESULTING CONDITION CODE:

00 The contents of general register R1 equals the second operand
11 The contents of general register R1 are less than the second operand
01 The contents of general register R1 are greater than the second operand

INDICATORS:

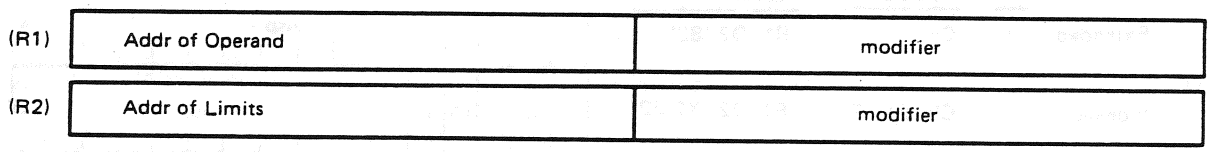
The overflow and carry indicators are not changed by this instruction.

4.6 COMPARE BETWEEN LIMITS

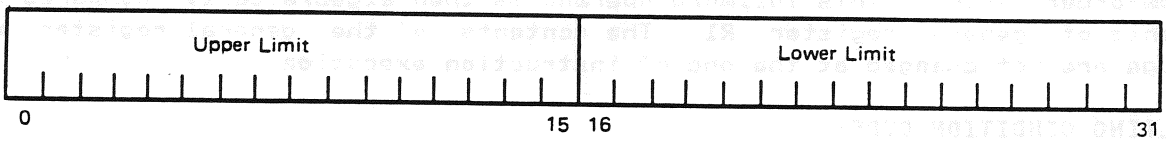


DESCRIPTION:

A compare between limits instruction occurs. The condition code reflects the result of the comparison.



The address of a 16-bit twos complement integer operand is contained in bits 0 through 15 of general register R1. The address of a fullword with the following format containing the upper and lower limits is contained in bits 0 through 15 of the general register R2:



These limits are 16-bit twos complement integers.

In bits 16 through 31 of general registers R1 and R2 are 16-bit twos complement integer modifiers. After the addresses in bits 0 through 15 have been used to locate the operands, each modifier is added to the most significant 16 bits of the registers. The result replaces the most significant 16 bits. The modifier is not changed, overflows and carry out of the most significant address bit are ignored.

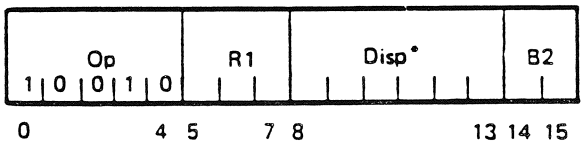
RESULTING CONDITION CODE:

00	Within Limits:	Lower Limit ≤ Operand ≤ Upper Limit
01	Above Upper Limit:	Operand > Upper Limit
11	Below Lower Limit:	Operand < Lower Limit

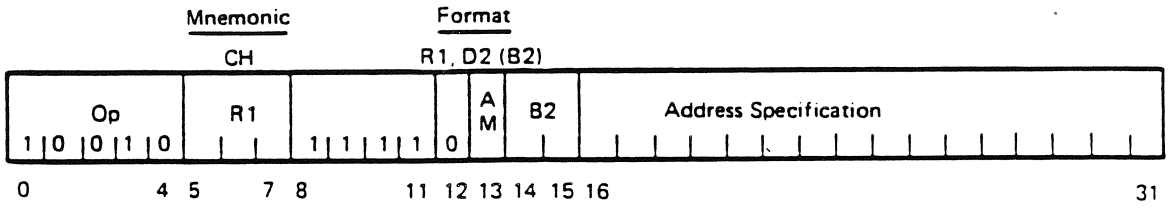
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

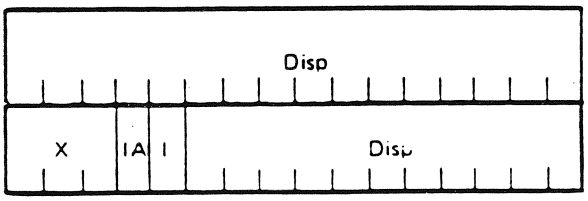
4.7 COMPARE HALFWORD



*Displacements of the form 111XXX are not valid.



	AM	Mnemonic	Format
Extended:	0	CH	R1, D2 (B2)
Indexed:	1	CH[@] [#]	R1, D2 (X2,B2)



DESCRIPTION:

The halfword second operand is first developed into a fullword operand by appending 16 low-order zeros. This fullword operand is then algebraically compared with the contents of general register R1. The contents of the general register and main storage are not changed at the end of instruction execution.

RESULTING CONDITION CODE:

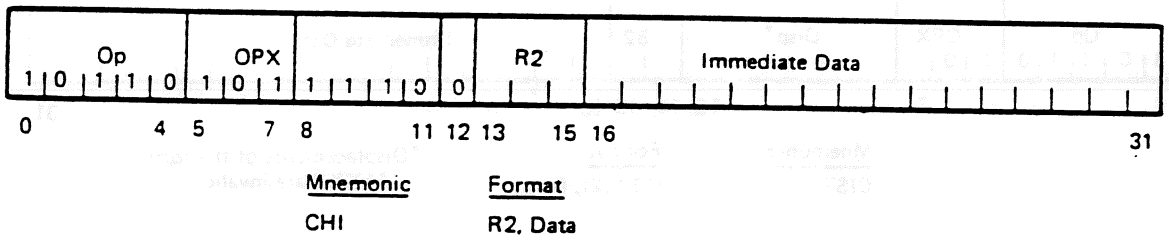
00 The contents of general register R1 equals the developed fullword operand
11 The contents of general register R1 are less than the developed fullword operand
01 The contents of general register R1 are greater than the developed fullword operand

INDICATORS: The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

After development, all 32 bits of the fullword operand participate in the comparison.

4.8 COMPARE HALFWORD IMMEDIATE



Mnemonic	Format
CHI	R2, Data

DESCRIPTION:

Instruction bits 16 though 31 are treated as immediate data. This halfword of immediate data is first developed into a fullword operand by appending 16 low-order zeros. This fullword operand is then algebraically compared with the contents of general register R2. The contents of the general register and main storage are not changed at the end of instruction execution.

RESULTING CONDITION CODE:

00 The contents of general register R2 equals the developed fullword operand
11 The contents of general register R2 are less than the developed fullword operand
01 The contents of general register R2 are greater than the developed fullword operand

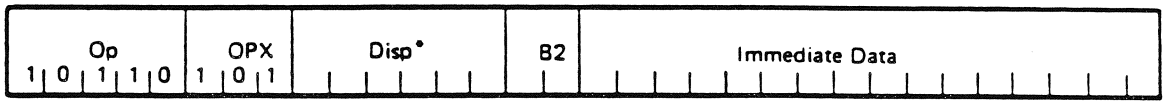
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

After development, all 32 bits of the fullword operand participate in the comparison.

4.9 COMPARE IMMEDIATE WITH STORAGE



Mnemonic
CIST

Format
D2 (B2), Data

*Displacements of the form 111XXX are invalid.

DESCRIPTION:

Instruction bits 16 through 31 are treated as immediate data. This is algebraically compared with the halfword main storage operand. The immediate data and the contents of main storage are not changed at the end of this instruction.

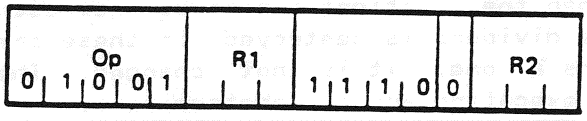
RESULTING CONDITION CODE:

00 The immediate data equals the halfword main storage operand
11 The immediate data is less than the halfword main storage operand
01 The immediate data is greater than the halfword main storage operand

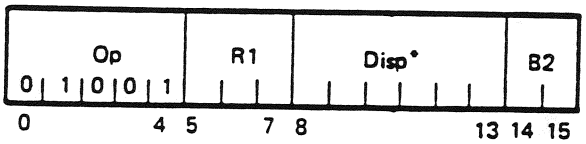
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

4.10 DIVIDE

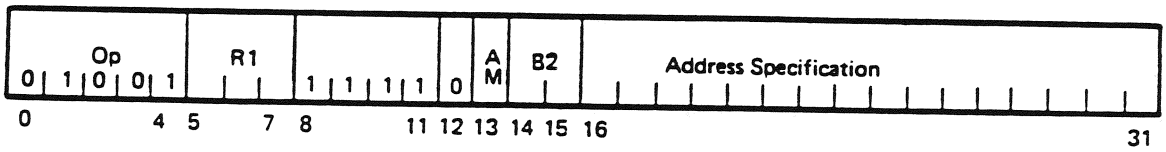


Mnemonic	Format
DR	R1, R2

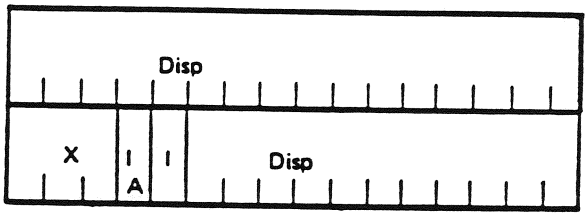


*Displacements of the form 111XXX are not valid.

Mnemonic	Format
D	R1, D2 (B2)



	AM	Mnemonic	Format
Extended:	0	D	R1, D2 (B2)
Indexed:	1	D [@] [#]	R1, D2 (X2, B2)



DESCRIPTION:

The first operand, a 64-bit, signed twos complement dividend, is contained in the general register pair R1 and (R1+1)mod8. The most significant portion is in R1. When R1 indicates an odd general register, the first operand is developed by appending 32 low-order zeros to the contents of R1. The second operand is the divisor.

The first operand is divided by the second operand. The unrounded quotient replaces the contents of general register R1. The remainder is not developed. When R1 is even, specifying an even/odd general register pair, the contents of (R1 + 1) mod 8 are indeterminant at the end of instruction execution. When R1 is odd, (R1 + 1) mod 8 is never changed. The second operand is not changed.

When the relative magnitude of dividend and divisor is such that the quotient cannot be expressed as a 32-bit signed fraction, an overflow is generated. In this event, the contents of both R1 (and R1 + 1 when R1 is even) are indeterminate upon instruction termination.

RESULTING CONDITION CODE:

The code is not changed.

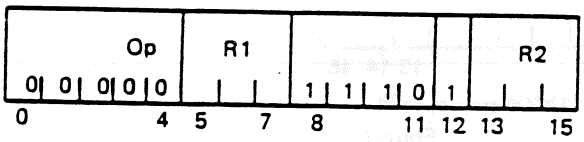
INDICATORS:

The overflow indicator is set to one when the quotient cannot be represented, or when division by zero is attempted. The dividend is destroyed in these cases. If the overflow indicator already contains a one, it is not changed. The carry indication has no significance following execution and is indeterminate.

PROGRAM INTERRUPTS:

Fixed point overflow

4.11 EXCHANGE UPPER AND LOWER HALFWORDS



Mnemonic	Format
XUL	R1, R2

DESCRIPTION:

The upper halfword of general register R1 is exchanged with the lower halfword of general register R2. Bits 0 through 15 of general register R1 replace bits 16 through 31 of general register R2, while simultaneously bits 16 through 31 of general register R2 replace bits 0 through 15 of general register R1.

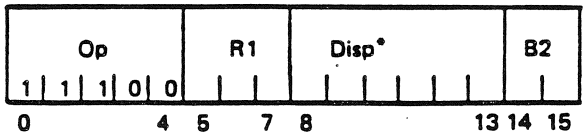
RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

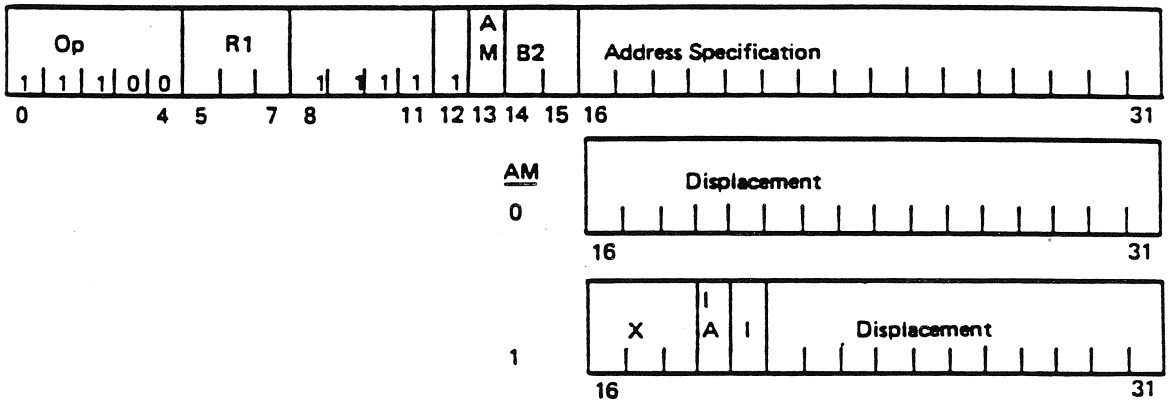
The overflow and carry indicators are not changed by this instruction.

4.12 INSERT ADDRESS LOW



* Displacements of the form 111XXX are not valid.

Mnemonic	Format
IAL	R1, D2 (B2)



	AM	Mnemonic	Format
Extended:	0	IAL	R1, D2 (B2)
Indexed:	1	IAL [@] [#]	R1, D2 (X2, B2)

DESCRIPTION:

A 16-bit effective address is developed in the normal manner utilizing halfword index alignment, if specified. This address itself replaces the 16 low-order bits of general register R1. The 16 high-order bits of general register R1 are not changed.

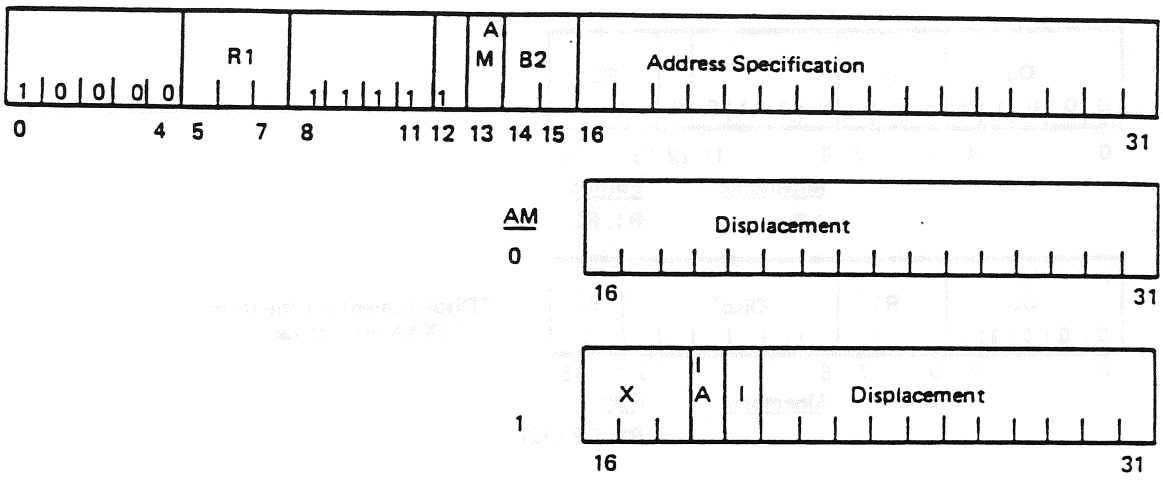
RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

4.13 INSERT HALFWORD LOW



	AM	Mnemonic	Format
Extended:	0	IHL	R1, D2 (B2)
Indexed:	1	IHL [@] [#]	R1, D2 (X2, B2)

DESCRIPTION:

The halfword second operand replaces the contents of bits 16-31 of general register R1. Bits 0-15 of general register R1 are not changed. The second operand is not changed.

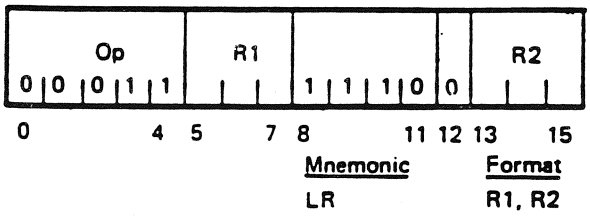
RESULTING CONDITION CODE:

The code is not changed.

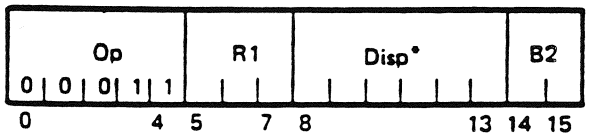
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

4.14 LOAD

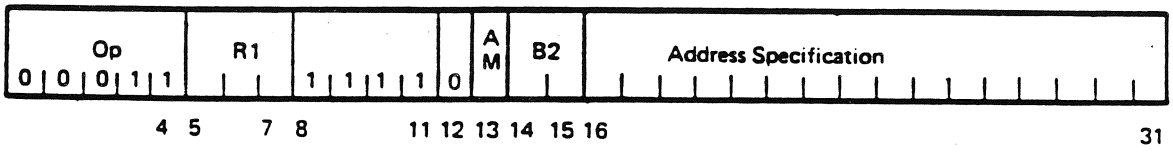


Mnemonic	Format
LR	R1, R2

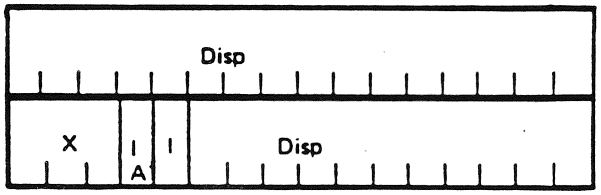


Mnemonic	Format
L	R1, D2 (B2)

*Displacements of the form 111XXX are not valid.



	AM	Mnemonic	Format
Extended:	0	L	R1, D2 (B2)
Indexed:	1	L [@] [#]	R1, D2 (X2, B2)



DESCRIPTION:

The fullword second operand is placed in general register R1. The second operand is not changed.

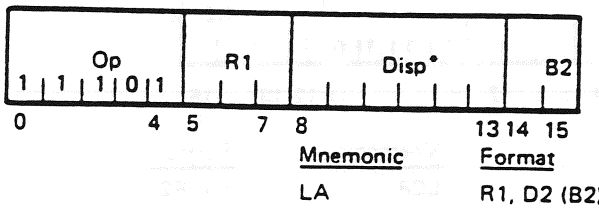
RESULTING CONDITION CODE:

00 The second operand is zero
11 The second operand is negative
01 The second operand is positive (>0)

INDICATORS:

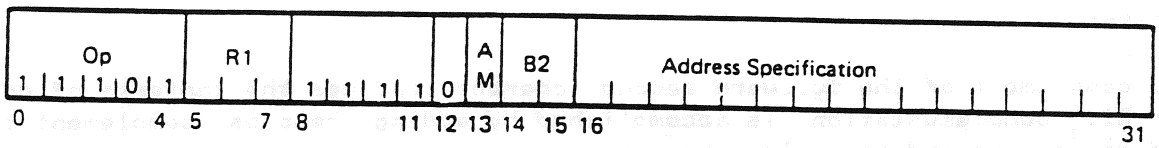
The overflow and carry indicators are not changed by this instruction.

4.15 LOAD ADDRESS

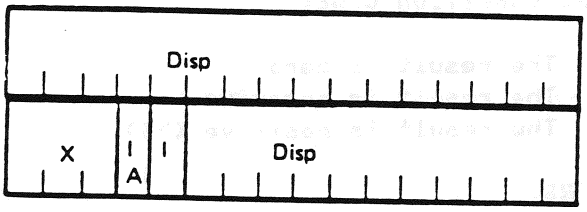


*Displacements of the form 111XXX are not valid.

Mnemonic	Format
LA	R1, D2 (B2)



	AM	Mnemonic	Format
Extended:	0	LA	R1, D2 (B2)
Indexed:	1	LA [@] [#]	R1, D2 (X2, B2)



DESCRIPTION:

A 16-bit effective halfword address is developed in the normal manner without expanding to 19 bits. This address itself replaces the 16 high-order bits of general register R1. The 16 low-order bits of general register R1 are zeroed.

RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

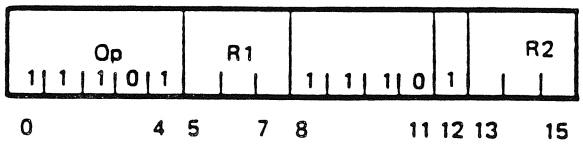
The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

When R1 = B2, it is possible to increment R1 by the displacement field.

In the RS format when B2 = 11 and AM = 0, this is functionally equivalent to a LOAD HALFWORD IMMEDIATE instruction. In this case, bits 16 through 31 are treated as immediate data. The immediate data is expanded to 32 bits by appending 16 low-order zeros. This resulting fullword operand replaces the contents of general register R1.

4.16 LOAD ARITHMETIC COMPLEMENT



Mnemonic	Format
LCR	R1, R2

DESCRIPTION:

The twos complement of the fullword second operand replaces the contents of general register R1. Complementation is accomplished by adding the ones complement of the fullword second operand and a low-order one.

RESULTING CONDITION CODE:

00 The result is zero
11 The result is negative
01 The result is positive (>0)

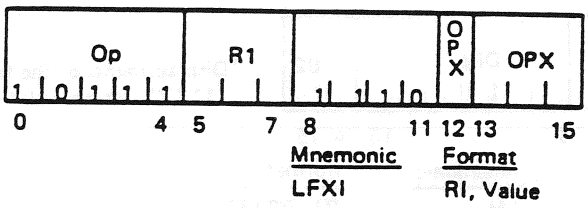
INDICATORS:

The overflow indicator is set to one when the maximum negative number is complemented. If the overflow indicator already contains a one, it is not altered by this instruction. The carry indicator is set to indicate whether or not there is a carry out of high-order bit position of general register. The carry indicator will only be set when the operand is zero.

PROGRAM INTERRUPTS:

Fixed point overflow

4.17 LOAD FIXED IMMEDIATE



DESCRIPTION:

A fixed point literal value is loaded into the general register specified by R1.

The immediate values are -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 or 13. The immediate is loaded into bits 0 through 15 of general register R1. Bits 16 through 31 of general register R1 are set to zero.

OPX (Bits 12, 13, 14 & 15) (hex)	Immediate Value --> R1 (hex)
0	FFFE0000
1	FFFF0000
2	00000000
3	00010000
4	00020000
5	00030000
6	00040000
7	00050000
8	00060000
9	00070000
A	00080000
B	00090000
C	000A0000
D	000B0000
E	000C0000
F	000D0000

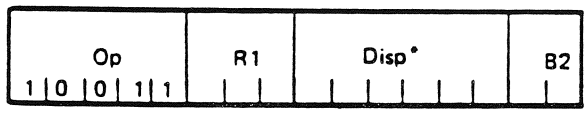
RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

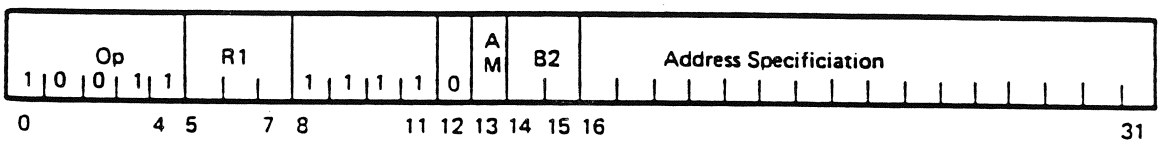
The overflow and carry indicators are not changed by this instruction.

4.18 LOAD HALFWORD

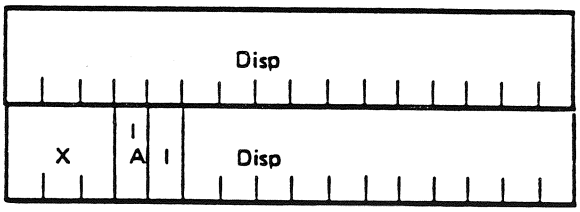


*Displacements of the form 111XXX are not valid.

Mnemonic	Format
LH	R1, D2 (B2)



	AM	Mnemonic	Format
Extended:	0	LH	R1, D2 (B2)
Indexed:	1	LH [@] [#]	R1, D2 (X2, B2)



DESCRIPTION:

The halfword second operand is developed into a fullword operand by appending 16 low-order zeros. The resulting fullword operand replaces the contents of general register R1. The second operand is not changed.

RESULTING CONDITION CODE:

00 The fullword operand is zero
11 The fullword operand is negative
01 The fullword operand is positive (>0)

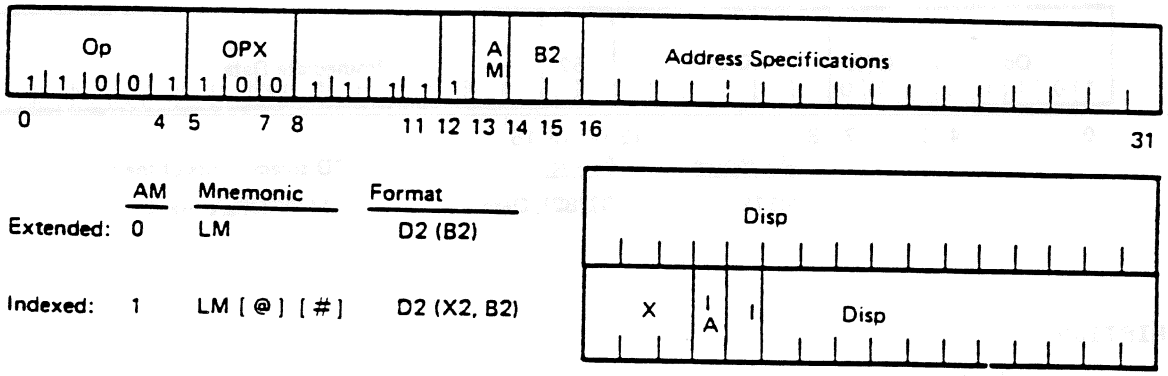
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

This instruction clears the low-order half of general register R1.

4.19 LOAD MULTIPLE



	AM	Mnemonic	Format
Extended:	0	LM	D2 (B2)
Indexed:	1	LM [@] [#]	D2 (X2, B2)

DESCRIPTION:

All eight general registers are loaded from the eight fullword locations starting at the fullword, second operand address. The general registers are loaded in ascending order.

RESULTING CONDITION CODE:

The code is not changed.

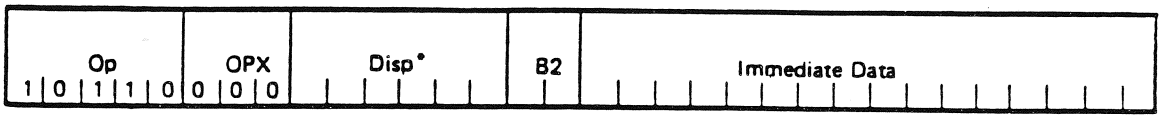
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

This instruction will always have halfword index alignment and will be excluded from automatic index alignment.

4.20 MODIFY STORAGE HALFWORD



Mnemonic	Format	*Displacements of the form
MSTH	D2(B2), Data	111XXX are invalid.

DESCRIPTION:

Instruction bits 16 through 31 are treated as immediate data representing a twos complement integer. This immediate data is added to the halfword main storage operand. The result replaces the halfword main storage operand. The contents of the general registers are not changed. Only the contents of the halfword main storage operand location are altered.

RESULTING CONDITION CODE:

00 The result is zero
11 The result is negative
01 The result is positive (>0)

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

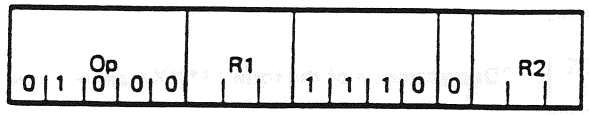
PROGRAMMING NOTES:

The MSTH immediate data (mask) is algebraically added to the halfword operand in main storage. Tally up and tally down is thus possible.

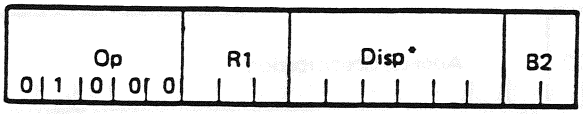
WARNING!

This instruction requires multiple memory accesses. The CPU does not prohibit IOP accesses of the selected main storage location during the time between the fetch of the operand and store of the result. Therefore, this instruction should not be used with any memory locations that might be DMA'd into.

4.21 MULTIPLY

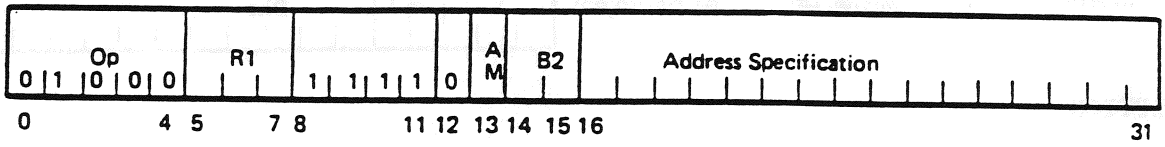


Mnemonic: MR
Format: R1,R2

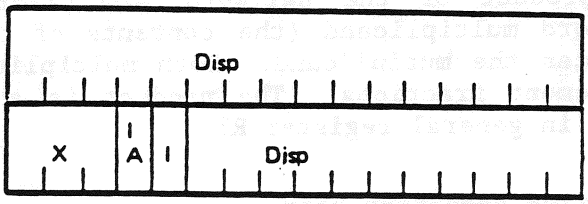


Mnemonic: M
Format: R1,D2(B2)

*Displacements of the form 111XXX are not valid.



	AM	Mnemonic	Format
Extended:	0	M	R1,D2(B2)
Indexed:	1	M [@] [#]	R1,D2(X2,B2)



DESCRIPTION:

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 twos complement fractions. The product is a 64-bit, signed twos complement fraction number and occupies an even/odd register pair when the R1 field references an even-numbered general register. When R1 is odd, only the most significant 32 bits of the product is saved in general register R1.

RESULTING CONDITION CODE:

The code is not changed.

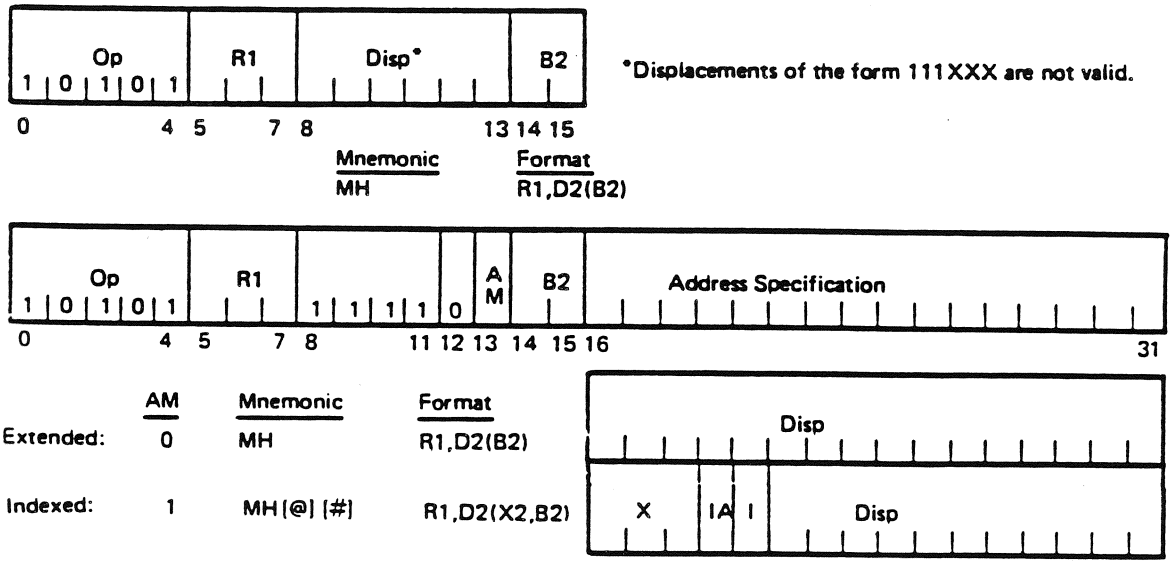
INDICATORS:

The overflow indicator is set to one when -1 is multiplied by -1. If the overflow indicator already contains a one, it is not altered by this instruction.

PROGRAM INTERRUPTS:

Fixed point overflow

4.22 MULTIPLY HALFWORD



DESCRIPTION:

The product of the halfword multiplier (the halfword second operand) and the halfword multiplicand (the contents of bits 0 through 15 of general register R1) replaces the mutiplicand. Both multiplier and multiplicand are 16-bit signed twos complement fractions. The product is a 32-bit signed fraction. This product is saved in general register R1.

RESULTING CONDITION CODE:

The code is not changed.

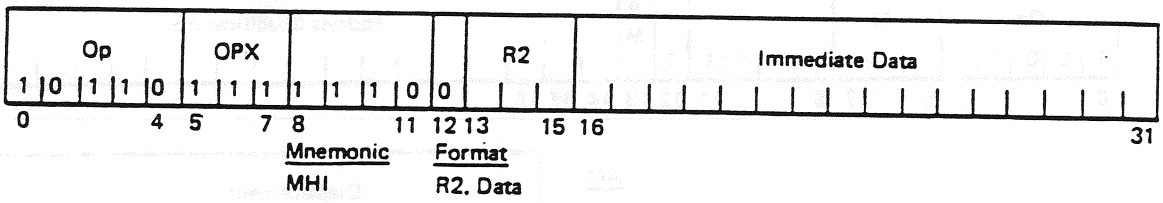
INDICATORS:

The overflow indicator is set to one when -1 is multiplied by -1. If the overflow indicator already contains a one, it is not altered by this instruction.

PROGRAM INTERRUPTS:

Fixed point overflow

4.23 MULTIPLY HALFWORD IMMEDIATE



DESCRIPTION:

Instruction bits 16 through 31 are treated as immediate data. This halfword of immediate data is the multiplier. The contents of bits 0 through 15 of general register R2 are the halfword multiplicand. The product of the multiplier and the multiplicand is a 32-bit signed fraction. Both multiplier and multiplicand are 16-bit signed twos complement fractions. This product is saved in general register R2.

RESULTING CONDITION CODE:

The code is not changed.

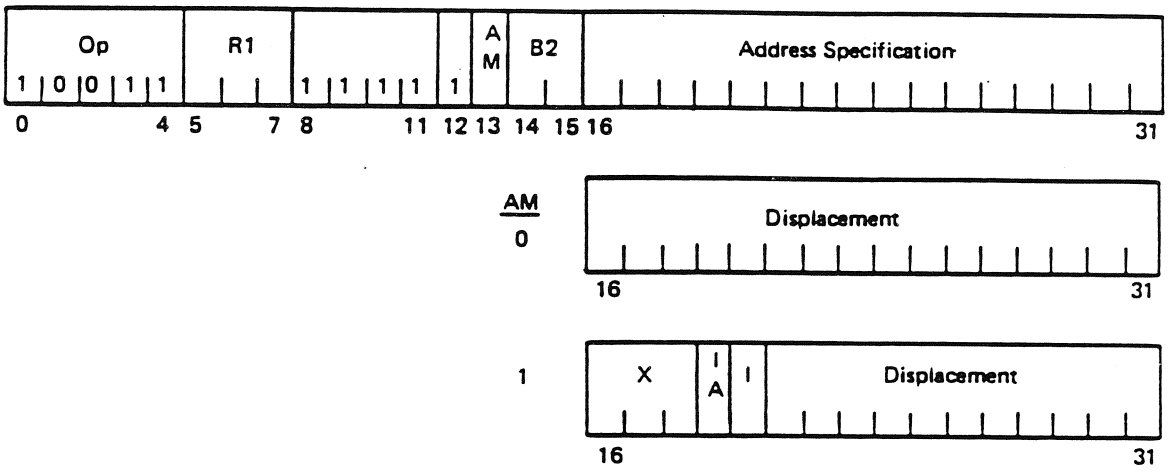
INDICATORS:

The overflow indicator is set to one when -1 is multiplied by -1. If the overflow indicator already contains a one, it is not altered by this instruction.

PROGRAM INTERRUPTS:

Fixed point overflow

4.24 MULTIPLY INTEGER HALFWORD



	AM	Mnemonic	Format
Extended:	0	MIH	R1, D2 (B2)
Indexed:	1	MIH [@] [#]	R1, D2 (X2, B2)

DESCRIPTION:

The product of the multiplier (the twos complement signed integer halfword second operand) and the twos complement signed integer halfword multiplicand (the contents of bits 0 through 15 of general register R1) replaces the multiplicand. An intermediate product is formed as a 31-bit signed integer. This product is algebraically shifted left 15 places, to form a twos complement signed halfword integer product. This halfword product replaces bits 0 through 15 of general register R1. Bits 16 through 31 of general register R1 are zeroed.

RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

The overflow indicator is set when the upper 16 bits of the intermediate product do not equal all ones or all zeroes. If the overflow indicator already contains a one, it is not altered by this instruction.

PROGRAM INTERRUPTS:

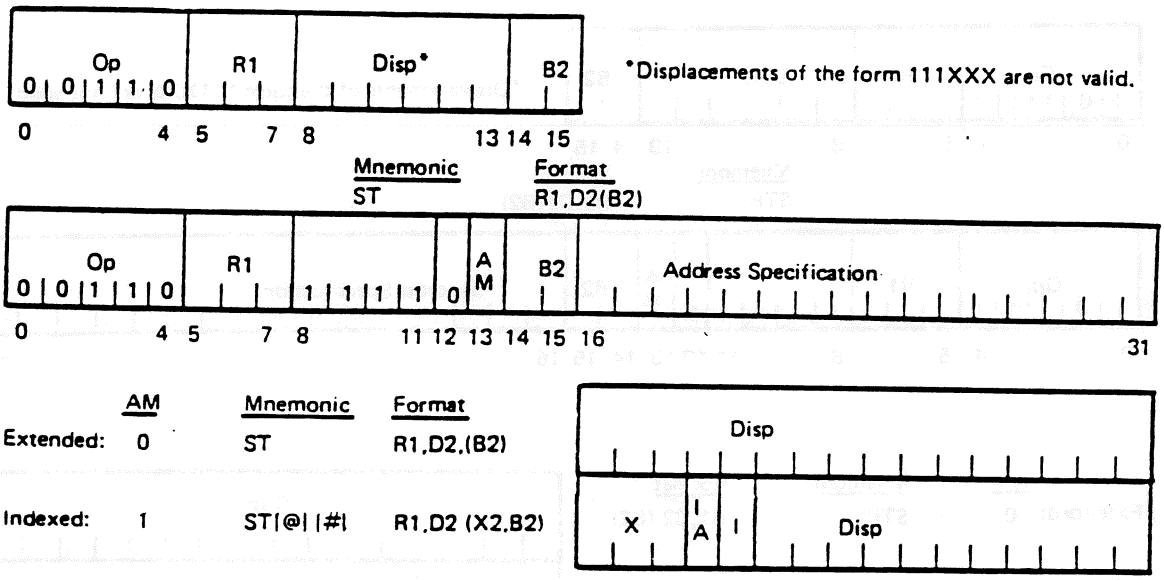
Fixed point overflow

PROGRAMMING NOTE:

If I, J, and K are halfword operands, the equation I*J+K may be solved with the following code:

```
LH      R1,I
MIH     R1,J
AH      R1,K
```

4.25 STORE



DESCRIPTION:

The contents of general register R1 are stored at the fullword second operand location. The contents of general register R1 are not changed.

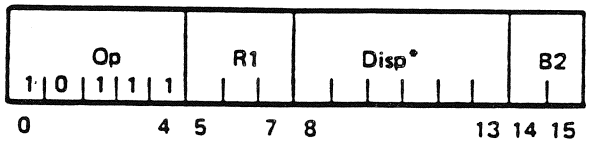
RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

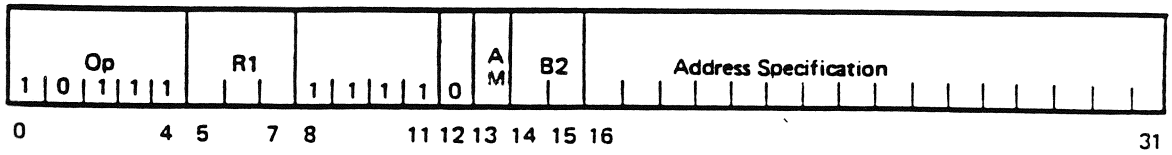
The overflow and carry indicators are not changed by this instruction.

4.26 STORE HALFWORD

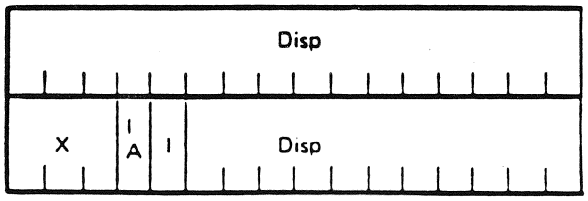


*Displacements of the form 111XXX are not valid.

Mnemonic	Format
STH	R1,D2(B2)



	AM	Mnemonic	Format
Extended:	0	STH	R1,D2 (B2)
Indexed:	1	STH [@] [#]	R1,D2 (X2,B2)



DESCRIPTION:

The most significant 16 bits (bits 0 through 15) of general register R1 are stored at the halfword second operand location. No other storage location is altered. The contents of general register R1 are not changed.

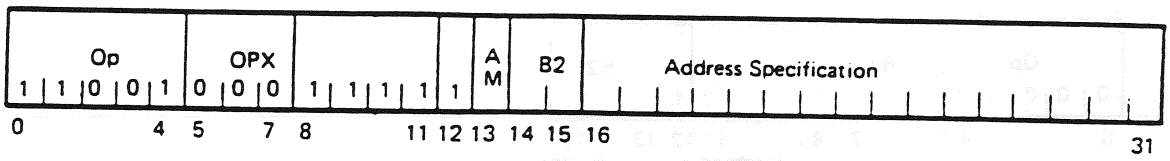
RESULTING CONDITION CODE:

The code is not changed.

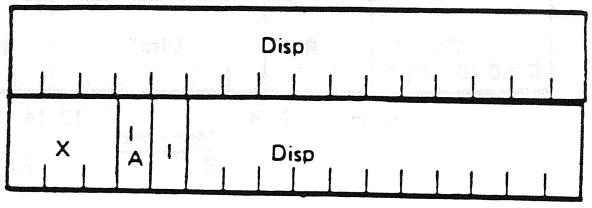
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

4.27 STORE MULTIPLE



	AM	Mnemonic	Format
Extended:	0	STM	D2(B2)
Indexed:	1	STM [@] [#]	D2(X2,B2)



DESCRIPTION:

All eight general registers are stored at the eight fullword locations starting at the fullword second operand address. The general registers are stored in ascending order.

RESULTING CONDITION CODE:

The code is not changed.

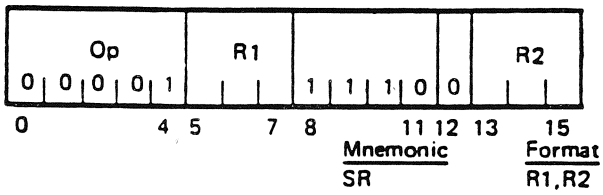
INDICATORS:

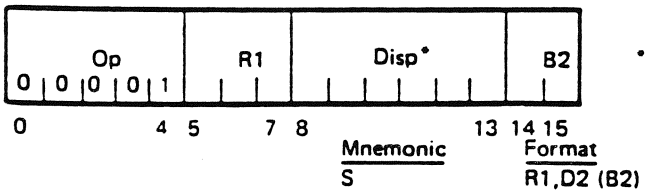
The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

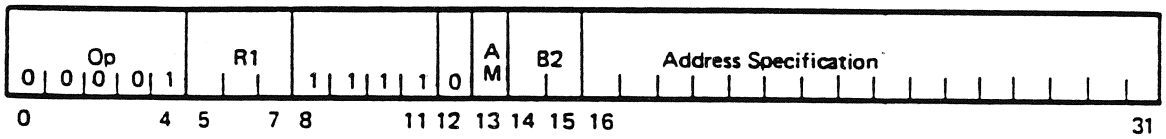
This instruction is excluded from automatic index alignment. Indexes will always specify the halfword.

4.28 SUBTRACT

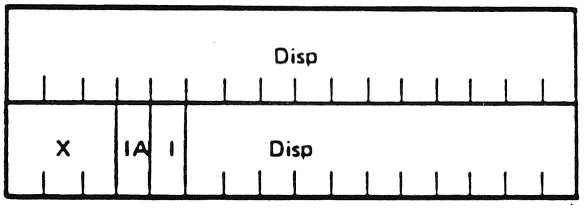




*Displacements of the form 111XXX are not valid.



	AM	Mnemonic	Format
Extended:	0	S	R1,D2 (B2)
Indexed:	1	S[@||#]	R1,D2 (X2,B2)



DESCRIPTION:

The fullword second operand is subtracted from the contents of general register R1. The result replaces the contents of general register R1. The second operand is not changed.

Subtraction is performed by adding the ones complement of the second operand and a low-order one to form the twos complement for the fullword. This fullword is added to the first operand. All 32 bits of both operands participate as in ADD. The overflow, carry, and condition code indicators reflect the result of this addition.

RESULTING CONDITION CODE:

00 The result is zero
11 The result is negative
01 The result is positive (>0)

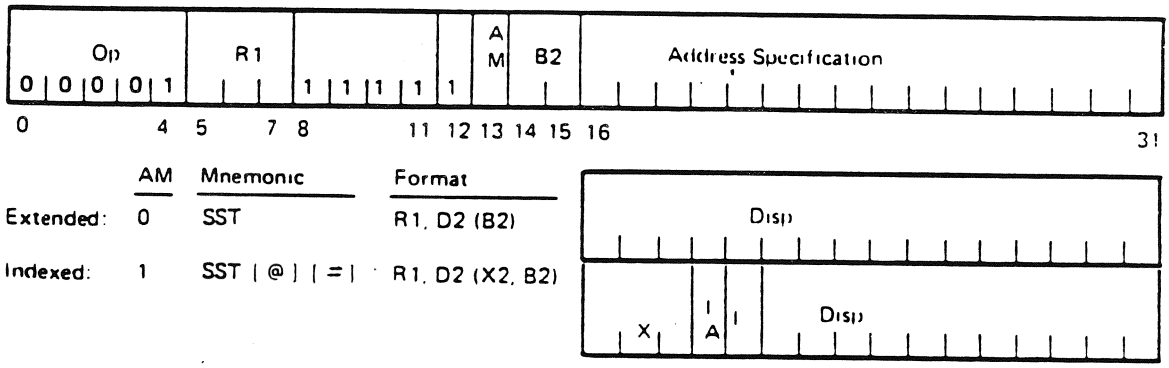
INDICATORS:

The overflow indicator is set to one if the magnitude of the difference is too large to be represented in R1; that is, greater than $1-2^{-31}$, or less than or = -1. If the overflow indicator already contains a one, it is not altered by this instruction. The carry indicator is set to indicate whether or not there is a carry out of the high-order bit position of R1.

PROGRAM INTERRUPTS:

Fixed point overflow

4.29 SUBTRACT FROM STORAGE



DESCRIPTION:

The contents of general register R1 are subtracted from the fullword second operand. The result replaces the contents of the second operand location. The first operand is not changed.

Subtraction is performed by adding the ones complement of the second operand and a low-order one to form the twos complement for the fullword. This fullword is added to the first operand. All 32 bits of both operands participate as in ADD. The overflow, carry, and condition code indicators reflect the result of this addition.

RESULTING CONDITION CODE:

00 The result is zero
11 The result is negative
01 The result is positive (>0)

INDICATORS:

The overflow indicator is set to one if the magnitude of the sum is too large to be represented in the second operand location. That is, greater than $1-2^{31}$, or less than or = -1. If the overflow indicator already contains a one, it is not altered by this instruction. The carry indicator is set to indicate whether or not there is a carry out of the high-order bit position of the result.

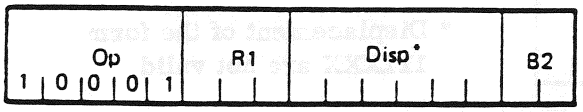
PROGRAM INTERRUPTS:

Fixed point overflow

WARNING!

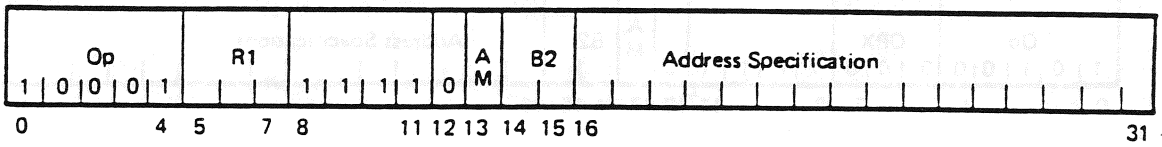
This instruction requires multiple memory accesses. The CPU does not prohibit IOP accesses of the selected main storage location during the time between the fetch of the operand and store of the result. Therefore, this instruction should not be used with any memory locations that might be DMA'd into.

4.30 SUBTRACT HALFWORD

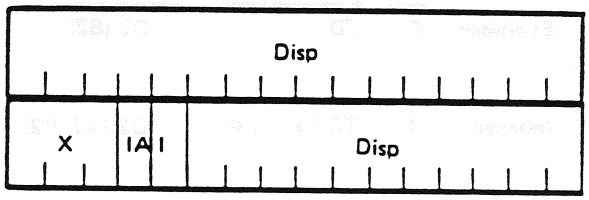


*Displacements of the form 111XXX are not valid.

Mnemonic: SH
Format: R1,D2(B2)



	AM	Mnemonic	Format
Extended:	0	SH	R1,D2,(B2)
Indexed:	1	SH [@] [#]	R1,D2(X2,B2)



DESCRIPTION:

The halfword second operand is first developed into a fullword operand by appending 16 low-order zeroes. This second operand is then subtracted from the contents of general register R1. The result replaces the contents of general register R1. The second halfword operand is not changed.

Subtraction is performed by adding the ones complement of the second operand and a low-order one to form the twos complement for the fullword. This fullword is added to the first operand. All 32 bits of both operands participate as in ADD. The overflow, carry, and condition code indicators reflect the result of this addition.

RESULTING CONDITION CODE:

00 The result is zero
11 The result is negative
01 The result is positive (>0)

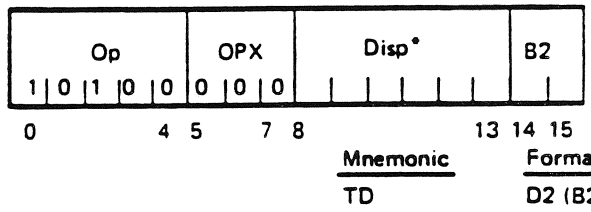
INDICATORS:

The overflow indicator is set to one if the magnitude of the sum is too large to be represented in R1; that is, greater than $1-2^{-31}$, or less than or = -1. If the overflow indicator already contains a one, it is not altered by this instruction. The carry indicator is set to indicate whether or not there is a carry out of the high-order bit position of R1.

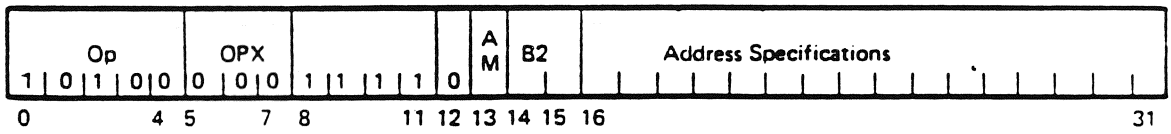
PROGRAM INTERRUPTS:

Fixed point overflow

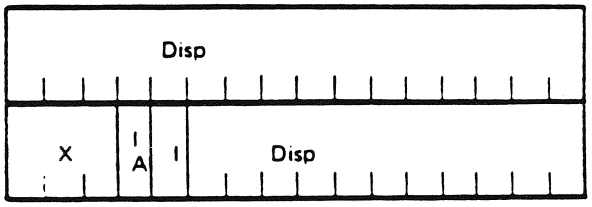
4.31 TALLY DOWN



* Displacement of the form 111XXX are not valid



	AM	Mnemonic	Format
Extended:	0	TD	D2 (B2)
Indexed:	1	TD [@] [#]	D2 (X2, B2)



DESCRIPTION:

The main storage halfword operand is decremented by one, and the result replaces the halfword operand. The contents of the general registers are not changed. Only the contents of the main storage operand are altered.

RESULTING CONDITION CODE:

00 The result is zero
11 The result is negative
01 The result is positive (>0)

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

This instruction is similar to the MODIFY STORAGE HALFWORD instruction with an implied operand of all ones.

WARNING!

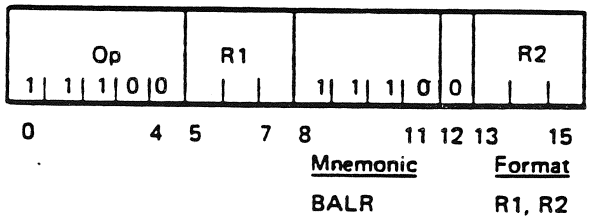
This instruction requires multiple memory accesses. The CPU does not prohibit IOP accesses of the selected main storage location during the time between the fetch of the operand and store of the result. Therefore, this instruction should not be used with any memory locations that might be DMA'd into.

5.0 BRANCHING

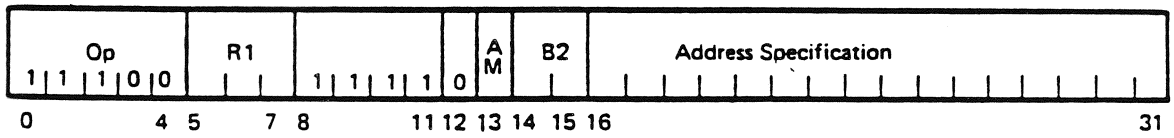
Instructions are executed, by the central processing unit, primarily in the sequential order of their locations. A departure from this normal sequential operation may occur when branching is performed. The branching instructions provide a means to make a two-way choice, to reference a subroutine, or to repeat a segment of coding.

Branching is performed by introducing a branch address as the new instruction address. The 19-bit branch address is generated as described under Expanded Addressing. Therefore, when a branch is taken, the branch address is used as the address of the next instruction. If Instruction Protection Monitor is enabled, an interrupt will occur, regardless of the branch address contents, should the branch be attempted and the destination location is not storage protected.

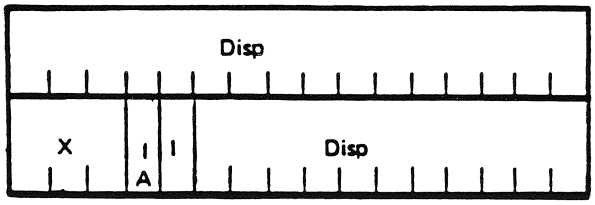
5.1 BRANCH AND LINK



Mnemonic	Format
BALR	R1, R2



	AM	Mnemonic	Format
Extended:	0	BAL	R1, D2 (B2)
Indexed:	1	BAL [@] [=]	R1, D2, (X2, B2)



DESCRIPTION:

First, the branch address is computed. Then, the first word of the current PSW (bits 0 - 31) is loaded into general register R1. Thus, the address of the next sequential instruction is preserved in register R1 (bits 0 - 15). The remaining bits of general register R1 (bits 16 - 31) will contain the condition code, the carry indicator, overflow indicator, the fixed point overflow mask, the exponent underflow mask, the significance mask, and the contents of the branch and data sector registers.

For the RR format, the branch address is contained in bits 0 through 15 of general register R2, if field R2 ≠ 0. This 16-bit branch address is expanded to a 19-bit branch address. (See Expanded Addressing.) If field R2 = 0, see programming notes.

RESULTING CONDITION CODE:

The code is not changed.

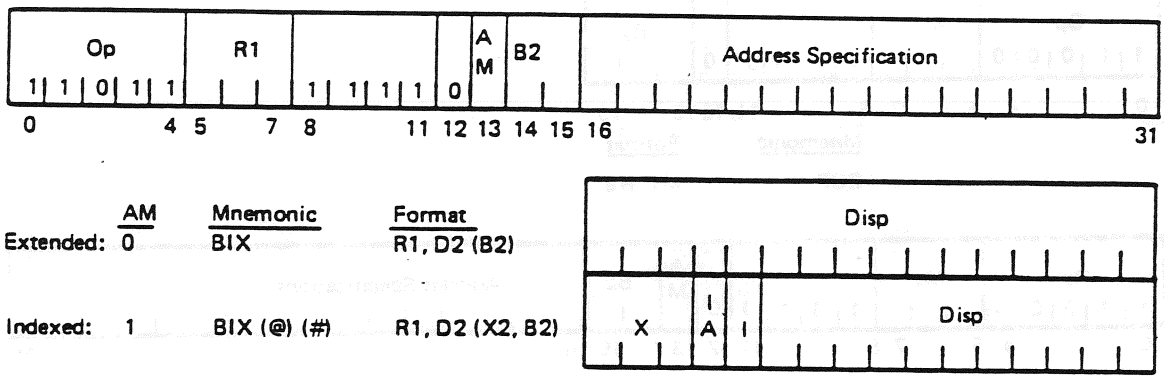
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

In the case where R2 = 0, (BALR R1, 0), no branch is taken.

5.2 BRANCH AND INDEX



	AM	Mnemonic	Format
Extended:	0	BIX	R1, D2 (B2)
Indexed:	1	BIX (@) (#)	R1, D2 (X2, B2)

DESCRIPTION:

Bits 0 through 15 of the general register specified by R1 contain an index. Bits 16 through 31 of general register R1 contain a count. An effective address is computed in the normal manner for the extended class. (For the indexed addressing mode, the fullword indirect address pointer must contain zeros in bit locations 22 and 23.) Next, the index is incremented by one. Then the count is decremented by one. If the count prior to update is greater than zero, a branch to the effective address is taken. If the count prior to update is less than or equal to zero, no branch occurs.

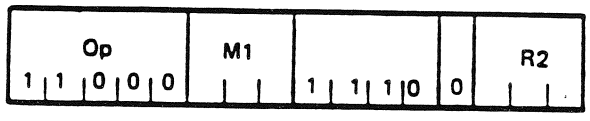
RESULTING CONDITION CODE:

The code is not changed.

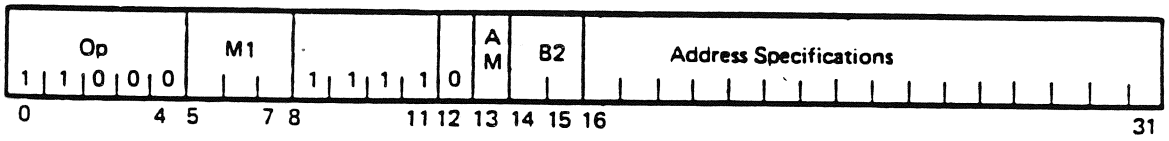
INDICATORS:

The carry and overflow indicators are not changed by this instruction.

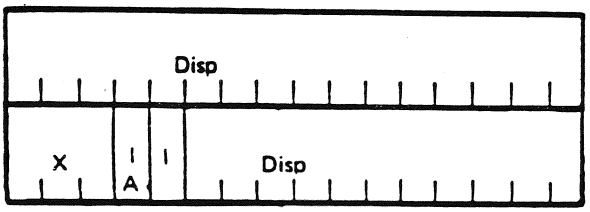
5.3 BRANCH ON CONDITION



Mnemonic	Format
BCR	M1, R2



	AM	Mnemonic	Format
Extended:	0	BC	M1, D2 (B2)
Indexed:	1	BC [@] [≠]	M1, D2 (X2, B2)



DESCRIPTION:

This instruction tests the PSW condition code status bits. Instruction bits 5 through 7 (the M1 field) specify which condition code (bits 16 and 17 of the PSW) is to be tested. Instruction bit 5 tests for a code equal 00, instruction bit 6 tests for a code equal 11, and instruction bit 7 tests for a code equal 01. Whenever the condition code test is successful, the branch is taken. Thus, when more than one bit of the M1 field is a one, the branch is taken for any successful test (e.g., M1 = 111 always branches, M1 = 000 never branches).

The branch address is contained in bits 0 through 15 of general register R2 for the RR format. This 16-bit branch address is expended to a 19-bit branch address. (See Expanded Addressing.)

RESULTING CONDITION CODE:

The condition code was set following all arithmetic, logical, test and compare instructions, and otherwise remains unchanged unless the program status word is altered. The code is not changed by this instruction.

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

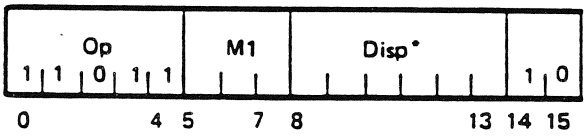
PROGRAMMING NOTES:

The result and test conditions are shown as follows:

	M1 Field (Test)		
	(5)	(6)	(7)
Arithmetic and Tally			
Zero	1	0	0
Negative	0	1	0
Positive (>0)	0	0	1
Logical			
Zero	1	0	0
Not Zero	0	1	0
Test			
Zero	1	0	0
Mixed	0	1	0
All ones	0	0	1
Compare			
Equal	1	0	0
$O_1 < O_2$	0	1	0
$O_1 > O_2$	0	0	1

It is possible to combine tests. For example, following the MSTH instruction, an M1 field of 1 0 1 specified branch on nonnegative (zero or positive).

5.4 BRANCH ON CONDITION BACKWARD



*Displacements of the form 111XXX are not valid.

Mnemonic	Format
BCB	M1, D2

DESCRIPTION:

This instruction tests the PSW condition code status bits. Instruction bits 5 through 7 (the M1 field) specify which condition code (bits 16 and 17 of the PSW) is to be tested. Instruction bit 5 tests for a code equal 00, instruction bit 6 tests for a code equal 11, and instruction bit 7 tests for a code equal 01. Whenever the condition code test is successful, the branch is taken by subtracting the Disp from the updated IC. Thus, when more than one bit of the M1 field is a one, the branch is taken for any successful test (e.g., M1 = 111 always branches).

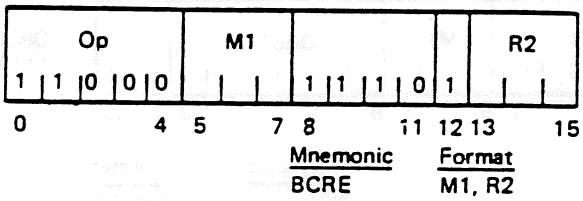
RESULTING CONDITION CODE:

The condition code was set following all arithmetic, logical, test, and compare instructions, and otherwise remains unchanged unless the program status word is altered. The code is not changed by this instruction.

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

5.5 BRANCH ON CONDITION (EXTENDED)



DESCRIPTION:

This instruction tests the PSW condition code status bits. Instruction bits 5 through 7 (the M1 field) specify which condition code (bits 16 and 17 of the PSW) is to be tested. Instruction bit 5 tests for a code equal 00, instruction bit 6 tests for a code equal 11, and instruction bit 7 tests for a code equal 01. Whenever the condition code test is successful, the branch is taken. Thus, when more than one bit of the M1 field is a one, the branch is taken for any successful test (e.g., M1 = 111 always branches).

When the branch is taken, PSW bits 0 through 15 and 24 through 31 are replaced by corresponding bits in general register R2.

RESULTING CONDITION CODE:

The condition code was set following all arithmetic, logical test, and compare instructions, and otherwise remains unchanged unless the program status word is altered. The code is not changed by this instruction.

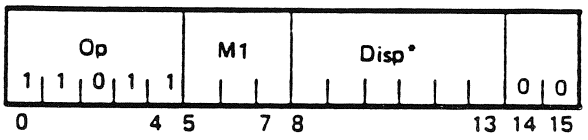
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

This instruction is similar to the RR version of the BRANCH ON CONDITION instruction. It is provided to facilitate subroutine returns across sector boundaries after general register R2 has been initialized by the use of the BRANCH AND LINK instruction.

5.6 BRANCH ON CONDITION FORWARD



*Displacements of the form 111XXX are not valid.

Mnemonic	Format
BCF	M1, D2

DESCRIPTION:

This instruction tests the PSW condition code status bits. Instruction bits 5 through 7 (the M1 field) specify which condition code (bits 16 and 17 of the PSW) is to be tested. Instruction bit 5 tests for a code equal 00, instruction bit 6 tests for a code equal 11, and instruction bit 7 tests for a code equal 01. Whenever the condition code test is successful, the branch is taken by adding the Disp to the updated IC. Thus, when more than one bit of the M1 field is a one, the branch is taken for any successful test (e.g., M1 = 111 always branches).

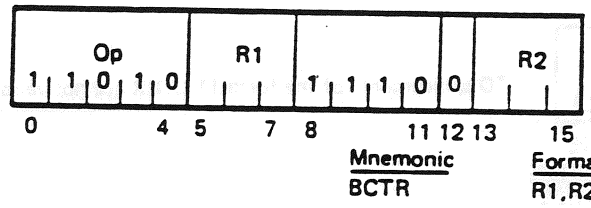
RESULTING CONDITION CODE:

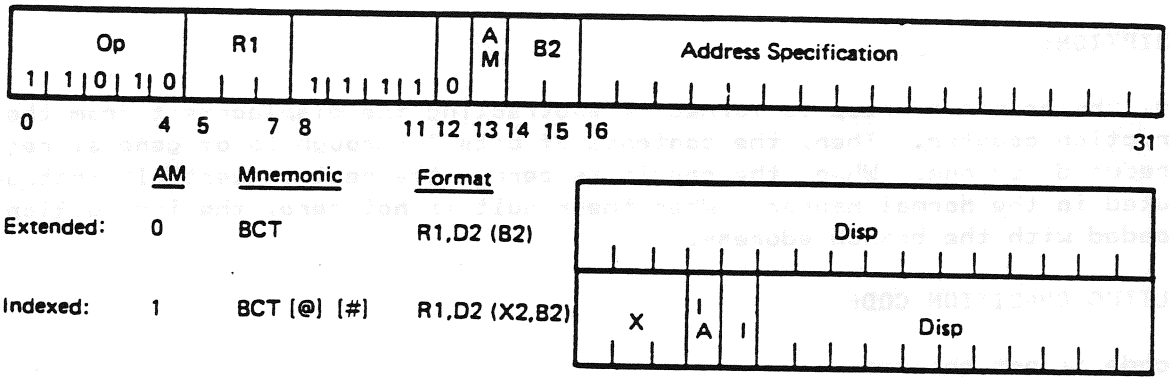
The condition code was set following all arithmetic, logical, test, and compare instructions, and otherwise remains unchanged unless the program status word is altered. The code is not changed by this instruction.

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

5.7 BRANCH ON COUNT





DESCRIPTION:

First, the branch address is computed. The branch address is contained in bits 0 through 15 of general register R2 for the RR format. This 16-bit branch address is expanded to a 19-bit branch address. (See Expanded Addressing.)

Then, the contents of bits 0 through 15 of general register R1 are reduced by one. When the result is zero, the next sequential instruction is executed in the normal manner. When the result is not zero, the instruction counter is loaded with the branch address.

RESULTING CONDITION CODE:

The code is not changed.

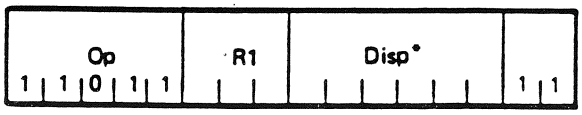
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

An initial count of one results in zero, and no branch takes place. An initial count of zero results in a minus one and causes branching to be executed. The low-order 16 bits of R1 do not participate in the count or zero test.

5.8 BRANCH ON COUNT BACKWARD



*Displacements of the form 111XXX are not valid.

Mnemonic	Format
BCTB	R1,D2

DESCRIPTION:

First, the branch address is formed by subtracting the displacement from the updated instruction counter. Then, the contents of bits 0 through 15 of general register R1 are reduced by one. When the result is zero, the next sequential instruction is executed in the normal manner. When the result is not zero, the instruction counter is loaded with the branch address.

RESULTING CONDITION CODE:

The code is not changed.

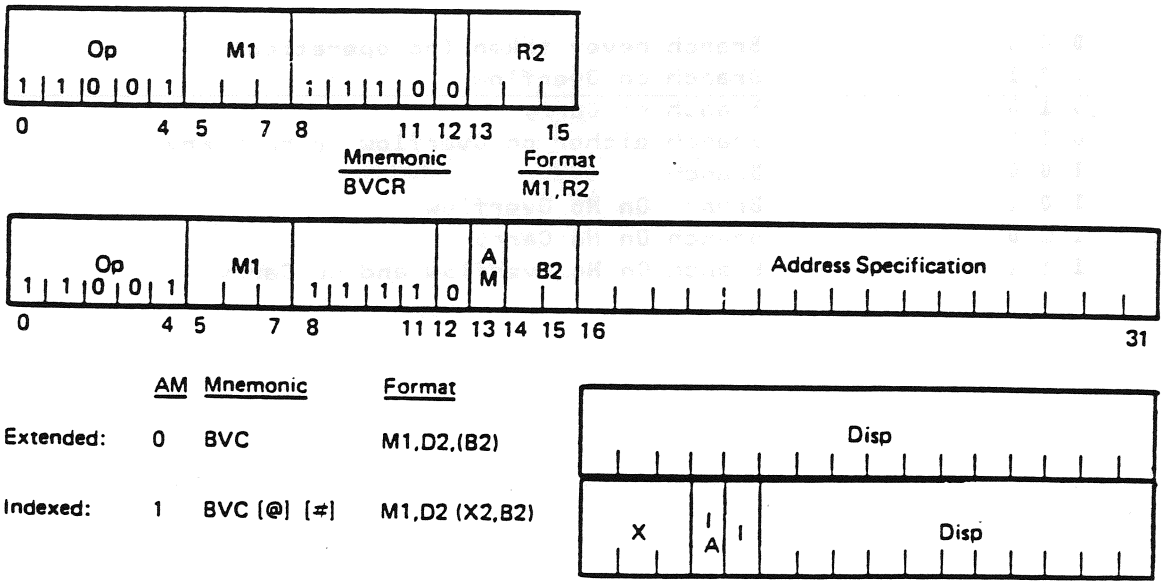
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

An initial count of one results in zero, and no branch takes place. An initial count of zero results in a minus one and causes branching to be executed. The low-order 16 bits do not participate in the count or zero test.

5.9 BRANCH ON OVERFLOW AND CARRY



DESCRIPTION:

This instruction tests the PSW overflow and carry indicator status bits. The M1 field, instruction bits 5 through 7 specifies the test. Instruction bit 6 is tested against PSW bit 18 (carry), and instruction bit 7 is tested against PSW bit 19 (overflow). Whenever a specified bit of the PSW is a one, the test is successful and the branch is taken. Thus, when both indicators are tested by M1 = 011, the branch is taken if either indicator contains a one. A one in instruction bit 5 inverts the logic, causing bits 6 and 7 to test the PSW bits for zero.

For the RR format, the branch address is contained in bits 0 through 15 of general register R2. This 16-bit branch address is expanded to a 19-bit branch address. (See Expanded Addressing.)

RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

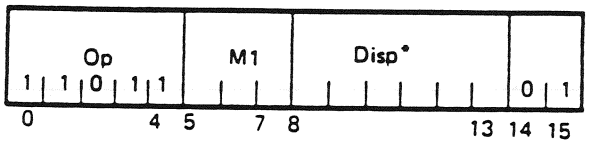
The overflow indicator is set 0 by this instruction. The carry indicator is not changed by this instruction.

PROGRAMMING NOTES:

The possible combinations of test conditions are shown as follows:

M1 Field			Test Conditions
5	6	7	
0	0	0	Branch never taken (no operation)
0	0	1	Branch on Overflow
0	1	0	Branch on Carry
0	1	1	Branch either on Overflow or on Carry
1	0	0	Branch
1	0	1	Branch On No Overflow
1	1	0	Branch On No Carry
1	1	1	Branch On No Overflow and No Carry

5.10 BRANCH ON OVERFLOW AND CARRY FORWARD



*Displacements of the form 111XXX are not valid.

Mnemonic	Format
BVCF	M1, D2

DESCRIPTION:

This instruction tests the PSW overflow and carry indicator status bits. Instruction bits 5 through 7 specify the test. Instruction bit 6 is tested against PSW bit 18, and instruction bit 7 is tested against PSW bit 19. Whenever a specified bit of the PSW is a one, the test is successful and the branch is taken by adding the Disp to the updated IC. Thus, when both indicators are tested by M1 = 011, the branch is taken if either indicator contains a one. A one in instruction bit 5 inverts the logic, causing bits 6 and 7 to test the PSW bits for zero.

The branch address is formed by adding the displacement to the updated instruction counter.

RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

The overflow indicator is set 0 by this instruction. The carry indicator is not changed by this instruction.

PROGRAMMING NOTES:

The possible combinations of test conditions are shown as follows:

M1 Field 5 6 7	Test Conditions
0 0 0	Branch never taken (no operation)
0 0 1	Branch on Overflow
0 1 0	Branch on Carry
0 1 1	Branch either on Overflow or on Carry
1 0 0	Branch
1 0 1	Branch On No Overflow
1 1 0	Branch On No Carry
1 1 1	Branch On No Overflow and No Carry

(This page intentionally left blank)

6.0 SHIFT OPERATIONS

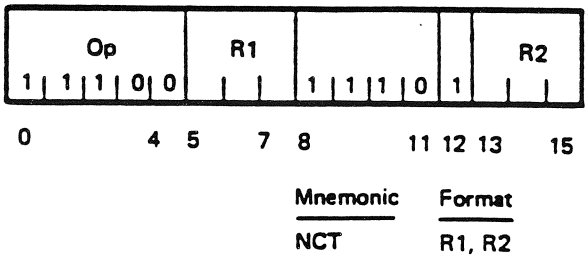
Shift instructions use the halfword format. The shift count is defined by the count field, as shown in Figure 6-1.

Instruction Bits 8-13	Shift Count Determined By
000000 (Zero)	No Operation
000001-110111 (1-55)	Instruction bits 8 through 13
111000 (56)	Bits 10 - 15 of general register 0
111001 (57)	Bits 10 - 15 of general register 1
111010 (58)	Bits 10 - 15 of general register 2
111011 (59)	Bits 10 - 15 of general register 3
111100 (60)	Bits 10 - 15 of general register 4
111101 (61)	Bits 10 - 15 of general register 5
111110 (62)	Bits 10 - 15 of general register 6
111111 (63)	Bits 10 - 15 of general register 7

Figure 6-1. Shift Count

If the shift count is 56 through 63, bits 10 through 15 of the corresponding general register (0 through 7) designate the shift count. When specified using the count field, the maximum shift count allowed for shift operations is 55. Shifts of up to 63 positions are allowed, when general register 0 through 7 is used to specify a computed shift.

6.1 NORMALIZE AND COUNT



DESCRIPTION:

First, all bits (0 through 31) of general register R1 are set to zero. For each position that the contents of general register R2 are shifted to the left, the high-order half of general register R1 (bits 0 through 15) is incremented by 1. The shift terminates when bit position 0 ≠ bit position 1 of general register R2. If the contents of general register R2 are initially zero, a count of zero is entered in general register R1. Zeros are entered into the vacated low-order bits of general register R2. Upon completion of this instruction, the count is contained in bits 0 through 15 of general register R1.

RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

The carry indicator will be zero at the end of the operation, if general register R2 contains zero. The carry indicator will be one at the end of the operation, if the shift is terminated by the detection of bit position one not equal to bit position 0 of the general register R2. The overflow indicator is not changed by this instruction.

PROGRAMMING NOTES:

If the initial condition of general register R2 was such that bit position 0 is not equal to bit position 1, the count in the high-order bit of general register R1 is zero, the carry indicator is one, and there is no shift. If the initial condition of R2 was all ones, the count is 31, the carry is one and R2 contains 80000000.

This instruction is executed as shown below in Figure 6-2.

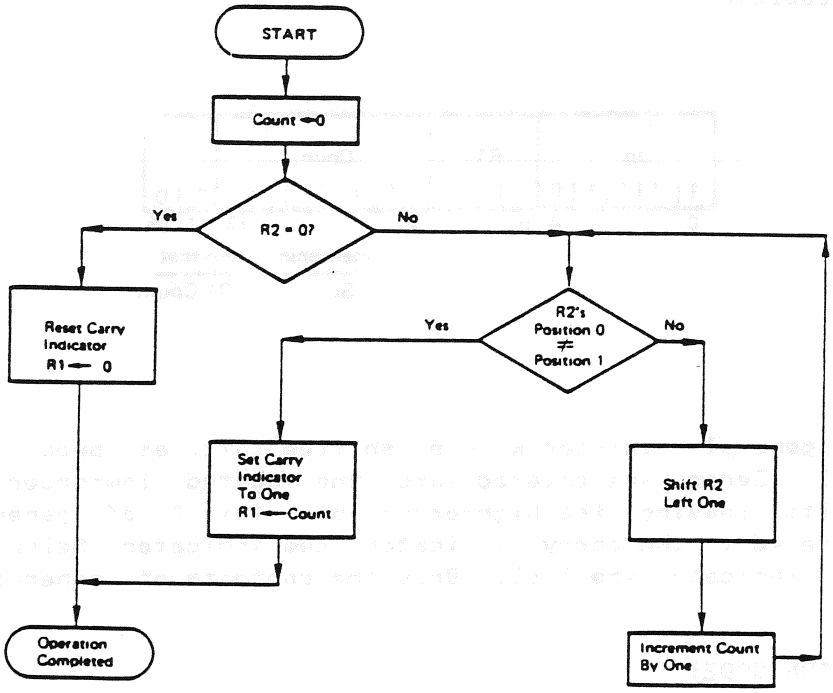
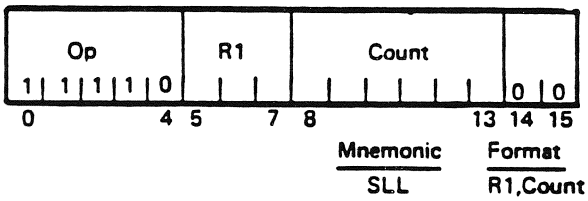


Figure 6-2. Normalize and Count Execution

6.2 SHIFT LEFT LOGICAL



DESCRIPTION:

The contents of general register R1 are shifted left, as specified by the shift count Figure 6-1. Zeros are entered into the vacated low-order bits of general register R1. Bits leaving the high-order bit (bit 0 of general register R1) position are entered in the carry indicator (see indicators below). Bits shifted out of the carry indicator are lost. Only the contents of general register R1 are changed.

RESULTING CONDITION CODE:

The code is not changed.

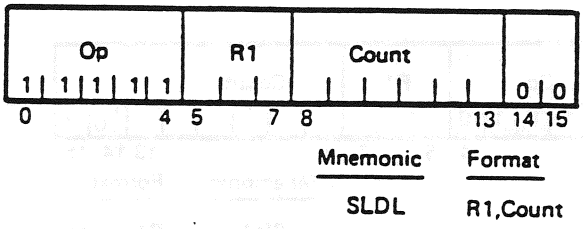
INDICATORS:

The carry indicator is set to one for each one, and to zero for each zero, shifted left from the high-order position of general register R1. The overflow indicator is not changed by this instruction.

PROGRAMMING NOTES:

When the shift count n is greater than 31, then the result of the shift of general register R1 is zero.

6.3 SHIFT LEFT DOUBLE LOGICAL



DESCRIPTION:

The contents of the pair of general registers (R1 and (R1+1)mod8) are shifted left as a 64-bit register. The number of positions shifted is specified by the shift count. Bits shifted out of bit position zero of general register (R1 + 1)mod8, are entered into bit position 31 of general register R1. Zeros are entered into the vacated low-order bits of general register (R1 + 1)mod8. Bits leaving the high-order bit position (bit position 0 of general register R1) are shifted into the carry indicator. Bits shifted out of the carry indicator are lost.

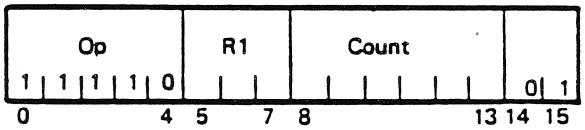
RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

The carry indicator is set to one for each one, and to zero for each zero, shifted left from the high-order bit position of general register R1. The overflow indicator is not changed by this instruction.

6.4 SHIFT RIGHT ARITHMETIC



Mnemonic	Format
SRA	R1,Count

DESCRIPTION:

The contents of general register R1 are shifted right the number of places indicated by the shift count. Bits equal to the sign are entered into vacated high-order bit positions. Bits shifted out of bit position 31 of general register R1 are lost.

RESULTING CONDITION CODE:

The code is not changed.

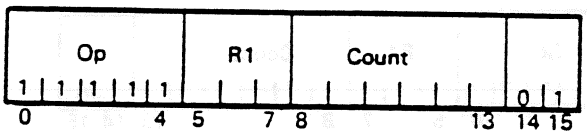
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

A shift right of n is equivalent to dividing the contents of general register R1 by 2^n.

6.5 SHIFT RIGHT DOUBLE ARITHMETIC



Mnemonic	Format
SRDA	R1,Count

DESCRIPTION:

The contents of the pair of general registers (R1 and (R1+1)mod8) are shifted right as a 64-bit register. The number of positions shifted is specified by the shift count. Bits shifted out of bit position 31 of general register R1, are entered into bit position 0 of general register (R1 + 1)mod8. Bits equal to the sign are entered into vacated high-order bit positions. Bits shifted out of bit position 31 of general register (R1 + 1)mod8 are lost.

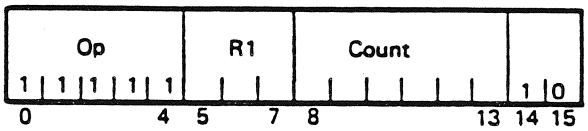
RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

6.6 SHIFT RIGHT DOUBLE LOGICAL



Mnemonic	Format
SRDL	R1,Count

DESCRIPTION:

The contents of the pair of general registers (R1 and (R1+1)mod8) are shifted right as a 64-bit register. The number of positions shifted is specified by the shift count. Zeros are entered into all vacated high-order bit positions. Bits shifted out of bit position 31 of general register R1, are entered into bit position 0 of general register (R1 + 1)mod8. Bits shifted out of bit position 31 of general register (R1 + 1)mod8 are lost.

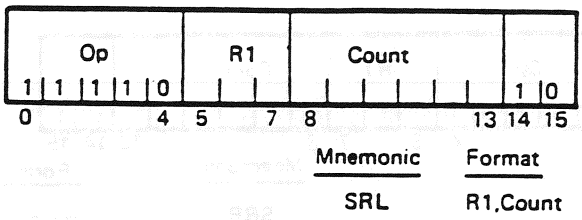
RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

6.7 SHIFT RIGHT LOGICAL



DESCRIPTION:

The contents of general register R1 are shifted right the number of places indicated by the shift count. Zeros are entered into all vacated high-order bit positions. Bits shifted out of bit position 31 of general register R1 are lost.

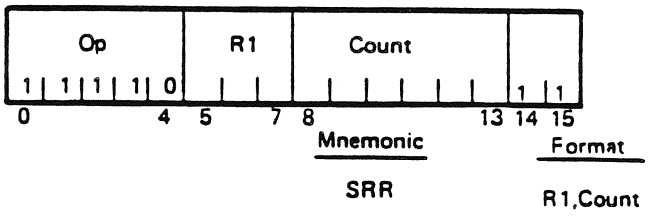
RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

6.8 SHIFT RIGHT AND ROTATE



DESCRIPTION:

The contents of general register R1 are shifted right the number of places indicated by the shift count. Bits shifted out of bit position 31 are entered into bit position 0. The general register thus becomes a circular register and no bits are lost.

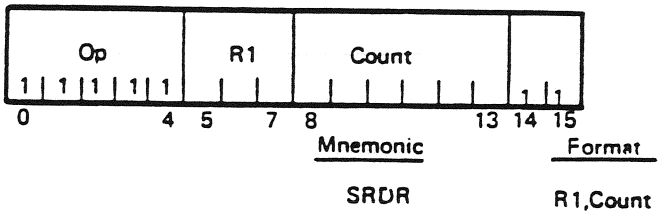
RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

6.9 SHIFT RIGHT DOUBLE AND ROTATE



DESCRIPTION:

The contents of the pair of general registers (R1 and (R1+1)mod8) are shifted right as a 64-bit register. The number of positions shifted is specified by the shift count. Bits shifted out of bit position 31 of general register R1 are entered into bit position 0 of general register (R1+1)mod8. Bits shifted out of bit position 31 of general register (R1+1)mod8 are entered into bit position 0 of general register R1. Thus, the two registers become a single, circular, 64-bit register, and no bits are lost.

RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

When the shift count equals 32, the contents of general register R1 and (R1+1)mod8 are exchanged.

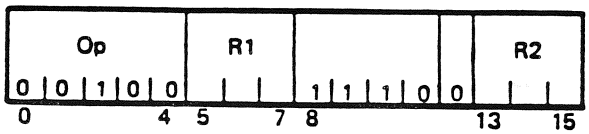
(This page intentionally left blank)

7.0 LOGICAL OPERATIONS

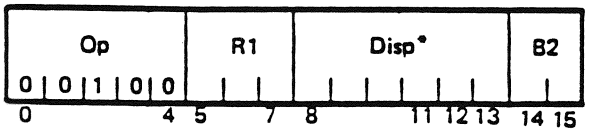
A set of instructions is provided for the logical manipulation of data. Fullword operands consist of 32 bits. Halfword immediate and storage operands are developed into fullword operands by appending 16 low-order zeros. The sign position is treated in the same manner as any other position.

There is no interdependence between bits for logical operations; that is, the result in position i is independent of bit j in either operand when i is not equal to j.

7.1 AND

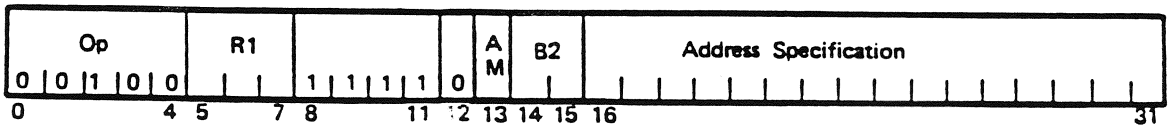


Mnemonic	Format
NR	R1,R2

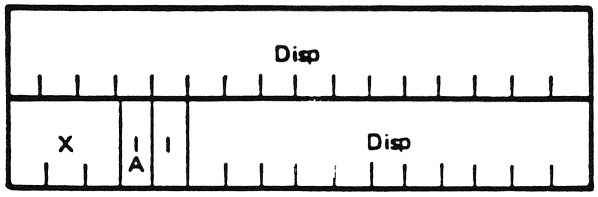


Mnemonic	Format
N	R1,D2 (B2)

* Displacements of the form 111XXX are not valid.



	AM	Mnemonic	Format
Extended:	0	N	R1,D2 (B2)
Indexed:	1	N [@] [#]	R1,D2 (X2, B2)



DESCRIPTION:

The logical product (AND), of the fullword second operand and the contents of general register R1, is formed bit-by-bit. The result replaces the contents of general register R1. The second operand is not changed. The following table defines the AND operation.

AND	
Storage	1 1 0 0
R1	1 0 1 0
Result	1 0 0 0

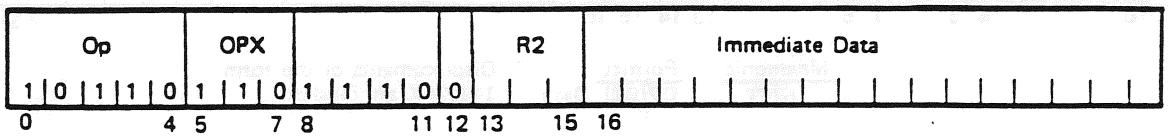
RESULTING CONDITION CODE:

00 The result is zero
11 The result is not zero

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

7.2 AND HALFWORD IMMEDIATE



Mnemonic	Format
NHI	R2,Data

DESCRIPTION:

Instruction bits 16 through 31 are treated as immediate data. The halfword immediate data is first developed into a fullword by appending 16 low-order zeros. The logical product (AND), of this fullword operand and the contents of general register R2, is formed bit-by-bit. The result replaces the contents of general register R2. The immediate operand is not changed. The following table defines the AND operation.

AND	
Immediate Data	1 1 0 0
R2	1 0 1 0
Result	1 0 0 0

RESULTING CONDITION CODE:

00 The result is zero
11 The result is not zero

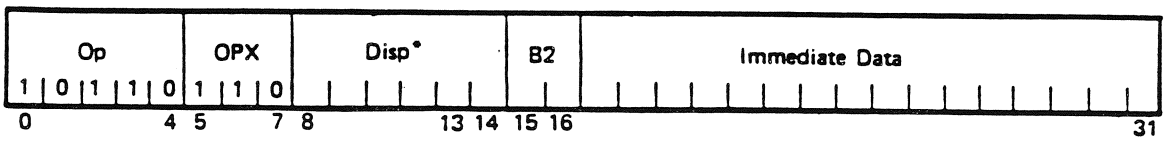
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

The least significant 16 bits of the result (bits 16 through 31) will always be zero.

7.3 AND IMMEDIATE WITH STORAGE



Mnemonic	Format	* Displacements of the form
NIST	D2(B2), Data	111XXX are invalid.

DESCRIPTION:

Bits 16 through 31 of this instruction are treated as halfword immediate data. The logical product (AND), of this immediate data and the halfword main storage operand, is formed bit-by-bit. The result replaces the halfword main storage operand.

RESULTING CONDITION CODE:

00 The result is zero
11 The result is not zero

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

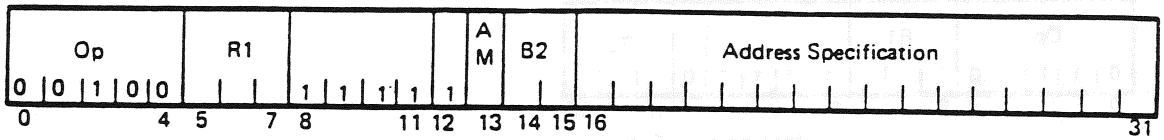
PROGRAMMING NOTES:

The zero bits in the immediate data specify the bits of the halfword first operand that are set to zero. Zero bits in the halfword main storage operand remain unaltered.

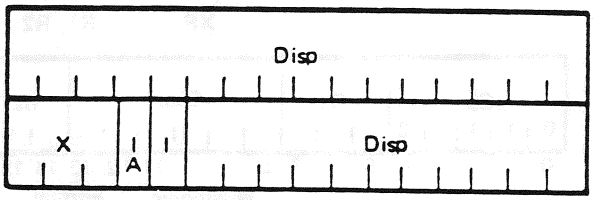
WARNING!

This instruction requires multiple memory accesses. The CPU does not prohibit IOP accesses of the selected main storage location during the time between the fetch of the operand and store of the result. Therefore, this instruction should not be used with any memory locations that might be DMA'd into.

7.4 AND TO STORAGE



	AM	Mnemonic	Format
Extended:	0	NST	R1,D2(B2)
Indexed:	1	NST [@] [#]	R1,D2(X2,B2)



DESCRIPTION:

The logical product (AND), of the fullword second operand and the contents of general register R1, is formed bit-by-bit. The result replaces the second operand. The contents of the general register are not changed. The following table defines the AND operation.

AND	
Storage	1 1 0 0
R1	1 0 1 0
Result	1 0 0 0

RESULTING CONDITION CODE:

00 The result is zero
11 The result is not zero

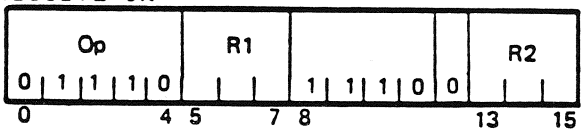
INDICATORS:

The overlow and carry indicators are not changed by this instruction.

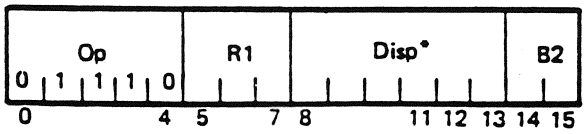
WARNING!

This instruction requires multiple memory accesses. The CPU does not prohibit IOP accesses of the selected main storage location during the time between the fetch of the operand and store of the result. Therefore, this instruction should not be used with any memory locations that might be DMA'd into.

7.5 EXCLUSIVE OR

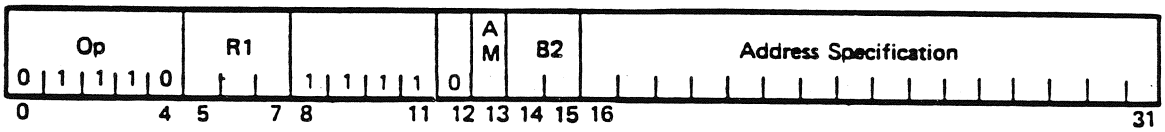


Mnemonic	Format
XR	R1, R2

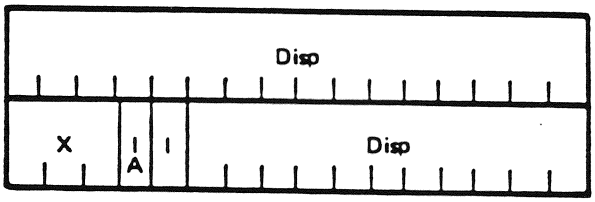


Mnemonic	Format
X	R1,D2(B2)

* Displacements of the form 111XXX are not valid.



	AM	Mnemonic	Format
Extended:	0	X	R1,D2(B2)
Indexed:	1	X [@] [#]	R1,D2(X2,B2)



DESCRIPTION:

The modulo-two sum (Exclusive OR), of the fullword second operand and the contents of general register R1, is formed bit-by-bit. The result replaces the contents of general register R1. The second operand is not changed. The following table defines the Exclusive OR operation.

Exclusive OR	
Storage	1 1 0 0
R1	1 0 1 0
Result	0 1 1 0

RESULTING CONDITION CODE:

00 The result is zero
11 The result is not zero

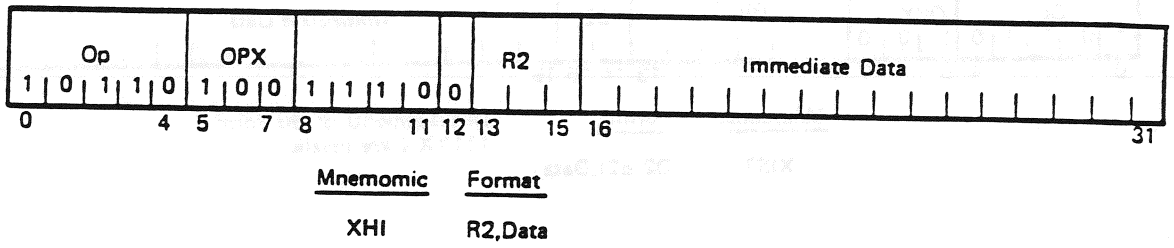
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

The ones complement of the general register is obtained when the second operand contains all ones.

7.6 EXCLUSIVE OR HALFWORD IMMEDIATE



Mnemomic	Format
XHI	R2,Data

DESCRIPTION:

Instruction bits 16 though 31 are treated as immediate data. The halfword of immediate data is first developed into a fullword by appending 16 low-order zeros. The modulo-two sum (Exclusive OR), of this fullword operand and contents of general register R2, is formed bit-by-bit. The result replaces the contents of general register R2. The immediate operand is not changed. The following table defines the Exclusive OR operation.

Exclusive OR	
Immediate Data	1 1 0 0
R2	1 0 1 0
Result	0 1 1 0

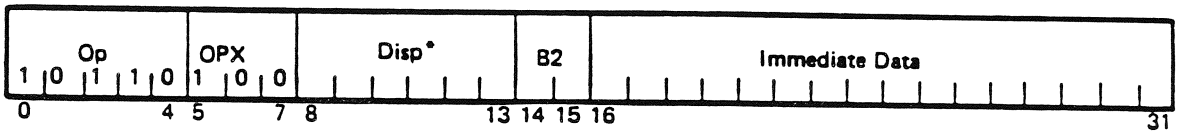
RESULTING CONDITION CODE:

00 The result is zero
11 The result is not zero

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

7.7 EXCLUSIVE OR IMMEDIATE WITH STORAGE



Mnemonic	Format
XIST	D2(B2),Data

* Displacements of the form 111XXX are invalid.

DESCRIPTION:

Bits 16 through 31 of this instruction are treated as halfword immediate data. The modulo-two sum (Exclusive OR), of this halfword immediate data and the halfword main storage operand, is formed bit-by-bit. The result replaces the halfword main storage operand.

RESULTING CONDITION CODE:

00 The result is zero
11 The result is not zero

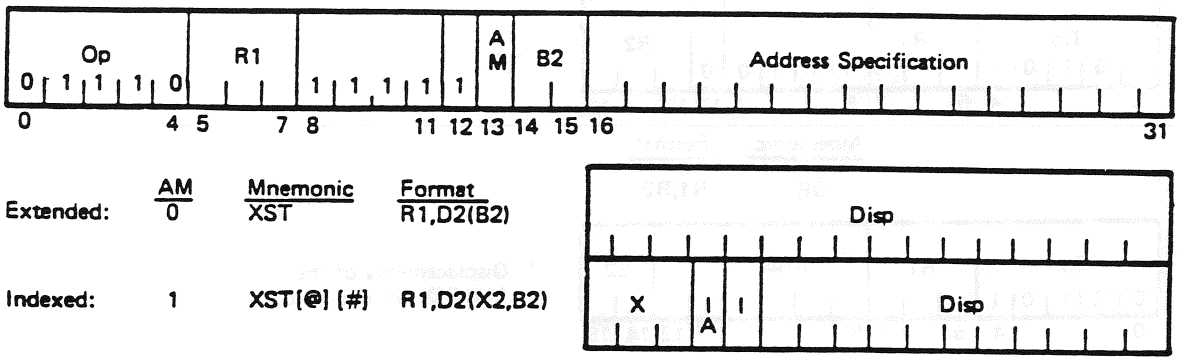
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

WARNING!

This instruction requires multiple memory accesses. The CPU does not prohibit IOP accesses of the selected main storage location during the time between the fetch of the operand and store of the result. Therefore, this instruction should not be used with any memory locations that might be DMA'd into.

7.8 EXCLUSIVE OR TO STORAGE



DESCRIPTION:

The modulo-two sum (Exclusive OR), of the fullword second operand and the contents of general register R1, is formed bit-by-bit. The result replaces the second operand. The contents of the general register are not changed. The following table defines the Exclusive OR operation.

Exclusive OR	
Storage	1 1 0 0
R1	1 0 1 0
Result	0 1 1 0

RESULTING CONDITION CODE:

00 The result is zero
11 The result is not zero

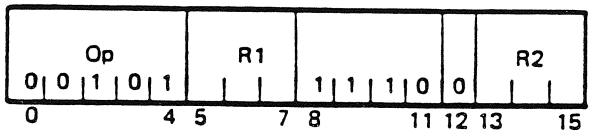
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

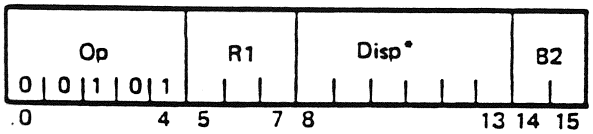
WARNING!

This instruction requires multiple memory accesses. The CPU does not prohibit IOP accesses of the selected main storage location during the time between the fetch of the operand and store of the result. Therefore, this instruction should not be used with any memory locations that might be DMA'd into.

7.9 OR

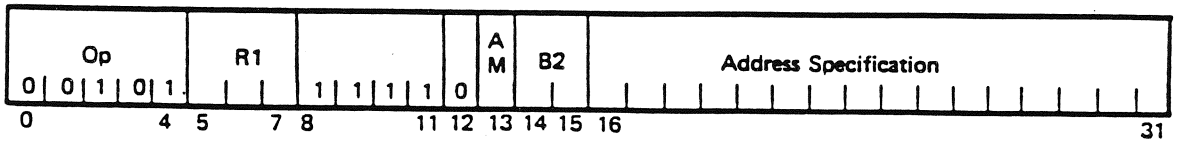


Mnemonic	Format
OR	R1,R2

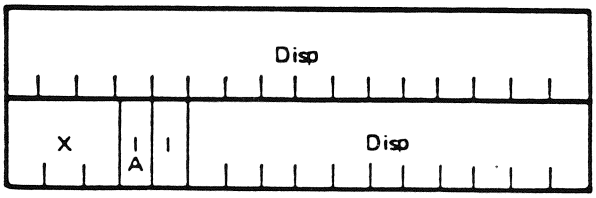


Mnemonic	Format
0	R1,D2(B2)

* Displacements of the form 111XXX are not valid.



	AM	Mnemonic	Format
Extended:	0	0	R1,D2(B2)
Indexed:	1	0 [@] [#]	R1,D2(X2,B2)



DESCRIPTION:

The logical sum (OR), of the fullword second operand and the contents of general register R1, is formed bit-by-bit. The result replaces the contents of general register R1. The second operand is not changed. The following table defines the OR operation.

OR	
Storage	1 1 0 0
R1	1 0 1 0
Result	1 1 1 0

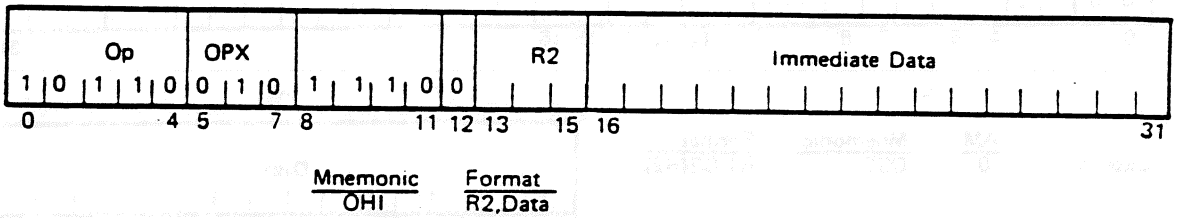
RESULTING CONDITION CODE:

00 The result is zero
11 The result is not zero

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

7.10 OR HALFWORD IMMEDIATE



DESCRIPTION:

Instruction bits 16 through 31 are treated as immediate data. The halfword of immediate data is first developed into a fullword operand by appending 16 low-order zeroes. The logical sum (OR), of the fullword operand and the contents of general register R2, is formed bit-by-bit. The result replaces the contents of general register R2. The immediate operand is not changed. The following table defines the OR operation.

OR	
Immediate Data	1 1 0 0
R2	1 0 1 0
Result	1 1 1 0

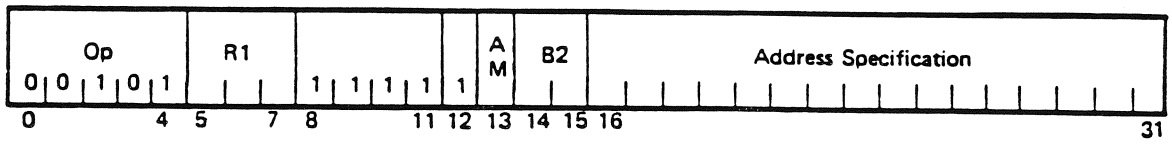
RESULTING CONDITION CODE:

00 The result is zero
11 The result is not zero

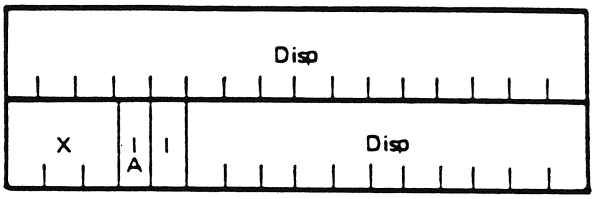
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

7.11 OR TO STORAGE



	AM	Mnemonic	Format
Extended:	0	OST	R1,D2(B2)
Indexed:	1	OST [@] [#]	R1,D2(X2,B2)



DESCRIPTION:

The logical sum (OR), of the fullword second operand and the contents of general register R1, is formed bit-by-bit. The result replaces the second operand. The contents of general register R1 are not changed. The following table defines the OR operation.

OR	
Storage	1 1 0 0
R1	1 0 1 0
Result	1 1 1 0

RESULTING CONDITION CODE:

00 The result is zero
11 The result is not zero

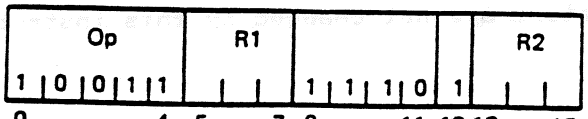
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

WARNING!

This instruction requires multiple memory accesses. The CPU does not prohibit IOP accesses of the selected main storage location during the time between the fetch of the operand and store of the result. Therefore, this instruction should not be used with any memory locations that might be DMA'd into.

7.12 SEARCH UNDER MASK



Mnemonic	Format
SUM	R1, R2

DESCRIPTION:

A variable search of an array under control of fields in a mask for specific bit patterns is performed. A twos complement 16-bit integer count is contained in bits 0 through 15 of the general register specified by R2. (This must be a positive number for correct execution of this instruction.)

The address of an array (Ai) is contained in bits 0 through 15 of the general register pair specified by R1 and (R1 +1)mod8. A twos complement integer modifier is contained in bits 16 through 31. After each Ai has been located via bits 0 through 15, the modifier is added to the most significant 16 bits of general register R1. This result replaces the most significant 16 bits. The modifier is not changed. A 16-bit mask (M) is contained in bits 0 through 15 of the general register specified by (R1+1)mod8 while field values (FV) are contained in bits 16 through 31.

The following equation is solved.

$$(Ai \wedge M) \oplus (FV \wedge M)$$

where

$i = 1, \ldots, count$

$\wedge$ = logical AND function

$\oplus$ = logical Exclusive-OR function.

$Ai \wedge M$ extracts bits selected by the mask out of array. $FV \wedge M$ extracts bits selected by the mask also. These latter bits are compared with $Ai \wedge M$. If they are equal, the comparison continues until the count is exhausted. The condition code reflects the result of this operation.

If the comparison indicates an inequality, the instruction is terminated with the address of the inequality operand located in general register R1, bits 0 through 15.

RESULTING CONDITION CODE:

00 All array items matched
11 An array item mismatched and general register R1 has the address where it failed

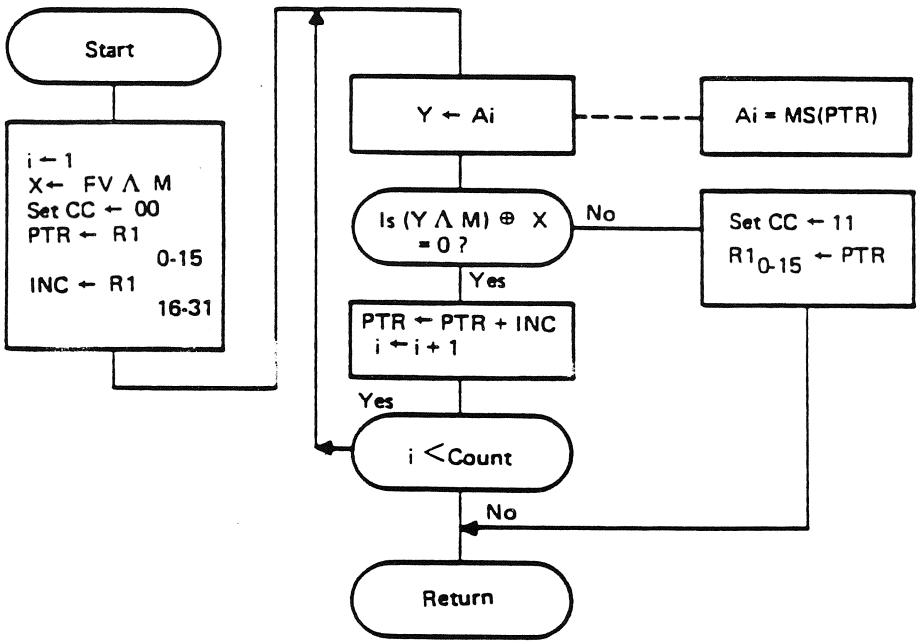
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

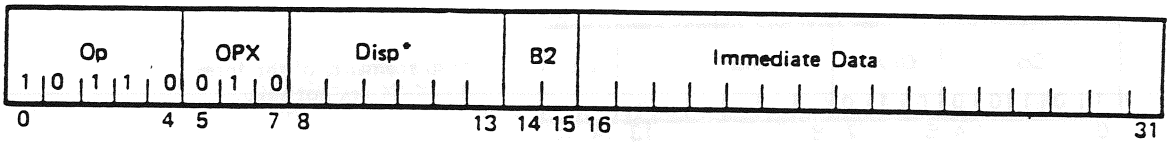
PROGRAMMING NOTES:

This is a variable length instruction execution. Care must be taken to ensure proper interrupt response by using sufficiently small count values. In order to assure proper completion of the putaway routine, the programmer must make sure that the count values do not exceed sixteen.

The following flowchart indicates how this instruction is executed:



7.13 SET BITS



Mnemonic	Format
SB	D2(B2),Data

* Displacements of the form 111XXX are invalid.

DESCRIPTION:

Bits 16 through 31 of this instruction are treated as halfword immediate data. The logical sum (OR), of the immediate data and the halfword main storage operand, is formed bit-by-bit. The result replaces the halfword main storage operand.

RESULTING CONDITION CODE:

00 The result is zero
11 The result is not zero

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

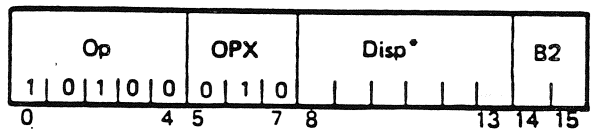
The one bits in the halfword mask specify the bits of the halfword second operand that are set one. The result replaces the halfword second operand. The following table defines this instruction.

SET BITS	
Mask	1 1 0 0
Storage	1 0 1 0
Result	1 1 1 0

WARNING!

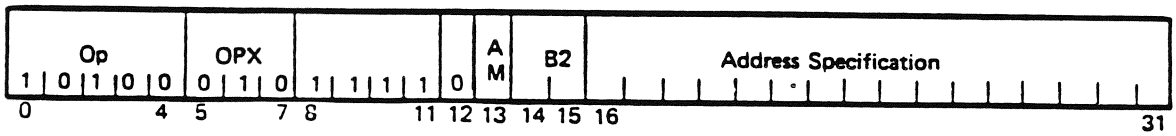
This instruction requires multiple memory accesses. The CPU does not prohibit IOP accesses of the selected main storage location during the time between the fetch of the operand and store of the result. Therefore, this instruction should not be used with any memory locations that might be DMA'd into.

7.14 SET HALFWORD

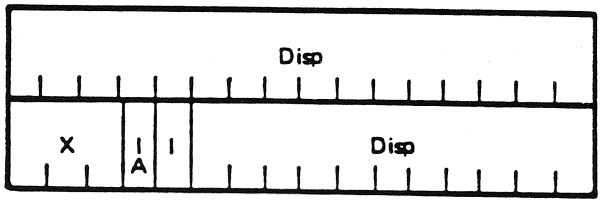


* Displacements of the form 111XXX are not valid.

Mnemonic: SHW
Format: D2(B2)



	AM	Mnemonic	Format
Extended:	0	SHW	D2(B2)
Indexed:	1	SHW[@][#]	D2(X2,B2)



DESCRIPTION:

The halfword main storage operand is set to all ones.

RESULTING CONDITION CODE:

The condition code is not changed by this instruction.

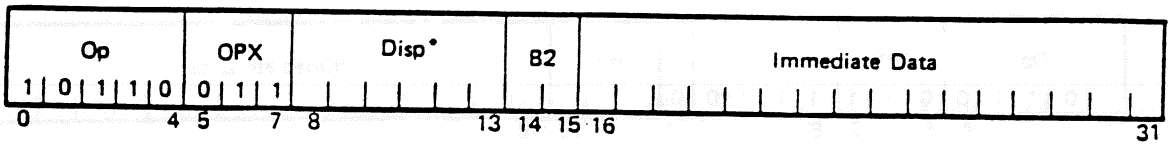
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

This instruction is similar to the SET BITS instruction with the mask (i.e., immediate data) equal to all ones.

7.15 TEST BITS



Mnemonic	Format
TB	D2(B2),Data

* Displacements of the form 111XXX are invalid.

DESCRIPTION:

Bits 16 through 31 of this instruction are treated as immediate data. This halfword immediate data is logically tested with the halfword main storage operand. A one in the immediate data tests the corresponding bit in the halfword main storage operand. The halfword main storage operand is not changed. The result of the test is given in the condition code.

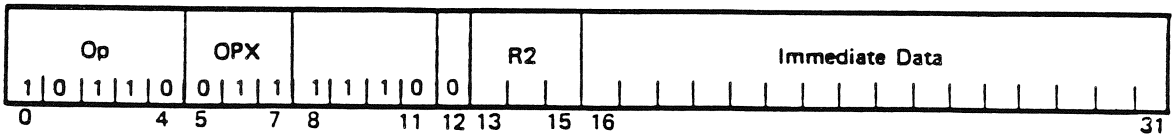
RESULTING CONDITION CODE:

00 Either the bits selected by the immediate data are zeros or the immediate data is all zeros
11 The bits selected by the immediate data are mixed with zeros and ones
01 The bits selected by the immediate data are all ones

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

7.16 TEST REGISTER BITS



Mnemonic	Format
TRB	R2,Data

DESCRIPTION:

Bits 16 through 31 of this instruction are treated as immediate data. A fullword operand is formed by appending 16 low-order zeros.

A one in this fullword tests the corresponding bit in general register R2. The corresponding bit position in general register R2 is not changed. The result of the test is given in the condition code.

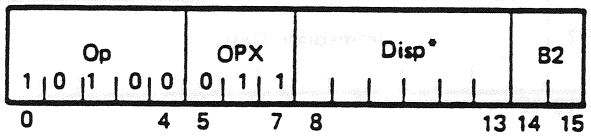
RESULTING CONDITION CODE:

00 Either the bits selected by the immediate data are all zeros or the immediate data is all zeros
11 The bits selected by the immediate data are mixed with zeros and ones
01 The bits selected by the immediate data are all ones

INDICATORS:

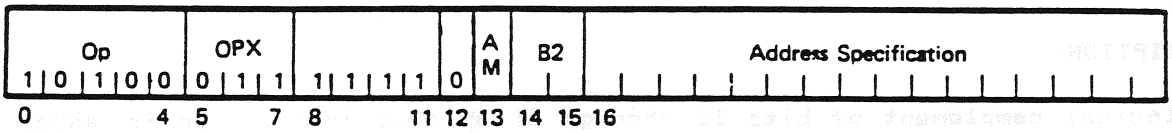
The overflow and carry indicators are not changed by this instruction.

7.17 TEST HALFWORD

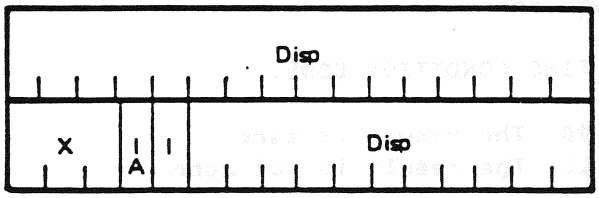


* Displacements of the form 111XXX are not valid.

Mnemonic	Format
TH	D2(B2)



	AM	Mnemonic	Format
Extended:	0	TH	D2(B2)
Indexed:	1	TH [@] [#]	D2(X2,B2)



DESCRIPTION:

All bits in the halfword main storage operand are tested. This operand is not changed. The result of the test is given in the condition code.

RESULTING CONDITION CODE:

00 The bits are all zeros
11 The bits are mixed with zeros and ones
01 The bits are all ones

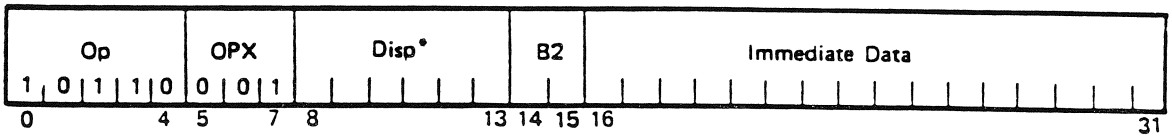
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

This instruction is the same as the TEST BITS instruction with the mask equal to all ones.

7.18 ZERO BITS



Mnemonic	Format
ZB	D2(B2),Data

* Displacements of the form 111XXX are invalid.

DESCRIPTION:

The logical complement of bits 16 through 31 of this instruction is ANDed to the halfword main storage operand bit-by-bit. The result replaces the halfword main storage operand.

RESULTING CONDITION CODE:

00 The result is zero
11 The result is not zero

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

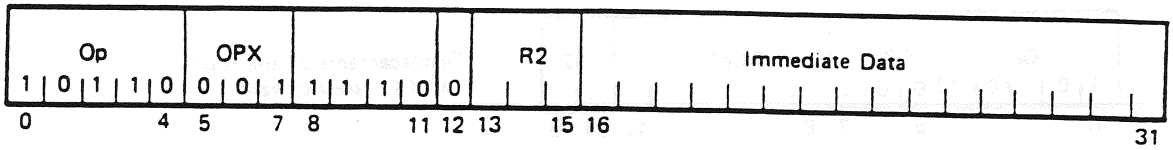
The one bits in the halfword immediate data specify the bits of the halfword main storage operand that are set zero. The result replaces the halfword main storage operand. The following table defines this instruction:

ZERO BITS	
Immediate Data	1 1 0 0
Storage	1 0 1 0
Result	0 0 1 0

WARNING!

This instruction requires multiple memory accesses. The CPU does not prohibit IOP accesses of the selected main storage location during the time between the fetch of the operand and store of the result. Therefore, this instruction should not be used with any memory locations that might be DMA'd into.

7.19 ZERO REGISTER BITS



Mnemonic	Format
ZRB	R2,Data

DESCRIPTION:

First, the halfword immediate data is expanded to a fullword by appending 16 low-order zeros. The logical complement of this fullword is then ANDed to the contents of general register R2. The result replaces general register R2.

RESULTING CONDITION CODE:

00 The result is zero
11 The result is not zero

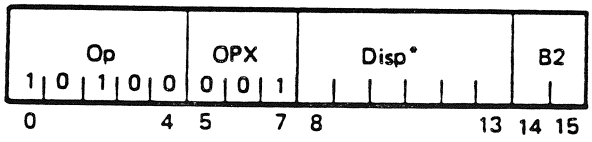
INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

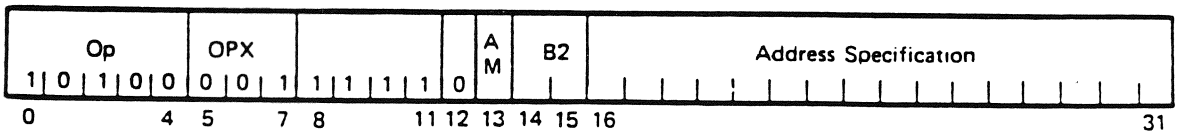
The one bits in the halfword immediate data specify the bits in the general register that are set zero. Bits 16 through 31 of general register R2 are not changed by this instruction.

7.20 ZERO HALFWORD

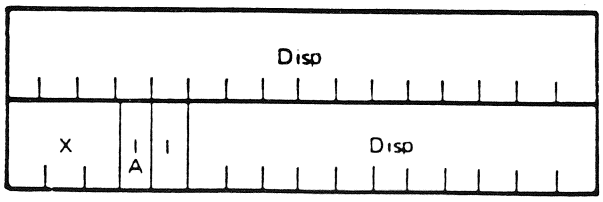


* Displacements of the form 111XXX are not valid.

Mnemonic	Format
ZH	D2(B2)



	AM	Mnemonic	Format
Extended:	0	ZH	D2 (B2)
Indexed:	1	ZH|@||=|	D2(X2,B2)



DESCRIPTION:

The halfword second operand is set to all zeros.

RESULTING CONDITION CODE:

The condition code is not changed by this instruction.

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

This instruction is similar to the ZERO BITS instruction with the mask equal to all ones.

8.0 FLOATING POINT OPERATIONS

The floating point instruction set is used to perform calculations on operands with a wide range of magnitude and to yield results scaled to preserve precision.

A floating point number consists of a signed exponent and a signed fraction. The quantity expressed by this number is the product of the fraction and the number 16 raised to the power of the exponent. The exponent is expressed in excess 64 binary notation; the fraction is expressed as a sign-magnitude hexadecimal number having a radix point to the left of the high-order fraction digit.

The floating point instruction set provides for loading, adding, subtracting, comparing, multiplying, dividing, and storing. Short operands generally provide faster processing and require less storage than long operands. On the other hand, long operands provide greater precision in computation. Operations may be either register-to-register or storage-to-register. All floating point instructions are part of the floating point feature including the two data conversion instructions. A normalized number is one in which the high-order hexadecimal digit of the fraction is not zero or else one in which both the fraction and characteristic are zero (true zero).

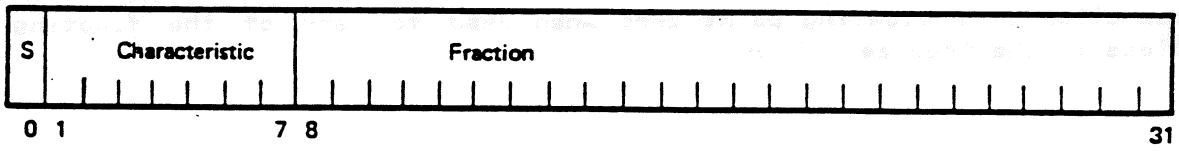
Maximum precision is preserved in addition, subtraction, multiplication, and division because all results are normalized.

The condition code is set as a result of all compare, add, subtract, and load operations.

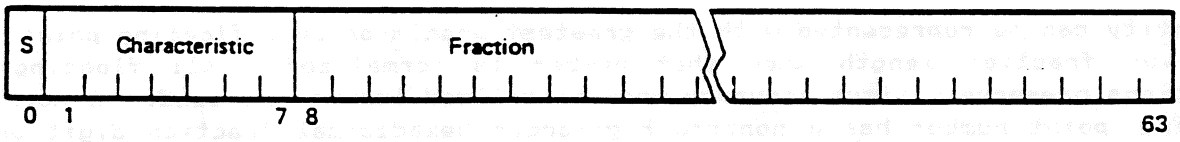
8.1 DATA FORMAT

Floating point data occupy a fixed-length format which may be either a fullword short format or a doubleword long format. Both formats may be used in main storage.

Short Floating-Point Number



Long Floating-Point Number



The first bit in either format is the sign bit(s). The subsequent seven bit positions are occupied by the characteristic. The fraction field may have either six or 14 hexadecimal digits.

Although final results have six fraction hexadecimal digits in short-precision, intermediate results may have additional low-order digits. These low-order digits, the guard digits, increase the precision of the final result.

8.2 NUMBER REPRESENTATION

The fraction of a floating point number is expressed in hexadecimal digits. The radix point of the fraction is assumed to be immediately to the left of the high-order fraction digit. To provide the proper magnitude for the floating point number, the fraction is considered to be multiplied by a power of 16. The characteristic portion, bits 1 through 7 of both floating point formats, indicates this power. The bits within the characteristic field can represent numbers from 0 through 127. To accommodate large and small magnitudes, the characteristic is formed by adding 64 to the actual exponent. The range of the exponent is thus -64 through +63. This technique produces a characteristic in excess 64 notation.

Both positive and negative quantities have a true fraction, the difference in sign being indicated by the sign bit. The number is positive or negative accordingly as the sign bit is zero or one.

The range covered by the magnitude (M) of a normalized floating point number is:

In short precision - $16^{-65} \leq M \leq (1-16^{-6}) \cdot 16^{63}$, and

In long precision - $16^{-65} \leq M \leq (1-16^{-14}) \cdot 16^{63}$,

Or approximately - $5.4 \cdot 10^{-79} \leq M \leq 7.2 \cdot 10^{75}$.

The short and long precisions contain 6.2 and 15.5 decimal digits, respectively.

A number with zero characteristic, zero fraction, and plus sign is called a true zero. A true zero may arise as the result of an arithmetic operation because of the particular magnitude of the operands. A true zero is forced when one or both operands of MULTIPLY or the dividend in DIVIDE has a zero fraction. The sign of a sum, difference, product, or quotient with zero fraction is positive. The proper representation of a floating point zero when used for any of the floating point operations is the true zero form.

8.3 NORMALIZATION

A quantity can be represented with the greatest precision by a floating point number of given fraction length when that number is normalized. All floating point operations preserve maximum accuracy when normalized inputs are used. A normalized floating point number has a nonzero high-order hexadecimal fraction digit or is a true zero (all digits zero). If one or more high-order fractional hexadecimal digits are zero, the number is said to be unnormalized unless it is a true zero. The process of normalization consists of shifting the fraction left until the high-order hexadecimal digit is nonzero and reducing the characteristic by the

number of hexadecimal digits shifted. A zero fraction cannot be normalized, and its associated characteristic therefore remains unchanged when normalization is called for. A floating point word of all zeros is defined as a true zero.

Normalization usually takes place when the intermediate arithmetic result is changed to the final result. This function is called postnormalization, and it is performed as part of instruction execution. Nonarithmetic instructions (i.e., Loads and Stores) do not normalize their outputs.

PROGRAMMING NOTES:

Floating point operands should be normalized prior to instruction execution; however, unnormalized inputs are not rejected via the unnormalized input interrupt as in earlier versions of this computer. Please note that although unnormalized inputs are accepted, programmers should expect a loss in accuracy for utilizing unnormalized numbers and their use is not recommended. Also note that for all arithmetic operations, any input with a zero fraction is treated as a true zero regardless of its sign or characteristic. A zero input to an arithmetic instruction will cause the bulk of the processing algorithm for the instruction to be bypassed, resulting in drastic decreases in execution time.

8.4 FLOATING POINT SECOND OPERANDS

Second operands for the floating point set are no longer restricted by hardware to even halfword boundary address locations.

8.5 FLOATING POINT REGISTERS

The registers used for floating point arithmetic are distinct or separate registers from those used for fixed point arithmetic. Register designation may be even or odd for short operands.

The first operand is contained in floating point register R1 when the second operand is a short 32-bit operand. If the second operand is a long or extended operand, the first operand is contained in the pair of floating point registers specified by R1 and (R1+1)mod8.

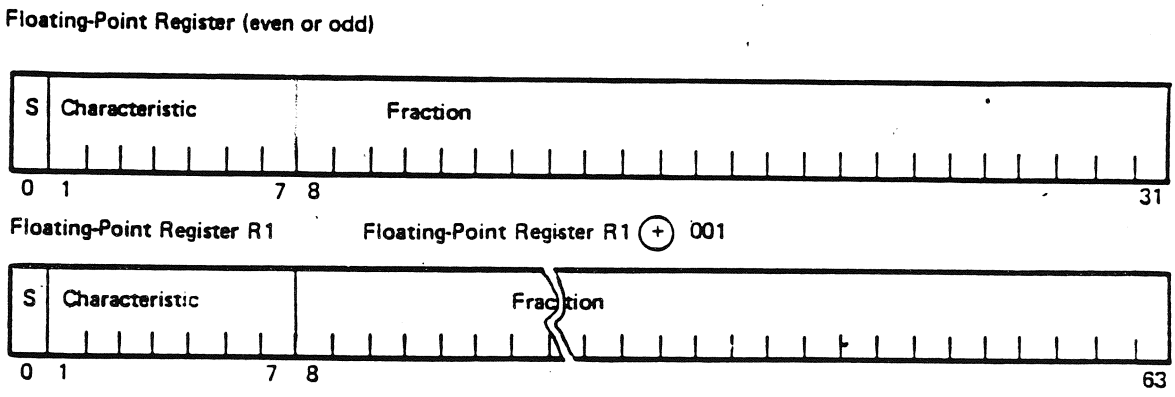


Figure 8-1. Floating Point Operands in Registers

A comprehensive set of floating point instruction is available for both short and long operands. Figure 8-2 summarizes the various combinations of fractional precision used for the floating point operands. For further detail, see the individual instructions.

Instructions	Short 2nd Operand			Long 2nd Operand		
		Operand			Operand	
	Result	1	2	Result	1	2
RRs						
A/S	24	← 24	± 24	56	← 56	± 56
C		24	: 24		56	: 56
M	24/48	← 24	x 24	56	← 56	x 56
D	24	← 24	÷ 24	56	← 56	÷ 56
Convert to Floating	24	←	32			
Convert to Fixed	32	←	24			
L	24	←	24			
SRSs						
A/S	24	← 24	± 24			
M	24/48	← 24	x 24			
D	24	← 24	÷ 24			
L	24	←	24			
ST	24	→	24			
RSs						
A/S	24	← 24	± 24	56	← 56	± 56
C		24	: 24		56	: 56
M	24/48	← 24	x 24	56	← 56	x 56
D	24	← 24	÷ 24	56	← 56	÷ 56
L	24	←	24	56	←	56
ST	24	→	24	56	→	56

Figure 8-2. Combinations of Fractional Precision for Floating Point Operands

8.6 FLOATING POINT INSTRUCTIONS

The floating point arithmetic instructions and their mnemonics, and descriptions follow. The following table indicates when the condition code is set and the exceptions in operand designations, data, or results that cause a program interruption.

Name	Mnemonic	Type	Exceptions
Add (Long Operands)	AEDR	RR C	U,E,S
Add (Long Operands)	AED	RS C	U,E,S
Add (Short Operands)	AER	RR C	U,E,S
Add (Short Operands)	AE	SRS,RS C	U,E,S
Compare (Long Operands)	CEDR	RR C	
Compare (Long Operands)	CED	RS C	
Compare (Short Operands)	CER	RR C	
Compare (Short Operands)	CE	RS C	
Convert to Fixed Point	CVFX	RR C	O
Convert to Floating Point	CVFL	RR C	
Divide (Extended Operands)	DEDR	RR	U,E,F
Divide (Extended Operands)	DED	RS	U,E,F
Divide (Short Operands)	DER	RR	U,E,F
Divide (Short Operands)	DE	SRS, RS	U,E,F
Load (Long Operands)	LED	RS C	XN
Load (Short Operands)	LE	SRS, RS C	XN
Load (Short Operands)	LER	RR C	XN
Load Complement (Short Operands)	LECR	RR C	XN*
Load Fixed Register	LFXR	RR	XN
Load Floating Immediate (Short Operands)	LFLI	RR	
Load Floating Register (Short Operands)	LFLR	RR	XN
Midvalue Select (Short Operands)	MVS	RS C	
Multiply (Extended Operands)	MEDR	RR	U,E
Multiply (Extended Operands)	MED	RS	U,E
Multiply (Short Operands)	MER	RR	U,E
Multiply (Short Operands)	ME	SRS, RS	U,E
Store (Long Operands)	STED	RS	XN
Store (Short Operands)	STE	SRS,RS	XN
Subtract (Long Operands)	SEDR	RR C	U,E,S
Subtract (Long Operands)	SED	RS C	U,E,S
Subtract (Short Operands)	SER	RR C	U,E,S
Subtract (Short Operands)	SE	SRS,RS,C	U,E,S

Notes:
- C — Condition code is set
- E — Exponent-overflow exception
- F — Floating point divide exception
- O — Overflow
- S — Significance exception
- U — Exponent-underflow exception
- XN — Output is not normalized
- XN* — Output is not normalized, but a true zero is written for an input with a zero fraction.

8.7 CONDITION CODE

The results of floating point add, compare, subtract, convert, load, and midvalue select operations are used to set the condition code. Multiplication, division, and stores leave the condition code unchanged. The condition code can be used for decision making by subsequent branch on condition instructions.

The condition code can be set to reflect the type of results for floating point instructions. The states 00, 11, or 01 indicate that the result is zero, less than zero, or greater than zero respectively. Load instructions which do not modify the input operand will set the condition code based upon the fraction of the operand only, thus it is possible to have a zero condition code set for a result which is not true zero. This interpretation is consistent since all floating point instructions interpret a fraction zero input as a true zero. Note that all arithmetic instructions always write a true zero when a fraction zero is encountered, so this condition can only occur for loads. State 10 is never set by floating point operations. The compare instruction indicates the relative arithmetic magnitude of the first operand (R1) and the second operand (called ϕ 2) (see Figure 8-3).

	00	11	01
Add S/L	zero	<zero	> zero
Compare S/L	(R1) = (ϕ2)	(R1) < (ϕ2)	(R1) > (ϕ2)
Load S/L	zero	< zero	> zero
Subtract S/L	zero	< zero	> zero
Converts	zero	< zero	> zero
Mid Value Select	within	above	below

Figure 8-3. Condition Code Setting for Floating Point Arithmetic

INDICATORS:

The overflow and carry indicators are not changed by floating point instructions.

8.8 FLOATING POINT ARITHMETIC EXCEPTIONS

Invalid operation codes, operand designations, data, or results cause a program interruption. When the interruption occurs, the current PSW is stored as an old PSW, and a new PSW is obtained. The interruption code in the old PSW identifies the cause of the interruption. The following exceptions cause a program interruption in floating point arithmetic.

Protection: Each halfword in main storage can be protected with a storage protection bit. The operation is terminated on a store violation.

Addressing: An address designates an operand location outside the available storage for the installed system. In most cases, the operation is terminated. The result data and the condition code, if affected, are unpredictable and should not be used for further computation.

Exponent Overflow: The result exponent in addition, subtraction, multiplication, or division exceeds 127 (16^{63}), and the result fraction is not zero. The operation is terminated without changing the operands, and a program interrupt occurs.

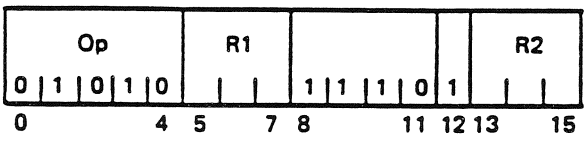
Exponent Underflow: The result exponent in addition, subtraction, multiplication, or division is less than zero (16^{-64}), and the result fraction is not zero. The operation is terminated, and a program interruption occurs if the exponent-underflow mask bit (PSW bit 22) is one.

The setting of the exponent-underflow mask also affects the result of the operation. When the mask bit is zero, the sign, exponent, and fraction are set to zero, thus making the result a true zero and no interrupt occurs. When the mask bit is one, the operation is terminated without changing the operands, and the interrupt is taken.

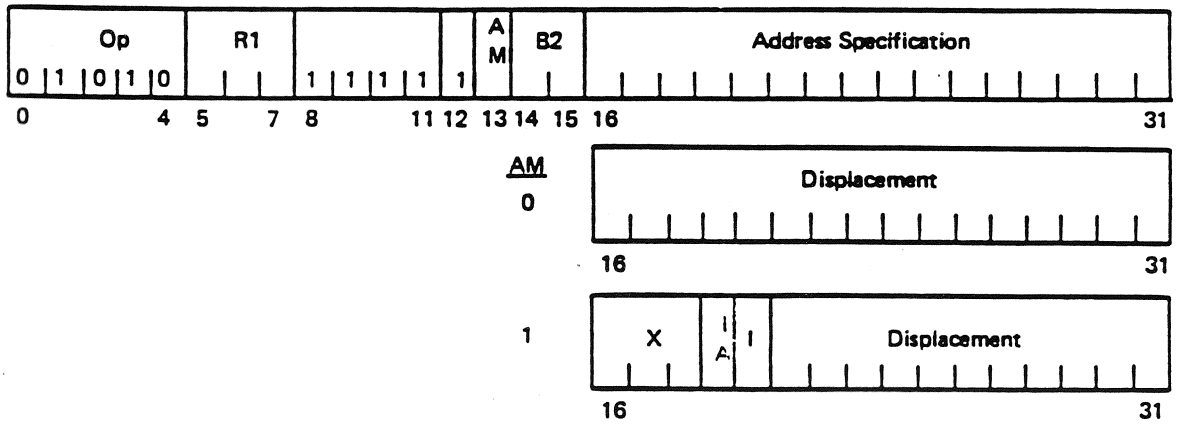
Significance: The result fraction of an addition or subtraction results in a zero fraction. A program interruption occurs if the significance mask bit (PSW bit 23) is one. The mask bit does not affect the result of the operation. A significance interrupt will result in a true zero answer with 00 condition code set.

Floating Point Divide: When division by an input with a zero fraction is attempted, the division is suppressed. The condition code and data in registers and storage remain unchanged.

8.9 ADD (LONG OPERANDS)



Mnemonic	Format
AEDR	R1, R2



	AM	Mnemonic	Format
Extended:	0	AED	R1, D2 (B2)
Indexed:	1	AED [@] [#]	R1, D2 (X2, B2)

DESCRIPTION:

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

The long 64-bit second operand is added with the contents of the floating point register pair specified by register R1. The normalized result is placed into floating point register pair specified by R1.

Addition of two floating point numbers consists of a characteristic comparison and a fraction addition. The characteristics of the two operands are compared, and the fraction with the smaller characteristic is right-shifted; its characteristic is increased by one for each hexadecimal digit of shift, until the two characteristics agree. The fractions are then added algebraically to form an intermediate sum. If a high-order carry occurs, the intermediate sum is right-shifted one hexadecimal digit, and the characteristic is increased by one. If this increase causes a characteristic overflow, an exponent-overflow exception is signaled, and a program interruption occurs.

The long intermediate sum consists of 15 hexadecimal digits, possible guard digits, and a possible carry.

After the addition, the intermediate sum is left-shifted as necessary to form a normalized fraction; vacated low-order digit positions are filled with zeros and the characteristic is reduced by the amount of shift.

If normalization causes the characteristic to underflow and the corresponding mask bit is one, a program interruption occurs and the operands remain unchanged (no result is written). If the mask bit is zero, a true zero is written as the result and no interrupt occurs.

When the intermediate sum is zero and the significance mask bit is one, a significance exception exists, and a program interruption takes place. Regardless of the sign of the significance bit, a true zero is written as the operations result. Exponent underflow does not occur for a zero fraction.

The sign of the sum is derived by the rules of algebra. The sign of a sum with zero result fraction is always positive.

RESULTING CONDITION CODE:

00 Result fraction is zero
11 Result is less than zero
01 Result is greater than zero.

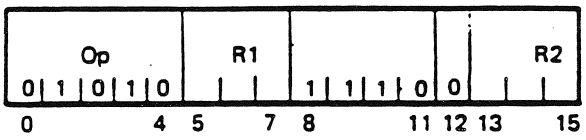
PROGRAM INTERRUPTS:

Significance
Exponent Overflow
Exponent Underflow

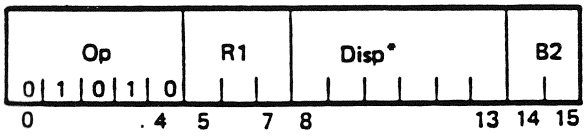
PROGRAMMING NOTES:

Interchanging the two operands in a floating point addition does not affect the value of the sum.

8.10 ADD (SHORT OPERANDS)

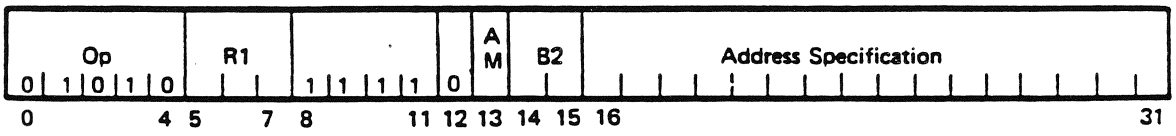


Mnemonic	Format
AER	R1,R2.

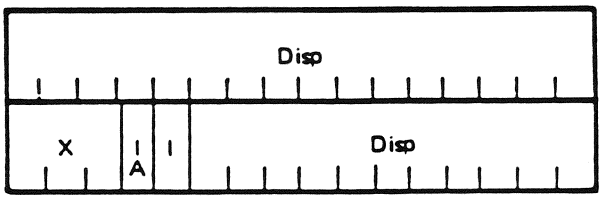


Mnemonic	Format
AE	R1,D2(B2)

* Displacements of the form 111XXX are not valid.



	AM	Mnemonic	Format
Extended:	0	AE	R1,D2(B2)
Indexed:	1	AE [@] [#]	R1,D2(X2,B2)



DESCRIPTION:

The short second operand is added to the short first operand, and the six digit normalized sum is placed in the first operand location.

Addition of two floating point numbers consists of a characteristic comparison and a fraction addition. The characteristics of the two operands are compared, and the fraction with the smaller characteristic is right-shifted; its characteristic is increased by one for each hexadecimal digit of shift, until the two characteristics agree. The fractions are then added algebraically to form an intermediate sum. If an overflow carry occurs, the intermediate sum is right-shifted one digit, and the characteristic is increased by one. If this increase causes a characteristic overflow, and exponent-overflow exception is signaled, and a program interruption occurs.

The short intermediate sum consists of seven hexadecimal digits and a possible carry. The low-order digits are guard digits retained from the fraction which is shifted right. The guard digits participate in the fraction addition. The guard digits are zero if no shift occurs.

After the addition, the intermediate sum is left-shifted as necessary to form a normalized fraction, vacated low-order digit positions are filled with zeros and the characteristic is reduced by the amount of shift.

If normalization causes the characteristic to underflow and the corresponding mask bit is one, a program interruption occurs and the operands remain unchanged (no result is written). If the mask bit is zero, a true zero is written as the result and no interrupt occurs.

When the intermediate sum is zero and the significance mask bit is one, a significance exception exists, and a program interruption takes place. Regardless of the setting of the significance bit, a true zero is written as the operation result. Exponent underflow does not occur for a zero fraction.

The sign of the sum is derived by the rules of algebra. The sign of a sum with zero result fraction is always positive.

RESULTING CONDITION CODE:

00	Result fraction is zero
11	Result is less than zero
01	Result is greater than zero

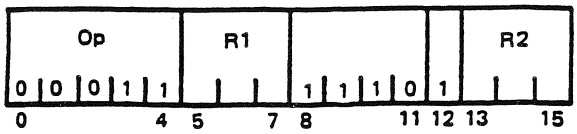
PROGRAM INTERRUPTS:

- Significance
- Exponent Overflow
- Exponent Underflow

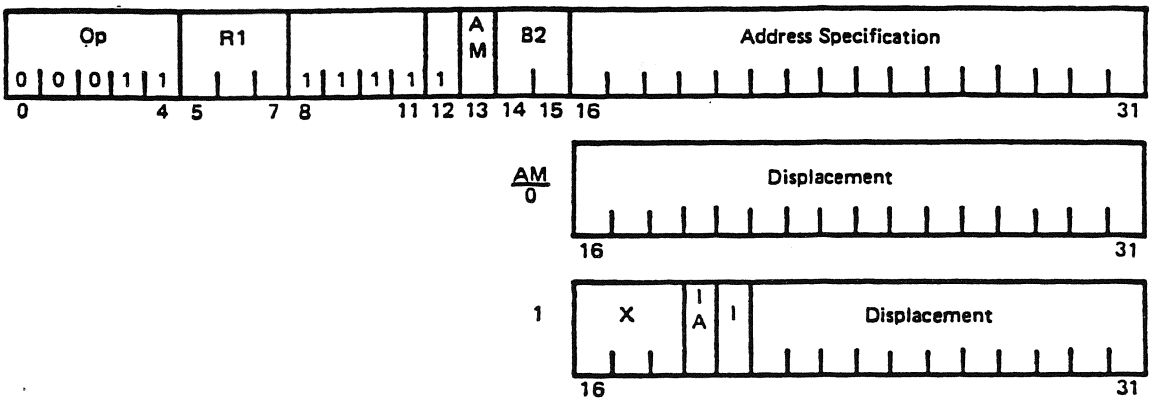
PROGRAMMING NOTES:

Interchanging the two operands in a floating point addition does not affect the value of the sum.

8.11 COMPARE (LONG OPERANDS)



Mnemonic	Format
CEDR	R1, R2



	AM	Mnemonic	Format
Extended:	0	CED	R1,D2(B2)
Indexed:	1	CED [@] [#]	R1,D2(X2,B2)

DESCRIPTION:

The long first operand is compared with the long second operand, and the condition code indicates the result.

The long second operand is compared with the contents of the floating point register pair specified by register R1. Comparison is algebraic, taking into account the sign, fraction, and exponent of each number. An equality is established by following the rules for normalized floating point subtraction. Neither operand is changed as a result of the operation.

Exponent overflow, exponent underflow, or loss significance cannot occur.

RESULTING CONDITION CODE:

- 00 Operands are equal
- 11 First operand is less than the second operand
- 01 First operand is greater than the second operand

PROGRAMMING NOTES:

Numbers with zero fraction compare equal even when they differ in sign or characteristic.

ANOMALY NOTE:

False indications of equality can occur in some cases when the fractional portion of the operands differ by x'80 0000' after prealignment.

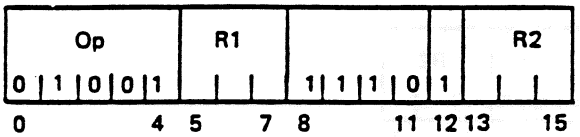
Prealignment shifts the fraction, of the operand with the smaller exponent, right a number of hex digits equal to the absolute value of the difference between the two exponents. The fraction being shifted is left filled with zeroes. After prealignment, the comparison is based on 64 fractional bits (right filled with zeroes) and a possible guard bit. Note that unnormalized numbers are not first normalized and are compared in the same manner as normalized numbers.

Examples of failing cases (return false indications of equality)

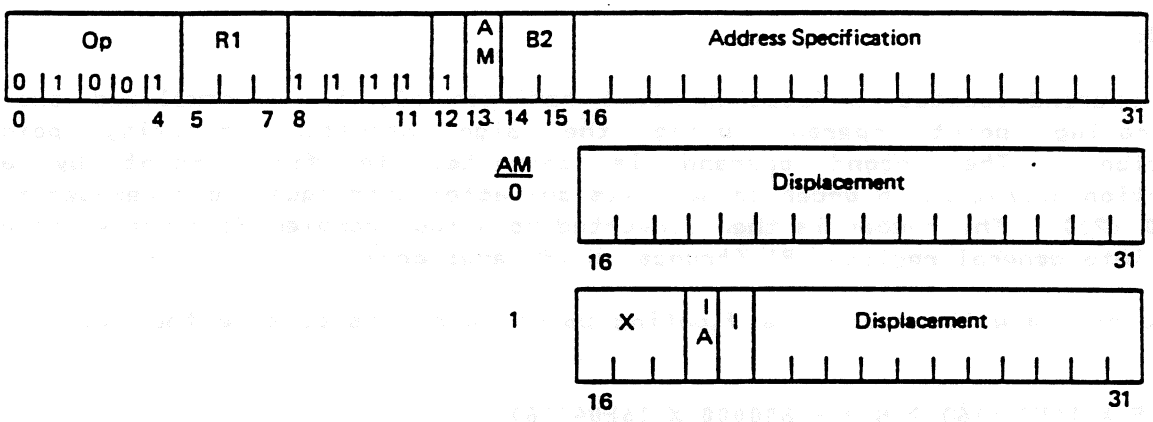
```
Operand 1:      423F FFFF 0000 1234
Operand 2:      423F FFFF 0080 1234
Absolute difference of OP2 and OP1 is .00 0000 0080 0000
Returns CC of 00 (equal); correct CC is 11 (OP1 < OP2)

Operand 1:      BEFF FFFF FB07 6890
Operand 2:      BF10 0000 0030 7689
Absolute difference of OP2 and OP1 is .00 0000 0080 0000
Returns CC of 00 (equal); correct CC is 01 (OP1 > OP2)

Operand 1:      4010 0000 0000 1234
Operand 2:      3FFF FFFF F801 2340
Absolute difference of OP2 and OP1 is .00 0000 0080 0000
Returns CC of 00 (equal); correct CC is 01 (OP1 > OP2)
```

(This page intentionally left blank)

8.12 COMPARE (SHORT OPERANDS)



Mnemonic	Format
CER	R1, R2



	AM	Mnemonic	Format
Extended:	0	CE	R1, D2 (B2)
Indexed:	1	CE [@] [#]	R1, D2 (X2, B2)

DESCRIPTION:

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

Comparison is algebraic, taking into account the sign, fraction, and exponent of each number. In short-precision, the low-order halves of the floating point registers are ignored. An equality is established by following the rules for normalized floating point subtraction. When the intermediate sum, including a possible guard digit, is zero, the operands are equal. Neither operand is changed as a result of the operation.

Exponent overflow, exponent underflow, or loss significance cannot occur.

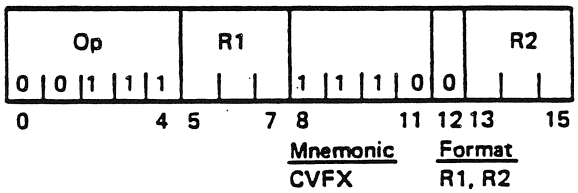
RESULTING CONDITION CODE:

00 Operands are equal
11 First operand is less than the second operand
01 First operand is greater than the second operand

PROGRAMMING NOTES:

Numbers with zero fraction compare equal even when they differ in sign or characteristic.

8.13 CONVERT TO FIXED POINT



DESCRIPTION:

The second operand located in floating point register R2, is a normalized short 32-bit floating point operand using the sign magnitude floating point representation. The second operand is converted to fixed point by an unnormalization operation in order to have its characteristic equal to a hexadecimal 44 (1000100 (2)). The number is then converted to a twos complement representation and placed into general register R1 (truncated if necessary).

A convert overflow will occur if a floating point number is outside the following range:

$$.7FFFFF \times 16E04(16) \geq N \geq -.800000 \times 16E04(16)$$

RESULTING CONDITION CODE:

00 Bits 0 through 15 of the result in general register R1 are zero
11 Bits 0 through 15 of the result in general register R1 are negative
01 Bits 0 through 15 of the result in general register R1 are positive

INDICATORS:

The overflow and carry indicators are not changed.

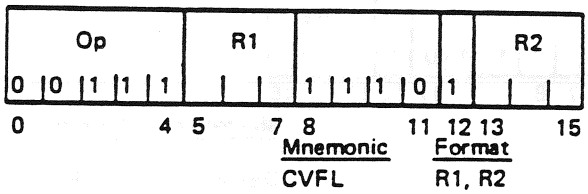
PROGRAM INTERRUPTS:

Convert overflow

PROGRAMMING NOTES:

Refer to the CONVERT TO FLOATING instruction.

8.14 CONVERT TO FLOATING POINT



DESCRIPTION:

The second operand is a 32-bit twos complement number with its binary point considered to be between bits 15 and 16. It is converted to sign magnitude floating point representation and placed into floating point register R1.

First, the sign bit of the fixed point number is placed into the sign bit of the intermediate result shown below. Then, bits 0 through 31 of the fixed point number are converted from twos complement representation to the magnitude of a sign-magnitude represention, and then placed into bits 8 through 39 of the intermediate result. The characteristic in bits 1 through 7 of the intermediate result is set to (1000100 (2)). Finally, the resulting intermediate number is normalized and only a short floating point representation (bits 0 through 31) is developed and placed into the floating point register R1.

RESULTING CONDITION CODE:

00 The floating point result is zero.
11 The floating point result is negative
01 The floating point result is positive (>0)

INDICATORS:

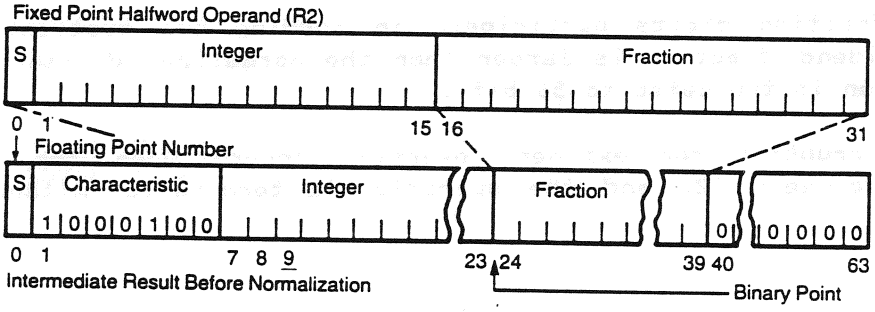
The overflow and carry indicators are not changed by this instruction.

PROGRAM INTERRUPTS:

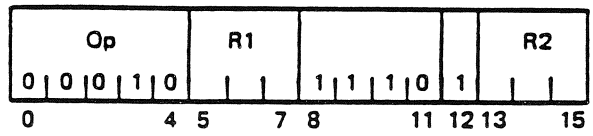
None

PROGRAMMING NOTES:

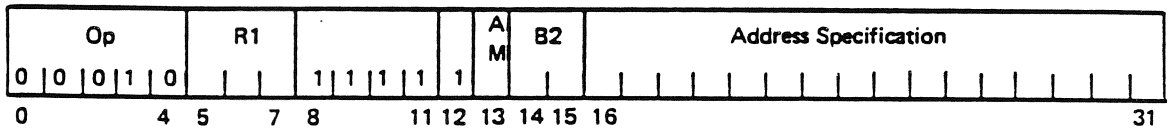
Unlike earlier versions of this machine, no significance interrupt is issued for conversion of fixed point zero to floating point true zero.

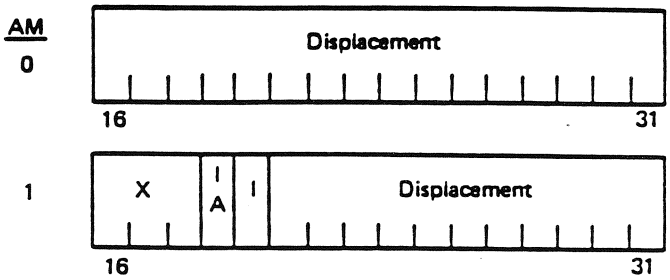


8.15 DIVIDE (LONG OPERANDS)



Mnemonic	Format
DEDR	R1, R2





	AM	Mnemonic	Format
Extended:	0	DED	R1, D2 (B2)
Indexed:	1	DED [@] [#]	R1, D2 (X2, B2)

DESCRIPTION:

The dividend (the long first operand) is divided by the divisor (the long second operand) and replaced by the quotient. No remainder is preserved.

The first operand is located in bits 0 through 63 of the pair of floating point registers specified by R1. The first operand is divided by the divisor, another long floating point operand, and the quotient replaces bits 0 through 63 of the pair of floating point registers specified by R1.

A floating point division consists of a characteristic subtraction and a fraction division. The difference between the dividend and divisor characteristics plus 64, is used as an intermediate quotient characteristic. The sign of the quotient is determined by the rules of algebra.

All dividend fraction digits participate in forming the quotient, even if the normalized dividend fraction is larger than the normalized divisor fraction. The quotient fraction is truncated to 56 bits.

A program interruption for exponent overflow occurs when the final quotient characteristic exceeds 127 and the operation is terminated, without changing the operands.

A program interruption for exponent underflow is possible if the final quotient characteristic is less than zero. If the corresponding mask bit is one a program interruption occurs and the operands remain unchanged (no result is written). If the mask bit is zero, a true zero is written as the result and no interrupt occurs. Underflow is not signaled for the intermediate quotient or for the operand characteristics during prenormalization.

When division by a zero divisor is attempted, the operation is suppressed. The dividend remains unchanged, and a program interruption for floating point divide exception occurs. When the dividend is a true zero, the quotient fraction will be zero. The quotient sign and characteristic are made zero, yielding a true zero result without taking the program interruptions for exponent underflow and exponent overflow. The program interruption for significance is never taken for division.

RESULTING CONDITION CODE:

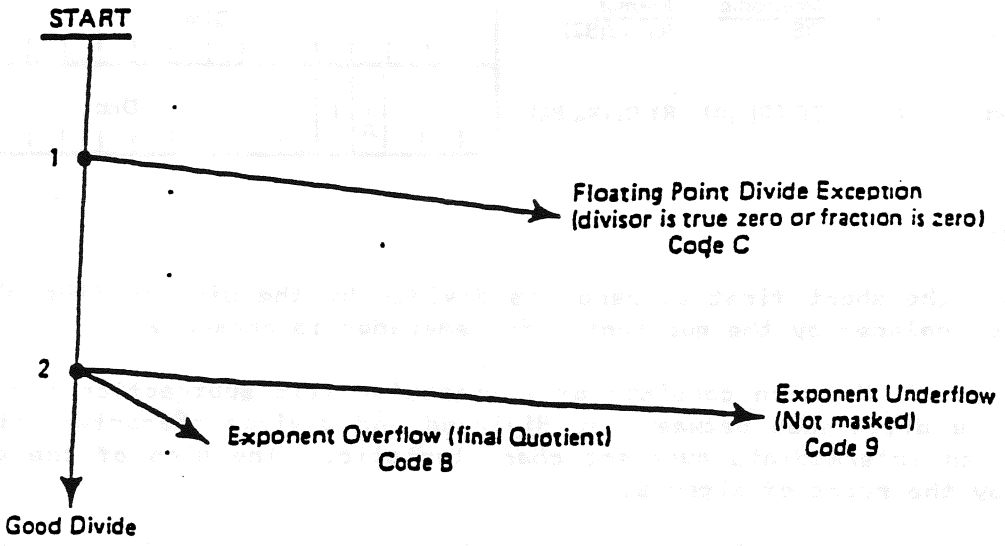
The code is not changed.

PROGRAM INTERRUPTS:

Exponent Overflow
Exponent Underflow
Floating Point Divide Exception

PROGRAMMING NOTES:

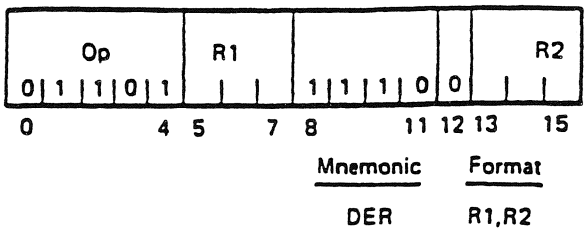
The divide instruction interrupt hierarchy for both long and short operands is given in the diagram below:



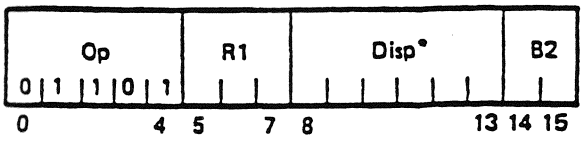
ANOMALY NOTE:

Under certain conditions, the accuracy of the quotient is limited to 29 fractional bits (counting 1 to 56). Since it is not feasible to characterize these conditions, the long divide instruction should not be used if more than 29 bits of precision are required.

8.16 DIVIDE (SHORT OPERANDS)

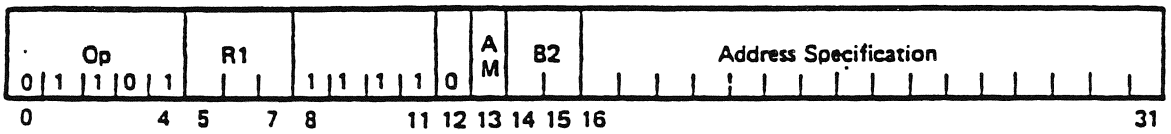


Mnemonic	Format
DER	R1,R2

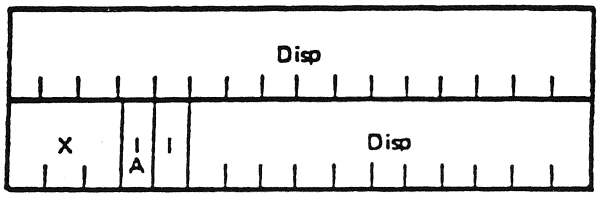


Mnemonic	Format
DE	R1,D2(B2)

* Displacements of the form 111XXX are not valid.



	AM	Mnemonic	Format
Extended:	0	DE	R1,D2(B2)
Indexed:	1	DE [@] [#]	R1,D2(X2,B2)



DESCRIPTION:

The dividend (the short first operand) is divided by the divisor (the short second operand) and replaced by the quotient. No remainder is preserved.

A floating point division consists of a characteristic subtraction and a fraction division. The difference between the dividend and divisor characteristics plus 64 is used as an intermediate quotient characteristic. The sign of the quotient is determined by the rules of algebra.

All dividend fraction digits participate in forming the quotient, even if the normalized dividend fraction is larger than the normalized divisor fraction. The quotient fraction is truncated to 24 bits.

A program interruption for exponent overflow occurs when the final quotient characteristic exceeds 127. The operation is terminated, without changing the operand.

A program interruption for exponent underflow is possible if the final quotient characteristic is less than zero. If the corresponding mask bit is one a program interruption occurs and the operands remain unchanged (no result written). If the

mask bit is zero, a true zero is written as the result and no interrupt occurs.

Underflow is not signaled for the intermediate quotient or for the operand characteristics during prenormalization.

When division by a zero divisor is attempted, the operation is suppressed. The dividend remains unchanged, and a program interruption for floating point divide exception occurs. When the dividend is a true zero, the quotient fraction will be zero. The quotient sign and characteristic are made zero, yielding a true zero result without taking the program interruptions for exponent underflow and exponent overflow. The program interruption for significance is never taken for division.

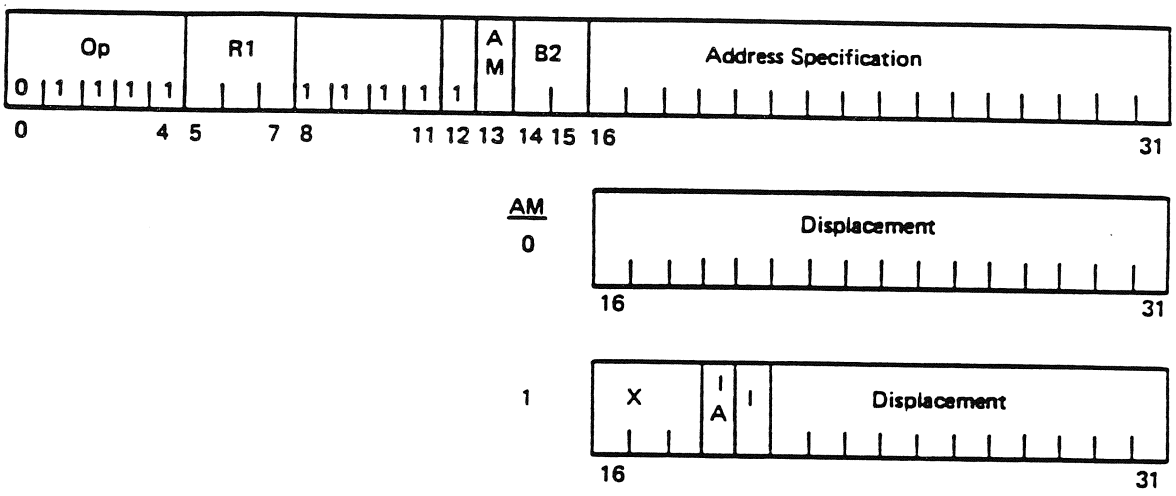
RESULTING CONDITION CODE:

The code is not changed.

PROGRAM INTERRUPTS:

Exponent Overflow
Exponent Underflow
Floating Point Divide Exception

8.17 LOAD (LONG OPERANDS)



	AM	Mnemonic	Format
Extended:	0	LED	R1, D2 (B2)
Indexed:	1	LED [@] [#]	R1, D2 (X2, B2)

DESCRIPTION:

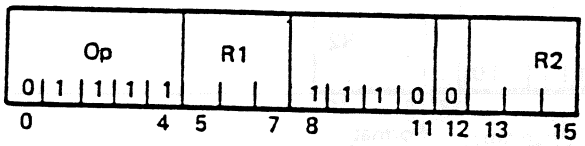
The long second operand is placed in the long first operand register. The second operand is not changed.

First, bits 0 through 31 of the doubleword main storage operand are loaded into floating point register R1. Then, bits 32 through 63 of the doubleword main storage operand are loaded into floating point register (R1+001)mod8. Exponent overflow, exponent underflow, or lost significance cannot occur.

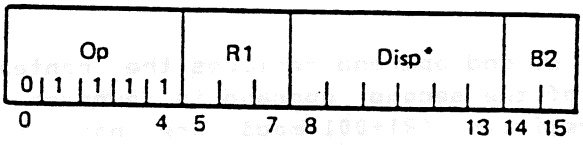
RESULTING CONDITION CODE:

00 The second operand has a zero fraction (not necessarily true zero operand)
11 The second operand is negative
01 The second operand is positive (>0)

8.18 LOAD (SHORT OPERANDS)

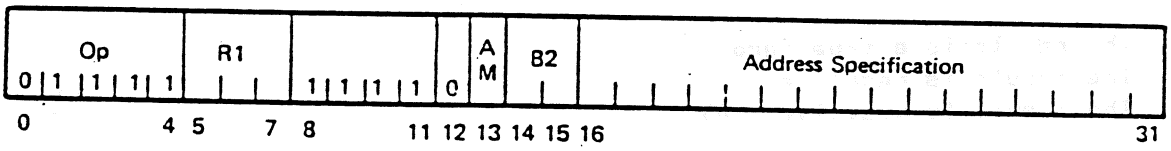


Mnemonic	Format
LER	R1,R2

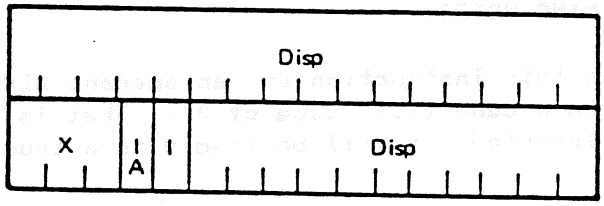


Mnemonic	Format
LE	R1;D2(B2)

* Displacements of the form 111XXX are not valid.



	AM	Mnemonic	Format
Extended:	0	LE	R1,D2(B2)
Indexed:	1	LE [@] [#]	R1,D2(X2,B2)



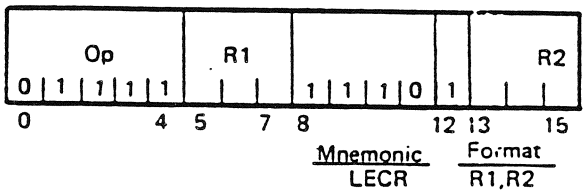
DESCRIPTION:

The long second operand is placed in floating point register R1. The second operand is not changed. The overflow, underflow, and carry indicators are not changed by this instruction.

RESULTING CONDITION CODE:

00 The second operand has a zero fraction (not necessarily true zero operand)
11 The second operand is negative
01 The second operand is positive (>0)

8.19 LOAD COMPLEMENT (SHORT OPERANDS)



DESCRIPTION:

The arithmetic complement of the fullword second operand replaces the contents of floating point register R1. The sign bit of the second operand is inverted, while the characteristic, the fraction, and register (R1+001)mod8 are not changed. Indicators are unchanged by this instruction.

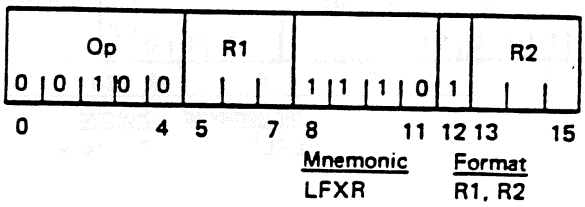
RESULTING CONDITION CODE:

00 The result is a true zero
11 The result is negative
01 The result is positive (>0)

PROGRAMMING NOTES:

Invoking this instruction on an operand with a fraction zero will result in a true zero with a condition code of 00. That is, an operand with zero fraction will not be complemented but will be loaded as a true zero regardless of characteristic.

8.20 LOAD FIXED REGISTER



DESCRIPTION:

The fullword contents of the floating point register specified by R2 are loaded into the general register specified by R1.

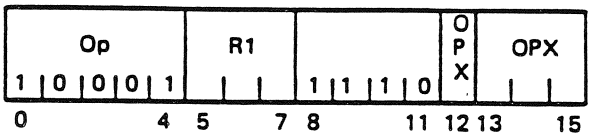
RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

8.21 LOAD FLOATING IMMEDIATE



Monemonic	Format
LFLI	R1, Value

DESCRIPTION:

A floating point immediate value is loaded into the floating point register specified by R1.

The immediate values are 0.,1.,2.,3.,4.,5.,6.,7.,8.,9.,10.,11., 12.,13.,14., and 15.

OPX (bits 12,13,14,15) (hex)	Immediate Values --> R1 (hex)
0	0000 0000 (TRUE ZERO)
1	4110 0000
2	4120 0000
3	4130 0000
4	4140 0000
5	4150 0000
6	4160 0000
7	4170 0000
8	4180 0000
9	4190 0000
A	41A0 0000
B	41B0 0000
C	41C0 0000
D	41D0 0000
E	41E0 0000
F	41F0 0000

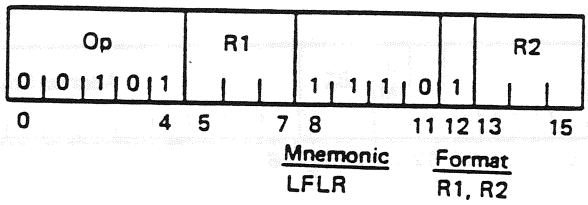
RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

8.22 LOAD FLOATING REGISTER



DESCRIPTION:

The fullword contents of the general register specified by R2 are loaded into the floating point register specified by R1.

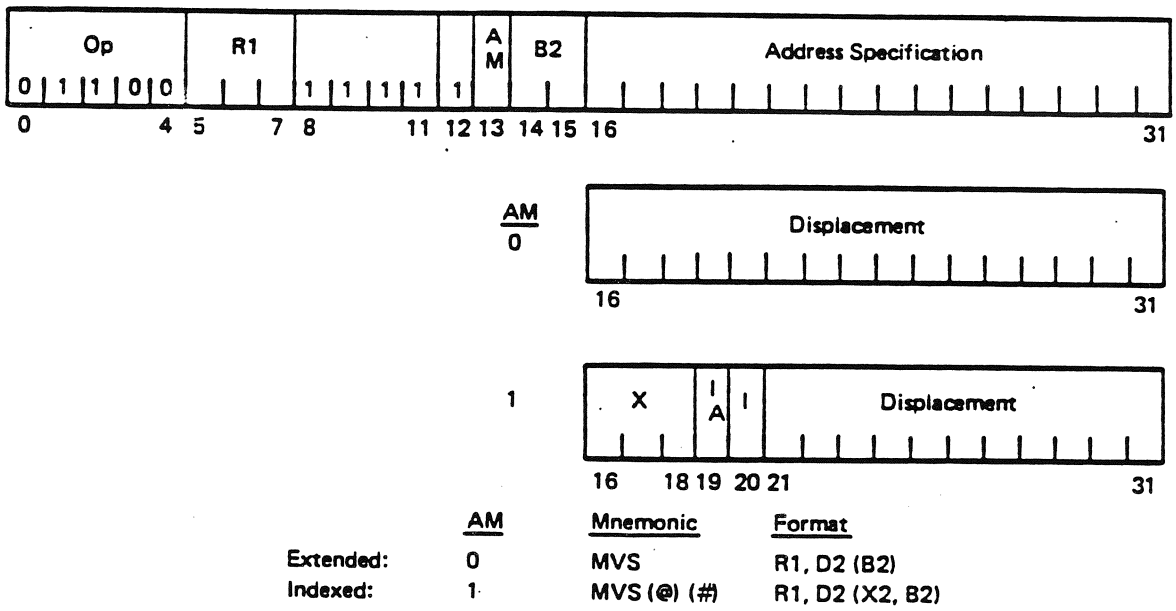
RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

8.23 MIDVALUE SELECT (SHORT OPERANDS)



	AM	Mnemonic	Format
Extended:	0	MVS	R1, D2 (B2)
Indexed:	1	MVS (@) (#)	R1, D2 (X2, B2)

DESCRIPTION:

The floating point registers specified by R1 and (R1+001)mod8 each contain a short (8/24) floating point operand. The third short floating point operand is located in the main storage effective address. The three operands are compared, and the midvalue operand is selected such that it is less than or equal to the maximum value operand. The normalized midvalue operand is then placed in the floating point register specified by R1. Both the main storage operand and the contents of Register (R1+001)mod8 are not changed.

RESULTING CONDITION CODE:

The condition code is set as a result of executing this instruction, but its value is, in general, meaningless when this instruction is used for midvalue selection. However, see the Programming Note for condition code settings when this instruction is used as a limiter.

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

This instruction can also be used as a limiter. The upper limit must be placed in (R1+1)mod8; the lower limit must be placed in the main store location. The input value to be tested must be placed in R1. The condition code will reflect the result of the instruction and, if the input value is outside the limit values, the appropriate limit value will be placed in R1.

When this instruction is used as a limiter, the condition code will be set as follows:

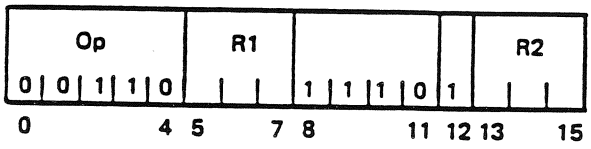
00	Within Limits:	Lower Limit (Main Storage Operand) ≤ Operand (Initial Contents of Register R1) ≤ Upper Limit (Contents of Register (R1+1)mod8. (R1 is midvalue)
01	Above Upper Limit:	Initial R1 Operand > Upper Limit (R1+1)mod8. (R1 is midvalue: originated in (R1+1)mod8)
11	Below Lower Limit:	Initial R1 Operand < Lower Limit (Main Storage Operand): (main storage operand is midvalue)

The programmer is responsible to ensure that the upper limit is not equal to the lower limit. If these conditions are inadvertently set up, the resulting condition code will be meaningless.

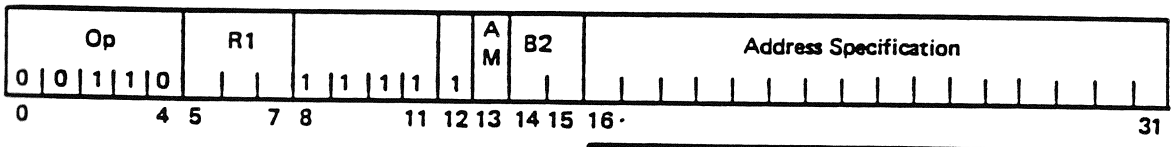
PROGRAM INTERRUPTS:

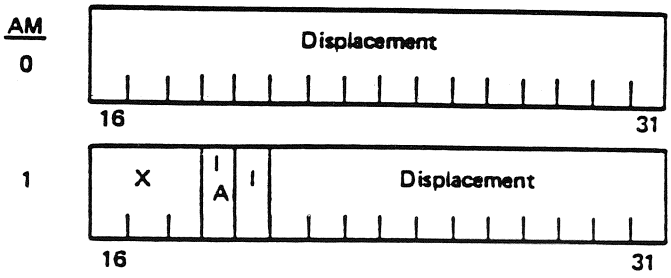
Underflow - the output of the MVS instruction, if normalized, could cause an exponent underflow.

8.24 MULTIPLY (LONG OPERANDS)



Mnemonic	Format
MEDR	R1, R2





	AM	Mnemonic	Format
Extended:	0	MED	R1, D2 (B2)
Indexed:	1	MED [@] [#]	R1, D2 (X2, B2)

DESCRIPTION:

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

The first operand is located in bits 0 through 63 of the pair of floating point registers specified by register R1. This operand is multiplied by the second operand. For the RR format, the second operand is located in bits 0 through 63 of the pair of floating point registers specified by R2. For the RS format, the second operand is located in bits 0 through 63 of the main storage word pair. The extended product replaces bits 0 through 63 of the pair of floating point registers specified by R1 and (R1+001)mod8.

The multiplication of two floating point numbers consists of a characteristic addition and a fraction multiplication. Fraction multiplication is accomplished by multiplying the three most significant fullword partial sum pairs and adding the results (to 68 bits), followed by normalization and truncation to 56 bits. The sum of the characteristic less 64 is used as the characteristic of an intermediate product.

The sign of the product is determined by the rules of algebra.

The product fraction is normalized by postnormalizing the intermediate product, if necessary, then truncating the product to 56 bits. The intermediate product characteristic is reduced by the number of left shifts.

Exponent overflow is possible if the final product characteristic exceeds 127. The operation is terminated, and a program interruption occurs without changing the operands. The overflow exception does not occur for an intermediate product characteristic exceeding 127, when the final characteristic is brought within range because of normalization.

Exponent underflow is possible if the final product characteristic is less than zero. If the floating point exponent underflow mask is a one, a program interruption occurs and operands remain unchanged (no result written). If the mask bit is zero, the result is made a true zero and no interrupt occurs.

When all digits of the intermediate product fraction are zero, the product sign and characteristic are made zero, yielding a true zero result. No interruption for exponent underflow or exponent overflow can occur when the result fraction is zero. The program interruption for lost significance is never taken for multiplication.

RESULTING CONDITION CODE:

The code is not changed.

PROGRAM INTERRUPTS:

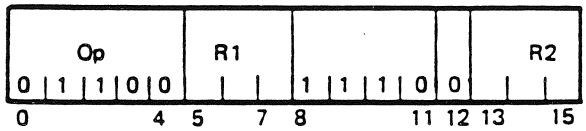
Exponent Overflow
Exponent Underflow

PROGRAMMING NOTES:

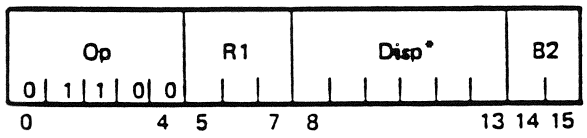
When either the multiplicand or multiplier is a true zero, the result is normally forced to a true zero without requiring the hardware to enter the longer multiply algorithm.

Interchanging the two operands will not affect the value of the product.

8.25 MULTIPLY (SHORT OPERANDS)

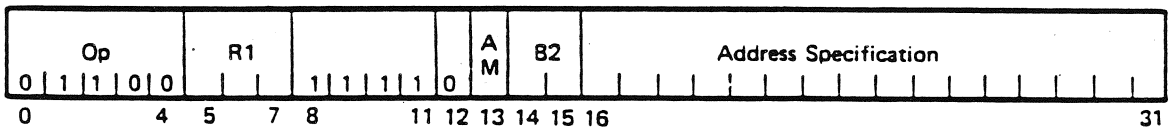


Mnemonic	Format
MER	R1,R2

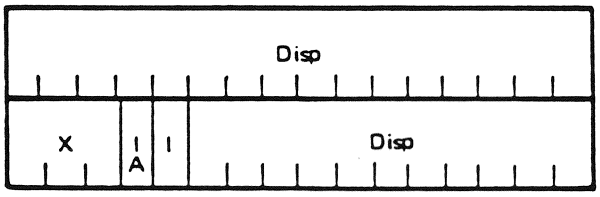


* Displacements of the form 111XXX are not valid.

Mnemonic	Format
ME	R1,D2(B2)



	AM	Mnemonic	Format
Extended:	0	ME	R1,D2,(B2)
Indexed:	1	ME [@] [#]	R1,D2(X2,B2)



DESCRIPTION:

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

The multiplication of two floating point numbers consists of a characteristic addition and a fraction multiplication. The sum of the characteristics less 64 is used as the characteristic of an intermediate product. The sign of the product is determined by the rules of algebra.

The product fraction is normalized by postnormalizing the intermediate product, if necessary. The intermediate product characteristic is reduced by the number of left shifts. For short operands (six-digit fractions), the product fraction has the full 14 digits of the long format with the two low-order fraction digits accordingly always zero.

Exponent overflow occurs if the final product characteristic exceeds 127. The operation is terminated, and a program interruption occurs without changing the operands. The overflow exception does not occur for an intermediate product characteristic exceeding 127, when the final characteristic is brought within range because of normalization.

Exponent underflow is possible if the final product characteristic is less than zero. If the floating point exponent underflow mask is a one, a program interrupt occurs and the operands are unchanged (no result written). If the mask bit is zero, the result is made a true zero and no interruption occurs.

When all 14 digits of the intermediate product fraction are zero, the product sign and characteristic are made zero, yielding a true zero result. No interruption for exponent underflow or exponent overflow can occur when the result fraction is zero. The program interruption for lost significance is never taken for multiplication.

If R1 is even, the least significant part of the product fraction replaces the contents of floating point register R1+001. The most significant part of the intermediate product fraction replaces the contents of floating point register R1.

RESULTING CONDITION CODE:

The code is not changed.

PROGRAM INTERRUPTS:

Exponent Overflow
Exponent Underflow

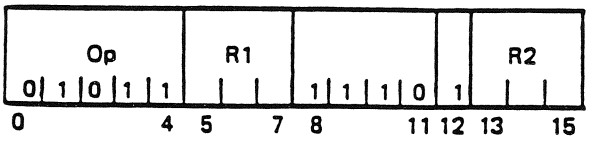
PROGRAMMING NOTES:

Interchanging the two operands in a floating point multiplication does not affect the value of the product.

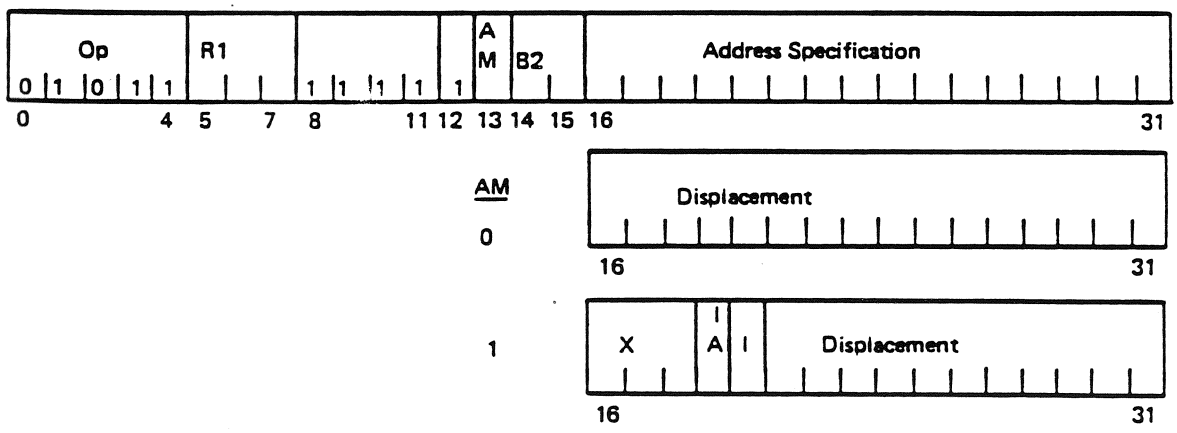
When either the multiplicand or multiplier is a true zero, the result is normally forced to a true zero without requiring the hardware to enter the longer multiply algorithm.

Notice that the MULTIPLY (short) instruction uses two registers for its result if R1 was even. This allows the programmer to use the additional precision without going to the extended form of the MULTIPLY. If R1 was odd, one register is used for the result (32 bit product).

8.26 SUBTRACT (LONG OPERANDS)



Mnemonic	Format
SEDR	R1, R2



	AM	Mnemonic	Format
Extended:	0	SED	R1, D2 (B2)
Indexed:	1	SED [@] [#]	R1, D2 (X2, B2)

DESCRIPTION:

The long second operand is subtracted from the long first operand, and the normalized difference is placed in the first operand location.

The long 64-bit second operand is subtracted from the contents of floating point register pairs specified by the register R1 and (R1+1)mod8. The normalized result is placed into floating point registers R1 and (R1+1)mod8.

The SUBTRACT (long operand) is similar to ADD (long operand), except that the sign of the second operand is inverted before addition.

The sign of the difference is derived by the rules of algebra. The sign of a difference with zero result fraction is always positive.

RESULTING CONDITION CODE:

00 Result is true zero
11 Result is less than zero
01 Result is greater than zero

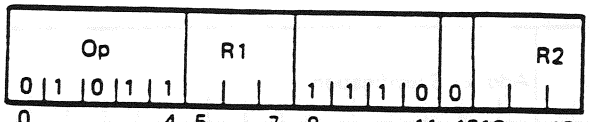
PROGRAM INTERRUPTS:

Significance
Exponent Overflow
Exponent Underflow

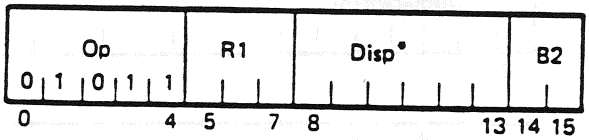
PROGRAMMING NOTES:

The technique used to clear a register by subtracting a floating point register from itself will work even though unnormalized numbers are used in the subtract operation. The result will be a true zero.

8.27 SUBTRACT (SHORT OPERANDS)

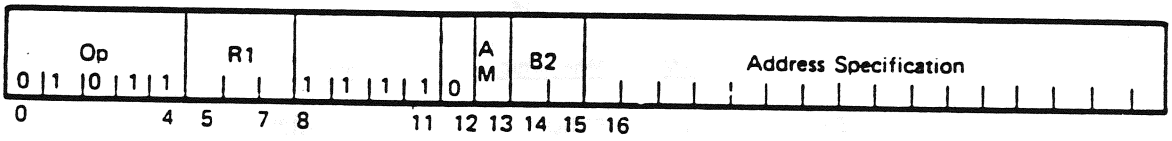


Mnemonic	Format
SER	R1,R2

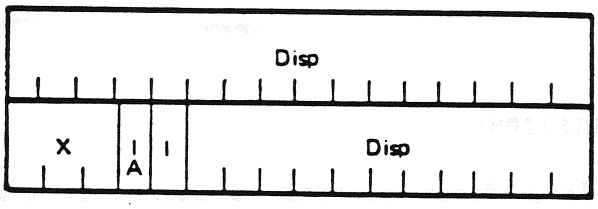


Mnemonic	Format
SE	R1,D2(B2)

* Displacements of the form 111XXX are not valid.



	AM	Mnemonic	Format
Extended:	0	SE	R1,D2(B2)
Indexed:	1	SE [@] [=]	R1,D2(X2,B2)



DESCRIPTION:

The short second operand is subtracted from the short first operand, and the normalized difference is placed in the first operand location.

The SUBTRACT (short operands) is similar to ADD (short operands), except that the sign of the second operand is inverted before addition.

The sign of the difference is derived by the rules of algebra. The sign of a difference with zero result fraction is always positive.

RESULTING CONDITION CODE:

00 Result is true zero
11 Result is less than zero
01 Result is greater than zero

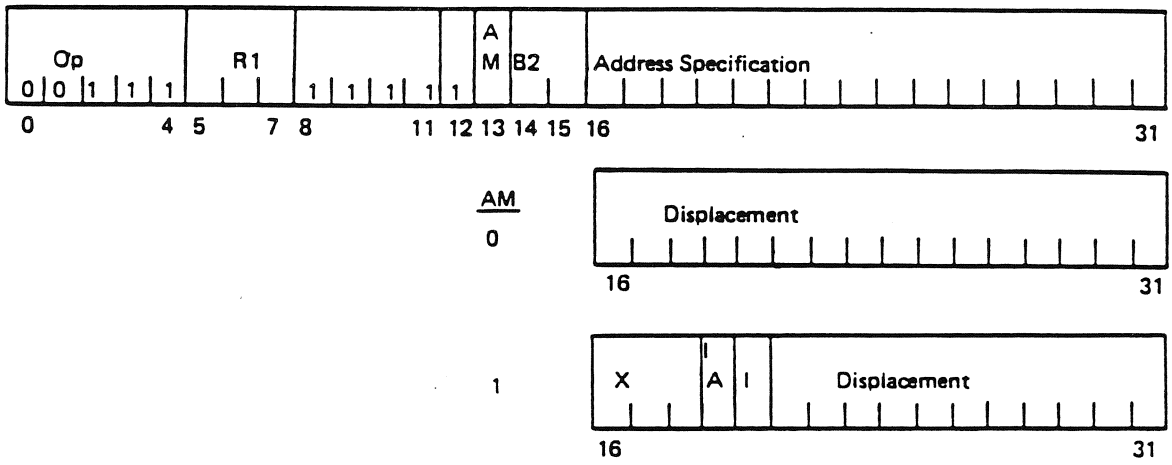
PROGRAM INTERRUPTS:

Significance
Exponent Overflow
Exponent Underflow

PROGRAMMING NOTES:

The technique used to clear a register by subtracting a floating point register from itself will work even though unnormalized numbers are used in the subtract operation. The result will be a true zero.

8.28 STORE (LONG OPERANDS)



	AM	Mnemonic	Format
Extended:	0	STED	R1, D2 (B2)
Indexed:	1	STED [@] [#]	R1, D2 (X2, B2)

DESCRIPTION:

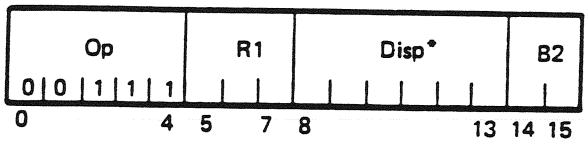
The long first operand is stored at the long second operand location. The first operand is not changed.

The first operand is located in the pair of floating point registers specified by register R1. First, bits 0 through 31 of floating point register R1 are stored in the fullword specified by the second operand fullword address. Bits 0 through 31 of floating point register (R1+1)mod8 are stored into the second fullword of the doubleword storage area starting with the second operand fullword address. The contents of register R1 and (R1 + 1)mod 8 are not changed.

RESULTING CONDITION CODE:

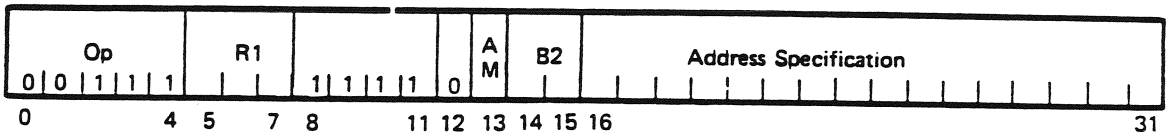
The code is not changed.

8.29 STORE (SHORT OPERANDS)

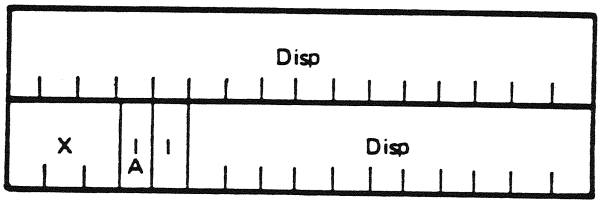


* Displacements of the form 111XXX are not valid.

Mnemonic	Format
STE	R1,D2(B2)



	AM	Mnemonic	Format
Extended:	0	STE	R1,D2(B2)
Indexed:	1	STE[@] [=]	R1,D2(X2,B2)



DESCRIPTION:

The contents of floating point register R1 are stored at the second operand location. The contents of R1 are not changed. The overflow and carry indicators are not changed by this instruction.

RESULTING CONDITION CODE:

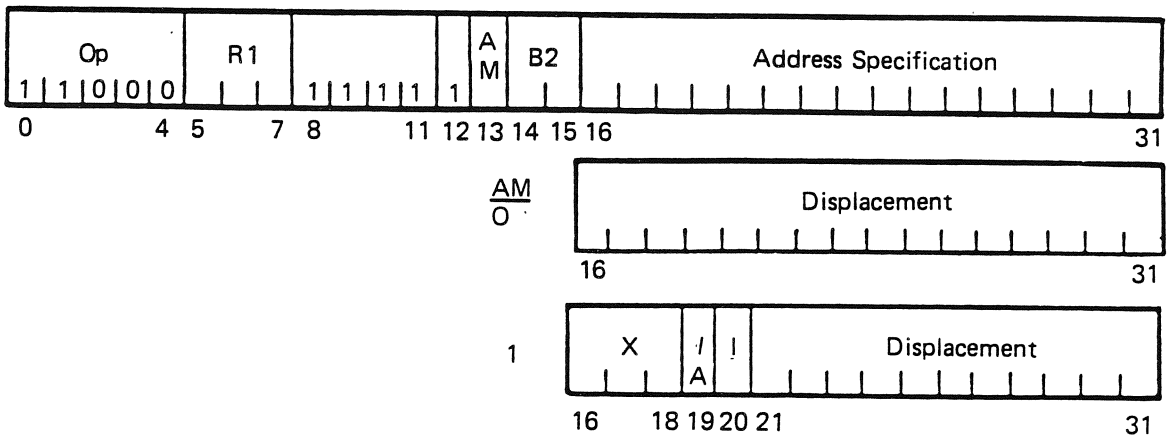
The code is not changed.

(This page intentionally left blank)

9.0 SPECIAL OPERATIONS

This section describes the special instructions. These instructions make possible the use of efficient pseudo subroutines, permit the specification of storage protection, perform status switching, control I/O, and loading and storing the Data Sector Register (DSE).

9.1 DIAGNOSE (DETECT)



	AM	Mnemonic	Format
Extended:	0	DIAG	R1,D2(B2)
Indexed:	1	DIAG(@)(#)	R1,D2(X2,B2)

DESCRIPTION:

A 16-bit effective address is developed in the normal manner without expanding to 19 bits. The effective address uniquely selects one of several special microprogram routines. These routines are used to perform built-in diagnostic functions to verify the proper functioning of the CPU hardware and to detect faulty components. The particular diagnostic operations performed are defined in Section 15.

The instruction is not intended for normal program usage. This is a privileged operation and can only be executed when the CPU is in the Supervisor state.

RESULTING CONDITION CODE:

00 The diagnostic result is "pass"
11 The diagnostic result is "fail"
01 --- (impossible)

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

AUTOMATIC INDEX ALIGNMENT:

This instruction aligns the index value assuming a halfword main storage operand.

PROGRAM CHECK EXCEPTIONS:

Privileged Instruction

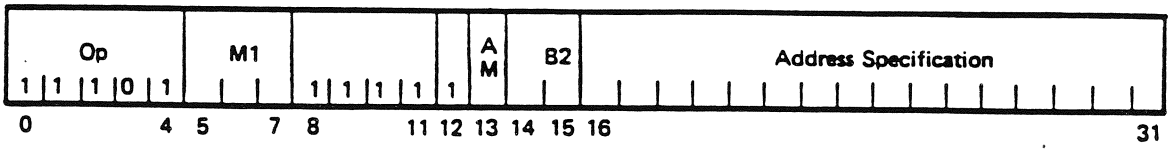
Address Specification - Address Violation for a fullword indirect address pointer.

Address Specification - Nonexistent Address for an indirect address pointer.

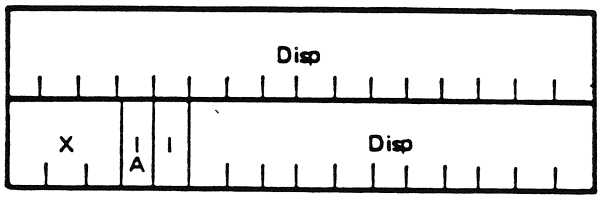
PROGRAMMING NOTES:

This instruction is not intended for general programming use; it is designed for the diagnostic programmer. Every programmer desiring to use the Diagnose instruction should be thoroughly familiar with the contents of the Diagnostic Function Appendix. Unexpected results can occur if this instruction is improperly used.

9.2 INSERT STORAGE PROTECT BITS



	AM	Mnemonic	Format
Extended:	0	ISPB	M1,D2(B2)
Indexed:	1	ISPB [@] [#]	M1,D2(X2,B2)



DESCRIPTION:

Bits 5 through 7, the M1 field, are decoded to set or reset the protection bit associated with each halfword in main-storage as specified by the EA. The contents of the specified location, however, are not changed.

The following defines the combinations of the M1 field and the corresponding result:

M1 Field	Result
000	Reset the storage protection bits for the halfword second operand.
001	Reset the storage protection bits for both halfwords in the fullword second operand.
010	Set the storage protection bits for the halfword second operand.
011	Set the storage protection bits for both halfwords in the fullword second operand.
100	Illegal
101	Illegal
110	Illegal
111	Illegal

This is a privileged operation and can only be executed when the CPU is in the Supervisor state.

RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

The carry and overflow indicators are not changed by this instruction.

PROGRAM INTERRUPTS:

Illegal operation
Privileged instruction

PROGRAMMING NOTES:

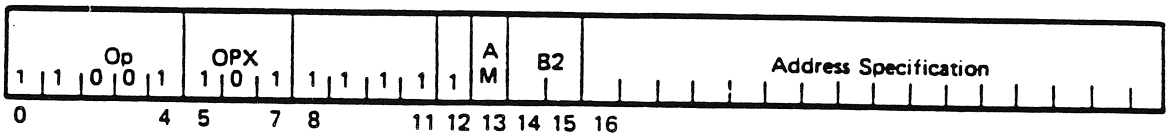
The low-order bit in the EA is used to specify the halfword when M1 is 000 or 010. When M1 is 001 or 011, the low-order bit of the EA should be 0 and will be ignored.

This instruction will always have halfword alignment and will be excluded from automatic index alignment.

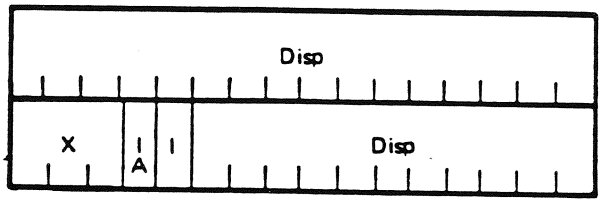
WARNING!

This instruction requires multiple memory accesses. The CPU does not prohibit IOP accesses of the selected main storage location during the time between the fetch of the operand and store of the result. Therefore, this instruction should not be used with any memory locations that might be DMA'd into.

9.3 LOAD PROGRAM STATUS



	AM	Mnemonic	Format
Extended:	0	LPS	D2(B2)
Indexed:	1	LPS[@] [#]	D2(X2,B2)



DESCRIPTION:

Two fullwords starting at the location designated by the fullword operand address replace the contents of the program status registers on the CPU, as described under Program Status Word (Section 2, Figure 2-19).

RESULTING CONDITION CODE:

The code is set or defined by the new PSW.

INDICATORS:

The carry and overflow indicators are set or defined by the new PSW.

PROGRAM INTERRUPT

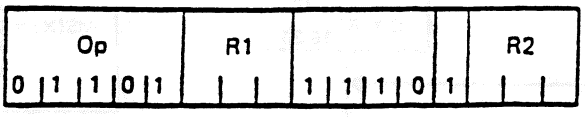
If PSW bits 19 and 20 are set, a fixed-point overflow interrupt will occur.

PROGRAMMING NOTES:

This is a privileged operation and can only be executed when the CPU is in the Supervisor state. This instruction will always have halfword index alignment and will be excluded from automatic index alignment.

PSW bits 40 through 43 are not changed by the load operation.

9.4 MOVE HALFWORD OPERANDS

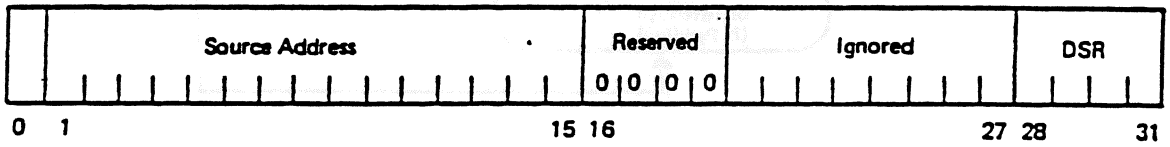


Mnemonic	Format
MVH	R1, R2

DESCRIPTION:

Bits 1 through 15 of the general register specified by R1 contain the offset of the destination address within a specified sector. When bit 0 in R1 is a one, the destination address is determined by concatenating the DSR value in the PSW with the offset. When bit 0 in R1 is a zero, the destination address is determined by concatenating the value in the corresponding DSE register with the offset. Bits 16 through 31 of R1 contain a count of halfwords to be moved. Since its representation uses a signed twos complement integer format, bit 16 (the sign bit) should be zero. A negative count (bit 16 equals 1), or a count equal to 0, indicates no data will be moved.

The content of the general register specified by R2 is as follows:



When bit 0 in R2 is zero, the source address uses an implied DSR of all zeros.

When bit 0 in R2 is one, the source address uses the DSR contained in bits 28-31.

Data (a block of contiguous halfwords) is moved a halfword at a time from a source whose address is determined by concatenating the value of the DSR in R2, with the Source Address in R2, and adding to it the value of the count in bits 16 through 31 of R1, which is decremented by one for each halfword moved. The data is moved to the destination whose address is determined by adding the current value of the count to the destination address. The move is completed when the count becomes zero (see Figure 9-1).

RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

The overflow and carry indicators are not changed.

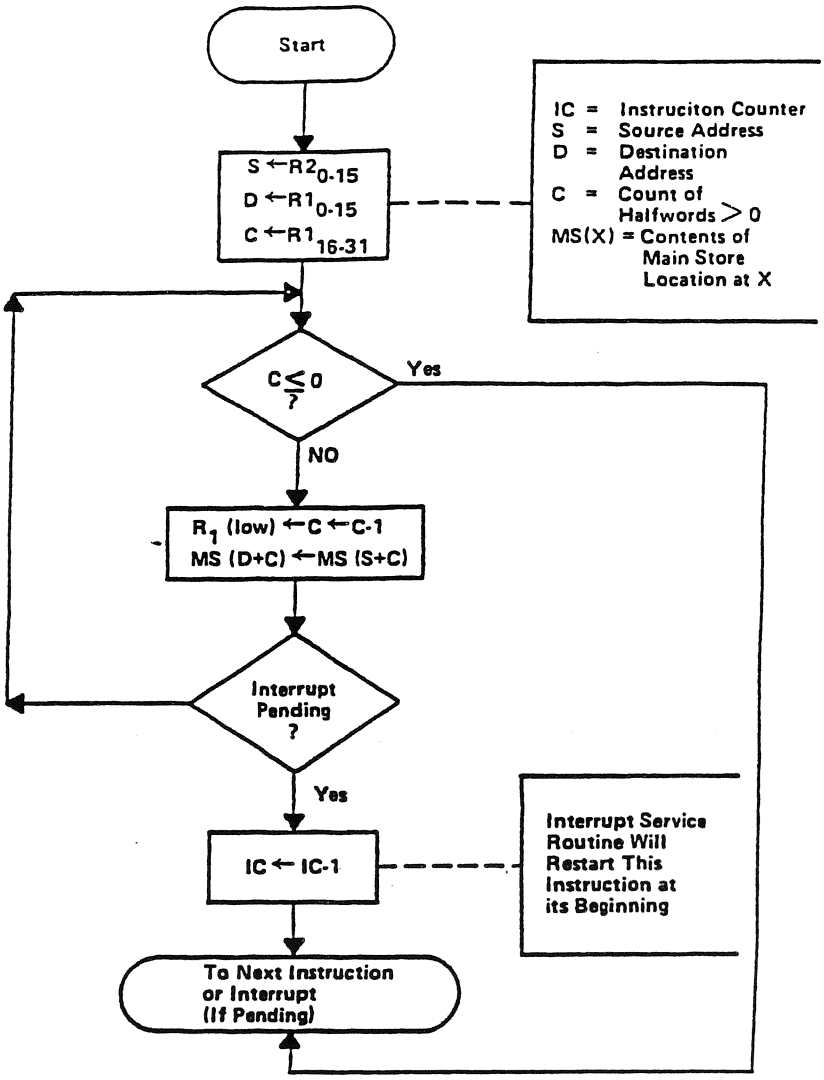


Figure 9-1. Move Halfword Execution

PROGRAMMING NOTES:

As in all instructions, main store addresses (for source and destination) must not be expected to cross 32K sector boundaries, because this instruction will not modify the DSR/DSE. If this is ever attempted, operands will be used from sector zero.

Because the MOVE HALFWORD instruction can execute for a long time, it has been designed to be interruptible by all interrupts except AGE halt, which only interrupts MOVE HALFWORD at the end of the instruction.

When MOVE HALFWORD ends prematurely due to any of the above pending interrupts, the instruction counter will be decremented such that when the interrupt is taken the old PSW contains the instruction address of the move instruction. Note that the count in R1 is modified to reflect the number of halfwords remaining to be moved. This will allow returning to the move instruction so that it can continue to be executed from where it was interrupted. Note that the DSEs associated with registers R4-R7 are not saved/restored by STDM/LDM instructions and therefore may not be saved by standard interrupt handlers.

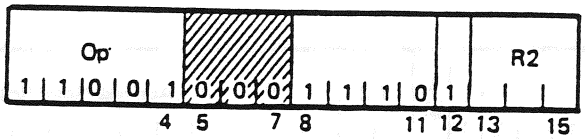
ANOMALY NOTE:

MOVE HALFWORD will not correctly move data when the expanded source address is exactly one greater than the expanded destination address and the most significant bit of R1 and R2 are not equal. To avoid this problem, the programmer should ensure that when the source and destination blocks overlap, the source address is not exactly one halfword greater than the destination address.

The recommended approach when using the MVH to initialize a block of memory is to initialize the last fullword of the block and make the source address 2 halfwords greater than the destination address, thereby moving fullwords instead of halfwords. This avoids the anomaly and executes in half the time.

(This page intentionally left blank)

9.5 SET PROGRAM MASK



Mnemonic	Format
SPM	R2

DESCRIPTION:

The contents of bits 16 through 23 of general register R2 replace the corresponding contents of the current program status registers on the CPU as follows:

Bits 16 and 17 become the new condition code
Bit 18 becomes the new carry indicator
Bit 19 becomes the new overflow indicator
Bit 20 becomes the fixed point overflow mask
Bit 21 (reserved)
Bit 22 becomes the floating point exponent underflow mask
Bit 23 becomes the significance mask.

RESULTING CONDITION CODE:

The code is changed as defined above.

INDICATORS:

The carry, overflow, underflow, and significance indicators are changed as defined above.

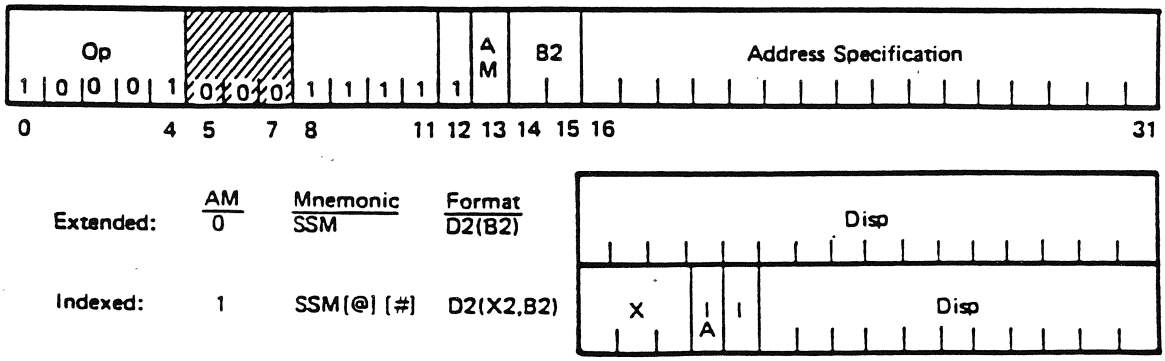
PROGRAM INTERRUPT:

If both bits 19 and 20 are set, the fixed-point overflow interrupt will occur.

PROGRAMMING NOTES:

Bits 5 through 7 are not used by this instruction. These bits should be set to zero as shown above and considered as an op code extension.

9.6 SET SYSTEM MASK



DESCRIPTION:

The halfword second operand replaces bits 32 to 47 of the PSW. This is a privileged operation and can only be executed when the CPU is in the Supervisor state.

RESULTING CONDITION CODE:

The code is not changed.

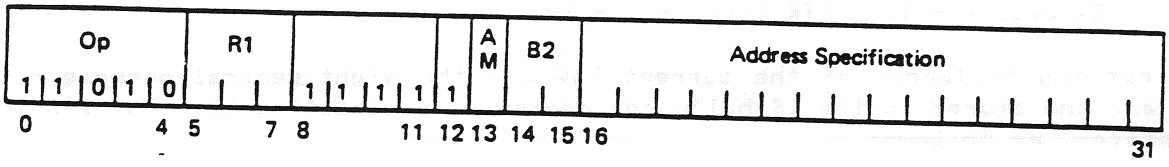
INDICATORS:

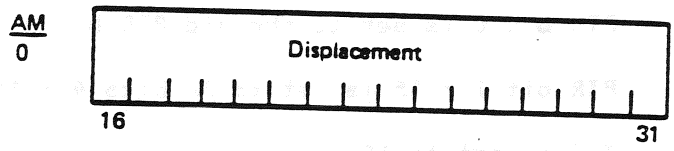
The carry and overflow indicators are not changed by this instruction.

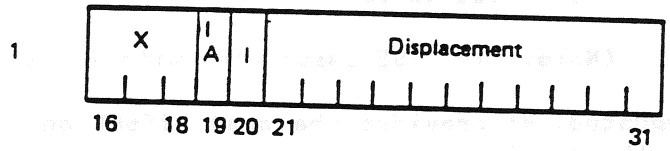
PROGRAMMING NOTES:

Bits 5 through 7 are not used by this instruction. These bits should be set to zero as shown above and considered as an op code extension.

9.7 STACK CALL



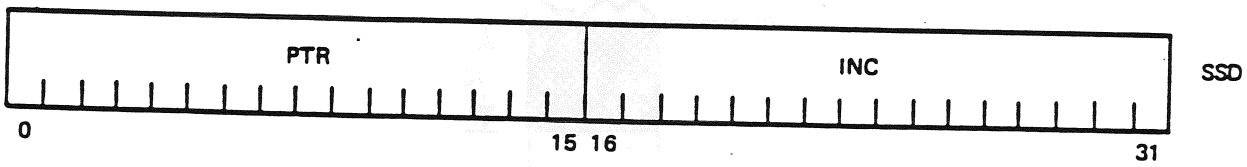




	AM	Mnemonic	Format
Extended:	0	SCAL	R1, D2 (B2)
Indexed:	1	SCAL [@] [#]	R1, D2 (X2, B2)

DESCRIPTION:

This instruction for calling subroutines automatically controls saving bits 0 through 31 of the current PSW and the 8 general registers into a stack space (frame) into main storage. When the Stack Call (SCAL) instruction is to be used, general register R1 must contain a Stack Status Descriptor word (SSD). Likewise, when the corresponding Stack Return (SRET) instruction is to be used to return from the called subroutine, general register R2 must contain an SSD. The contents of the general register containing the SSD are as follows:



First, a branch address is computed. A save area address on the stack is computed from values in the SSD in R1 and either the associated DSE or PSW DSR, as follows:

When bit zero of the PTR is one, the stack space save area address (SA) or pointer, which is represented by a 19-bit machine address, is determined as follows:

SA bits 0 - 3 = DSR contained in the PSW

SA bits 4 - 18 = PTR bits 1-15 + INC

Note: PTR bits 1-15 represent the offset or number of halfwords from the beginning of a specified sector.

When bit zero of the PTR is zero, the stack space save area address is determined as follows:

SA bits 0 - 3 = DSE associated with the register containing the SSD

SA bits 4 - 18 = PTR bits 1-15 + INC

The first two halfwords of the current PSW and the eight general-purpose registers (GPR) are the stored in the 18 halfwords beginning at location SA. The SSD in R1 is now updated, as follows:

PTR bit 0 is set to the old PTR bit 0 value

PTR bit 1 - 15 is set to SA bits 4 - 18

INC is set to 18

(Note: The DSE associated with R1 is not changed.)

When updated, R1 provides the base offset address of the current stack space frame within the specified sector.

Finally, the next instruction is taken from the branch address. This is essentially a BAL instruction which provides an automatic call stack function.

PROGRAMMING NOTES:

PTR is a 16-bit address in R1 which is used to formulate the location of a particular stack frame within a specified sector, i.e., SA (contiguous storage). INC represents the number of halfwords which have currently been used in the stack beyond SA. Since its representation uses a signed twos complement integer format, its sign bit should be zero (see Figure 9-2).

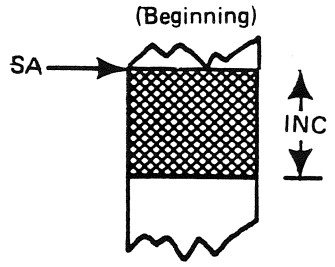


Figure 9-2. Current STACK Status - Prior to SCAL

When SCAL is executed, the new stack save address (SA) is calculated as indicated above, the current PSW and the eight general registers are saved in the new stack save area pointed to by SA so that the stack now appears as in Figure 9-3. Then the PTR in R1 is updated to the sum of the values in PTR + INC, and then INC is set at 18.

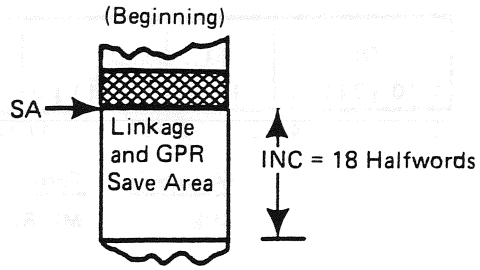


Figure 9-3. STACK Status - Upon Completion of SCAL

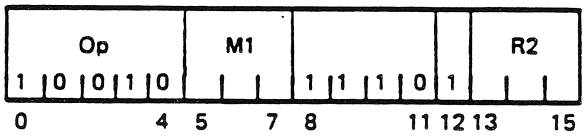
The programmer is free to use additional space in the stack, by simply using R1 as a base, and an offset which is greater than 18 (to avoid destroying the saved GPR contents). However, this additional information will be lost if he issues another SCAL without specifically adjusting INC in R1 to include this new space.

When SRET is executed, the first two halfwords of the PSW and the eight GPRs are automatically loaded from the stack frame save area at location SA. Note that this restores R1 to contain the SSD it had prior to the last SCAL, which means that the stack is automatically restored to the state of Figure 9-2 (refer to STACK RETURN).

PROGRAM INTERRUPTS:

Store protection

9.8 STACK RETURN



Mnemonic	Format
SRET	M1, R2

DESCRIPTION:

When SCAL is used to call a subroutine, the complementary branch instruction SRET is used to leave the called subroutine and return to the conditions prior to the last SCAL. This is a conditional branch instruction in the RR format which provides the first two halfwords of the PSW, and restores the GPR registers to the same state as existed at the time of the SCAL, (i.e., to the extent that the stack space save area has remained unchanged).

The instruction execution first matches the M1 field against the condition code to determine if the branch should be taken. If the branch should not be taken, the instruction terminates at this point. The remaining description applies when the branch should occur.

The stack frame pointer or offset within a specified sector, PTR, is defined by bits 0 though 15 of the general register specified by R2. (Note: This register should be the same as the R1 register specified in the SCAL instruction.) PTR, when concatenated with the sector specification, forms a 19-bit address which is used to address the initial location of the current stack frame. When SRET is invoked, the two halfwords located at this location are moved into the first half of the PSW, i.e., into bits 0 through 31. Next, the 16 halfwords located at the stack frame address + 2, are moved, in order, into the eight general-purpose registers. Finally, instruction execution continues from the address indicated by the active PSW.

RESULTING CONDITION CODE:

The value in the corresponding field is loaded from the stack.

INDICATORS:

The value in the corresponding field is loaded from the stack.

PROGRAMMING NOTES:

The following notes are intended to amplify and clarify the use of the stack and extended call facility.

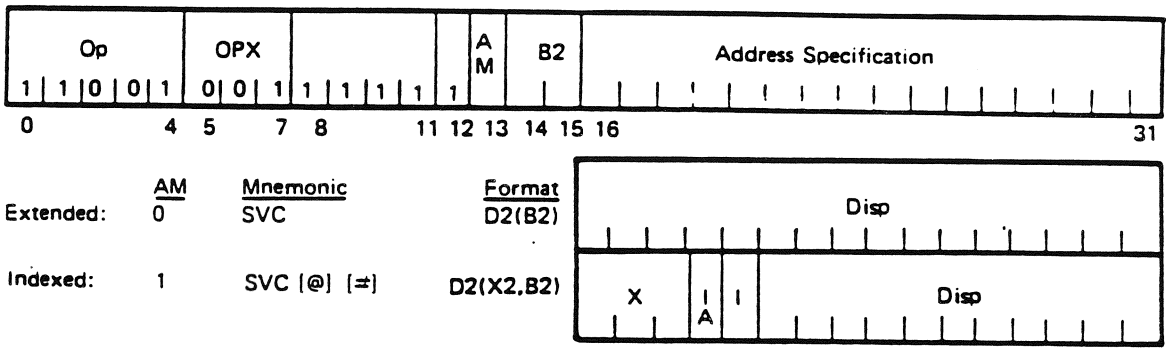
Since the stack is located in main store, any area of the current stack frame can be accessed by standard addressing techniques (i.e., using R1 as a base).

- One of the primary purposes of the stack is automatic register saving and restoring. Another important purpose is the allocation and deallocation of temporary work space, a function often required for efficient use of storage, and for reentrant programs. This latter function can be realized by judiciously increasing the INC value in the SSD.

- The total stack space (i.e., the space taken up by the total stack at any given time) is variable. It grows and shrinks as a function of the depth of the call tree and the amount of workspace used by the various programs. However, in the overall data structure of the total application, there must inevitably be a fixed limit on the amount of main store which can be allocated to the stack. Such limit would presumably be based on either statistics of usage plus a safety factor, or else on a detailed analysis of the usage of all possible call chains. In both cases (the latter as an error detection mechanism) it is important to have some mechanism to stop the call chain, if through some peculiar circumstances, the stack should exceed its allocated space. Unfortunately, there does not appear to be any foolproof scheme. However, most such situations would be caught by appending a few words at the end of the allocated space which have the store protect bit on. Any attempt to store into the appended space would result in a protection violation and interrupt.

- Since the PSW and the eight general-purpose registers are automatically restored on SRET, it is not possible to return results directly to the calling program in the registers. Rather, the value to be returned in a register must be stored into the appropriate slot in the general-purpose register save area in the stack. Then, when the registers are restored, the calling program will, in fact, find the value in the register. At the same time, additional values can be returned to the calling program in the work space in the stack, since the calling program can access that space by addressing relative to the base in R1 (SCAL). (There must, of course, be an agreed-upon convention as to the specific locations in the work space.) Note: The floating point registers are not affected by SCAL and SRET, so variables can be passed in these registers.

9.9 SUPERVISOR CALL



DESCRIPTION:

This instruction causes an interruption and a program status word switch. As a result of this instruction, the interrupt code stored in the old program status word is the 16-bit effective address. This is the only way to enter the supervisor state from the program state. The 4-bit extension (zzzz of Figure 2-18) of the 19-bit effective address is stored in old PSW bits 40 through 43.

RESULTING CONDITION CODE:

The condition code in the stored PSW is not changed by this instruction.

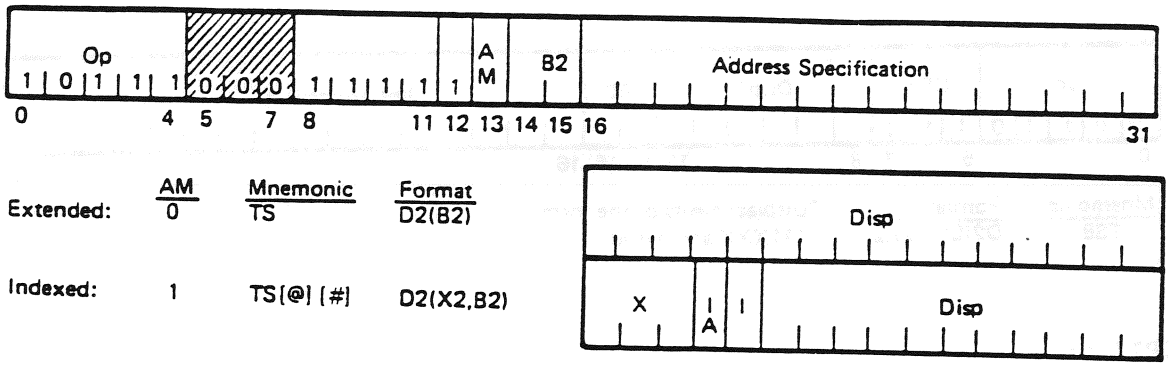
INDICATORS:

The overflow and carry indicators in the stored PSW are not changed by this instruction.

PROGRAMMING NOTES:

The new PSW sets or defines the condition code, overflow indicator, and carry indicator as well as all other bits in the new PSW.

9.10 TEST AND SET



	AM	Mnemonic	Format
Extended:	0	TS	D2(B2)
Indexed:	1	TS[@] [#]	D2(X2,B2)

DESCRIPTION:

Bits in the halfword second operand are tested to set the condition code, and the second operand is set to all ones. No other access to this location, including DMA, is permitted between the fetch and the storing of all ones.

RESULTING CONDITION CODE:

00 The bits are all zeros
11 The bits are mixed with zeros and ones
01 The bits are all ones

INDICATORS:

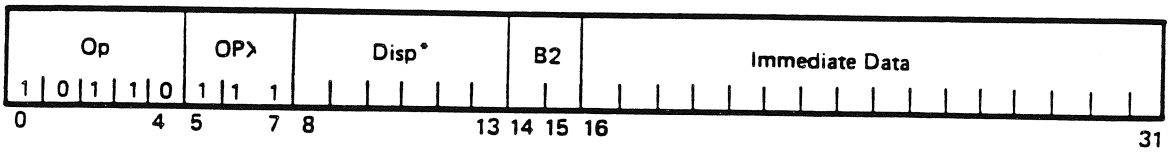
The carry and overflow indicators are not changed by this instruction.

PROGRAMMING NOTES:

TS can be used for the controlling and sharing of a common storage area by more than one program. To accomplish this, a halfword can be designated as control. The desired interlock can be achieved by establishing a program convention in which a zero halfword indicates that the common area is available, but a one means that the area is being used. Each using program then must examine this halfword by means of a Test and Set before making access to the common area. If the test sets the condition code to 00, the area is available for use; if it sets the condition code either 01 or to 11, the area cannot be used. Because Test and Set permits no access to the test halfword between the moment of fetching (for testing) and the moment of storing all ones (setting), the possibility is eliminated of a second program testing the halfword before the first program is able to reset it. Selective bits can be tested by using the TEST AND SET BITS instruction.

Bits 5 through 7 are not used by this instruction. These bits should be set to zero, as shown above and considered an op code extension.

9.11 TEST AND SET BITS



Mnemonic	Format	*Displacements of the form
TSB	D2(B2),Data	111XXX are invalid.

DESCRIPTION:

Bits 16 through 31 of this instruction are treated as halfword immediate data. The immediate data is logically tested with the halfword second operand. The logical sum (OR) of the immediate data and the halfword second operand is formed bit-by-bit. The result replaces the halfword second operand. No other access to this location, including DMA, is permitted between the fetching of the operand and the storing of the result.

RESULTING CONDITION CODE:

00 Either the bits selected by the immediate data are zeros or the immediate data is all zeros.
11 The bits selected by the immediate data are mixed with zeros and ones
01 The bits selected by the immediate data are all ones

INDICATORS:

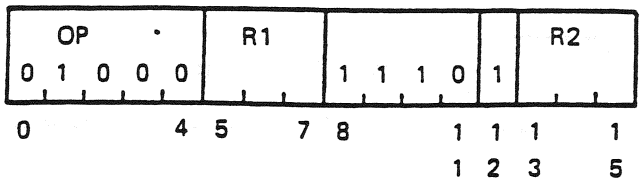
The overflow and carry indicators are not changed by this instruction.

PROGRAMMING NOTES:

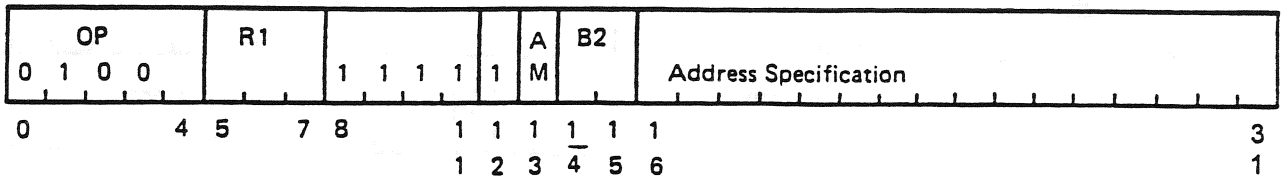
The one bits in the halfword mask specify the bits of the halfword second operand that are set one. The result replaces the halfword second operand. The following table defines this instruction.

TEST AND SET BITS	
Mask	1 1 0 0
Storage	1 0 1 0
Result	1 1 1 0

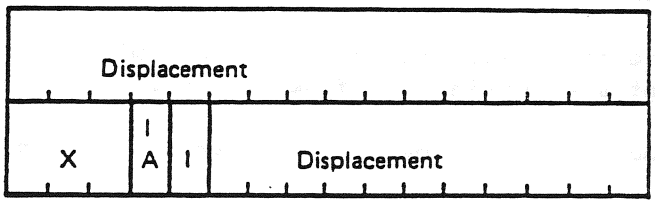
9.12 LOAD EXTENDED ADDRESS



Mnemonic	Format
LXAR	R1, R2



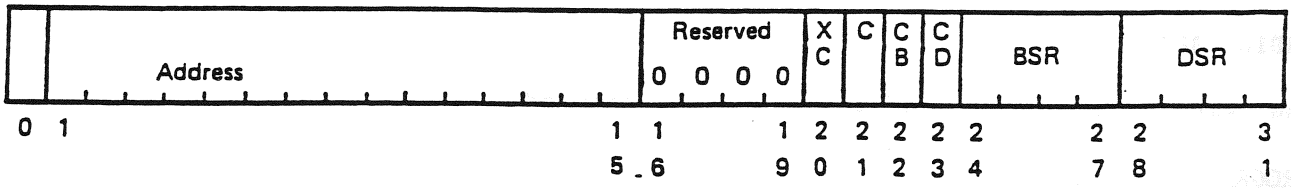
	AM	Mnemonic	Format
Extended:	0	LXA	R1, D2 (B2)
Indexed:	1	LXA[@] [#]	R1, D2 (X2, B2)



DESCRIPTION:

General register R1, and the associated Data Sector Extension (DSE), are initialized from the fullword second operand. Bits 0 and bits 16 through 31 of R1 are zeroed. Bits 1 through 15 of R1 are replaced by bits 1 through 15 of the full word constant, and the DSE associated with R1 is replaced by bits 28 through 31 of the fullword address constant.

The format of the fullword address constant second operand is:



RESULTING CONDITION CODE:

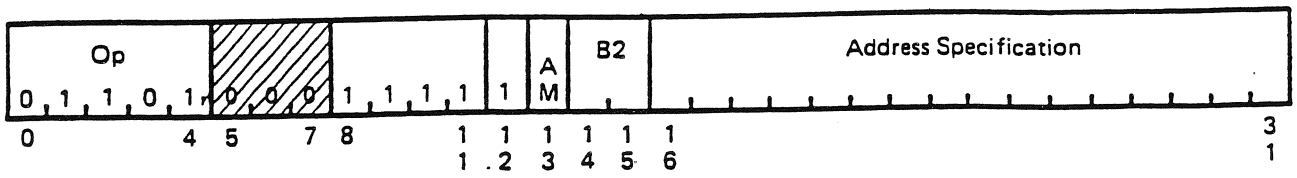
The code is not changed.

INDICATORS:

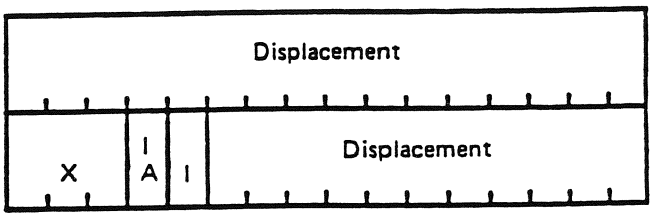
The overflow and carry indicators are not changed.

Note: If the DSR to be loaded and the current DSE are equal, an early out option is executed by microcode to significantly shorten instruction execution time.

9.13 LOAD DSE MULTIPLE



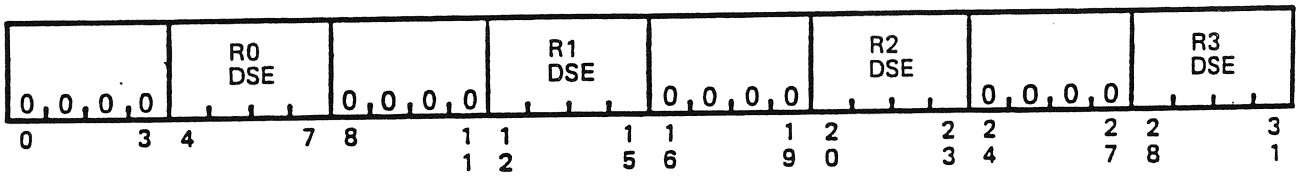
	AM	Mnemonic	Format
Extended:	0	LDM	D2(B2)
Indexed:	1	LDM[@] [#]	D2(X2,B2)



DESCRIPTION:

The four Data Sector Extensions (DSE) corresponding to R0-R3 of the current register set are initialized from the fullword second operand.

The format of the fullword second operand is:



RESULTING CONDITION CODE:

The code is not changed.

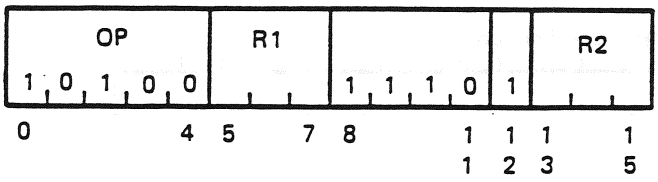
INDICATORS:

The overflow and carry indicators are not changed.

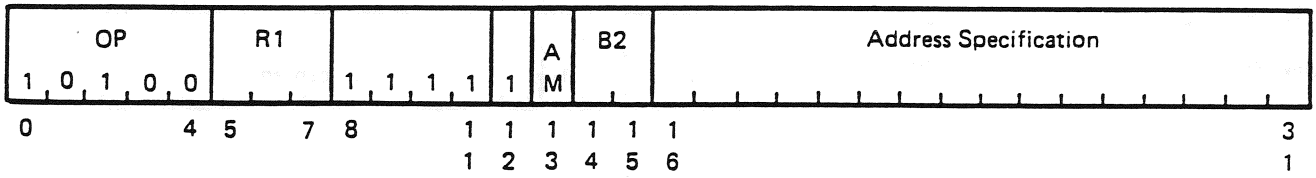
PROGRAMMING NOTES:

Bits 5 through 7 are not used by this instruction. These bits should be set to zero as shown above and considered as an op code extension.

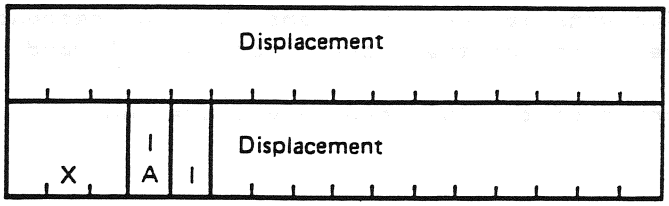
9.14 STORE EXTENDED ADDRESS



Mnemonic	Format
STXAR	R1, R2



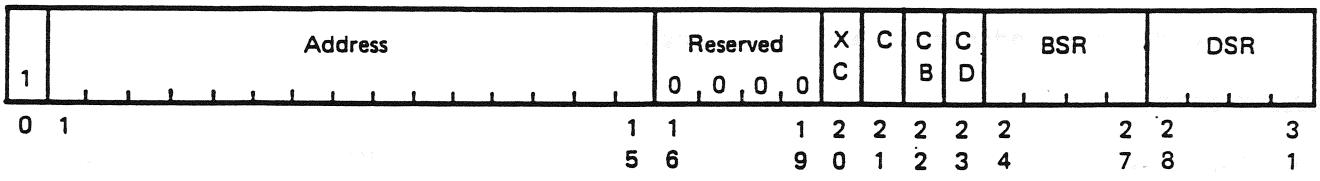
	AM	Mnemonic	Format
Extended:	0	STXA	R1, D2 (B2)
Indexed:	1	STXA[@] [#]	R1, D2 (X2, B2)



DESCRIPTION:

The extended data address contents of R1 plus R1 DSE are stored at the fullword second operand location in fullword address constant format. Bit 0 of the second operand is set to one, bits 1 through 15 are replaced by bits 1 through 15 of R1, bits 28 through 31 are replaced by the contents of R1 DSE, bits 16 through 19 are set to zero, and bits 20 through 27 are unchanged and ignored.

The format of the fullword address constant second operand is:



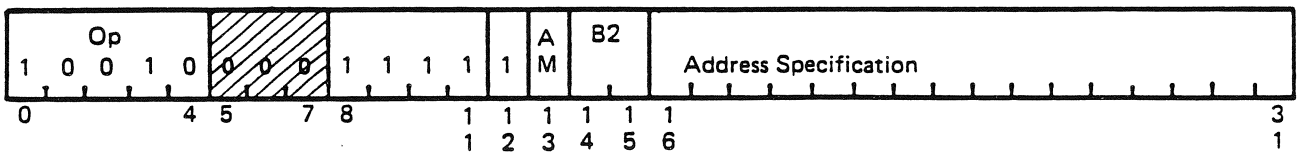
RESULTING CONDITION CODE:

The code is not changed.

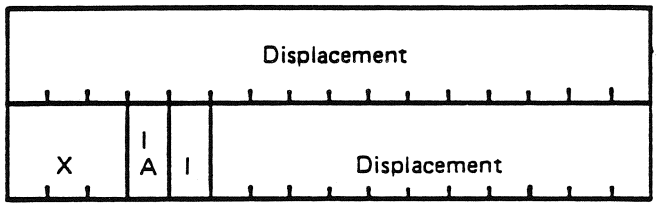
INDICATORS:

The overflow and carry indicators are not changed.

9.15 STORE DSE MULTIPLE



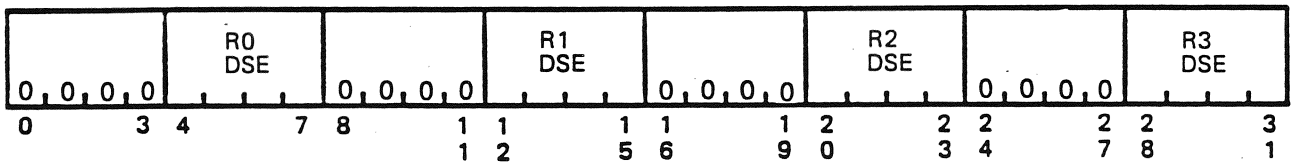
	AM	Mnemonic	Format
Extended:	0	STDM	D2(B2)
Indexed:	1	STDM[@] [#]	D2(X2,B2)



DESCRIPTION:

The four Data Sector Extensions (DSE) corresponding to R0-R3 of the current register set are stored at the location of the fullword second operand.

The format of the fullword second operand is:



RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

The overflow and carry indicators are not changed.

PROGRAMMING NOTES:

Bits 5 through 7 are not used by this instruction. These bits should be set to zero as shown above and considered as an op code extension.

10.0 INTERNAL CONTROL OPERATIONS

A CPU instruction will initiate an Internal Control operation that will perform the following functions, depending on the control word (CW) coding:

- A fullword will be transferred between general register R1 and counter 1 or 2. The high halfword of general register R1 (the most significant halfword) is transferred to or from the main store halfword location 00B0 for counter 1, or 00B1 for counter 2. The low halfword of general register R1 (the least significant halfword) is transferred to or from a 16-bit hardware binary counter 1 or counter 2. Section 2 contains a description of counter operations.

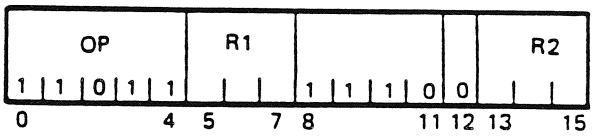
- An AGE command word, specified by bits 16 through 31 of the CW (R2), will be transferred to the AGE interface, and a halfword will be transferred to or from bits 0 through 15 of a general register (R1) and the AGE interface.

- Four discretes will be transferred from bits 0 through 3 of a general register (R1) to the I/O interface.

  0 - XMIT Disable
  1 - BCE Disable
  2 - Spare 1
  3 - Spare 2

- I/O channel reset. The channel reset operation issues a reset to the IO. The IO and CPU uses the signal to reset the IO/CPU interface logic. If an external interrupt 0 has occurred, this command must not be executed until IOP level A interrupt register has been read.

10.1 INTERNAL CONTROL

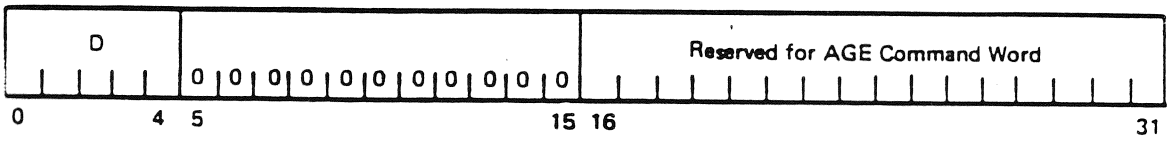


Mnemonic	Format
ICR	R1,R2

DESCRIPTION:

This instruction transfers a fullword to or from the general register specified by R1. Operations are further defined by a control word contained in bits 0 through 31 of the general register specified by R2. The CW format is shown below.

CONTROL WORD (CW):



Legal D Command	Meaning
00000	Read Counter 1
00001	Read Counter 2
00101	Read AGE
01000	Write Counter 1
01001	Write Counter 2
01100	Write Discretes
01101	Write AGE
10000	Channel Reset

No data transfer is associated with the Channel Reset operation.

RESULTING CONDITION CODE:

The code is not changed.

INDICATORS:

The overflow and carry indicators are not changed by this instruction.

PROGRAM INTERRUPTS:

Illegal operation
Privileged instruction
Clears any pending counter interrupts when counter is loaded

PROGRAMMING NOTES:

This is a privileged operation and can only be executed when the CPU is in the Supervisor state.

Command codes which are not defined in this document are illegal and should not be used. Unlike previous versions of this architecture, only the command 10000 causes a channel reset, not the general case 1XXXX.

The AGE command word, specified by bits 16 through 31 of the CW (R2), should be a valid AGE command, otherwise an illegal operation interrupt could occur. The valid AGE commands are: Read AGE (04XX), Load AGE (86XX), Direct AGE (84XX) and AGE Mode Control (87XX). Bits 24 through 31 of the CW are decoded by the AGE to perform a definite function.

When using either Counter 1 or Counter 2 as a counter (rather than as an incremental timer), a possibility exists that the counter could be in error during a single read by 65,536 microseconds (low order bit of location 00B0 or 00B1). This problem can be avoided by doing two consecutive reads and making comparisons to pick the correct reading. To further insure that one of the readings is correct and as a compensation for interrupt processing overhead a value of two (2) is added to the timer when it is read. The write Counter N commands reset the corresponding clock interrupt latch, clearing any pending interrupts.

In addition to the normal shuttle ICR command codes, the following hardware dependent ICR commands are defined for AP-101S series as an aid for diagnostic coding. General use of these codes is not encouraged. If the timer is loaded with a value and read while in the stop condition the value read will be incremented by a value of two.

Originate	Function of ICR	R/W	Code	Code in R2	Typical Execution Time (us)
	Read Soft Error Counters (1-7, low page; 9-15, high page)	R	0002	0002 0000	7.5
	Load Counter 0	W	8801	8801 0000	5.5
	Start Counter 0	W	8802	8802 0000	5.75
	Stop Counter 0	W	8803	8803 0000	5.75
	Load Counter 1	W	8809	8809 0000	5.75
	Start Counter 1	W	880A	880A 0000	5.75
	Stop Counter 1	W	880B	880B 0000	5.75
	Read Counters	R	0802	0802 0000	5.5
	Read Counters	R	0803	0803 0000	5.75
AGE & CPU	Read AGE	R	04XX	2800 04XX	20.25
AGE & CPU	Load AGE	W	86XX	6800 86XX	20.00
AGE & CPU	Direct AGE	W	84XX	6800 84XX	20.25
AGE & CPU	AGE Mode Control	W	87XX	6800 87XX	20.25
MMU	Read and Clear MFER (Figure 10-1)	R	1408	1408 0000	7.25
MMU	Read MMU (Figure 10-1)	R	140A	140A 0000	7.25
MMU	Read EDAC Address Reg. 00-15	R	140F	140F 0000	7.25
MMU	Read MFER (Figure 10-1)	R	140C	140C 0000	7.25
MMU	Clear Soft Error Counter	W	9402	9402 0000	5.75

Bit	MFER Function	MMU Function
0	Spare = 0	Spare = 0
1	Illegal Address = 1	Illegal Address = 1
2	Spare = 0	Spare = 0
3	Spare = 0	Spare = 0
4	Spare = 0	Spare = 0
5	MMP Store Protect = 1	MMP Store Protect = 1
6	MMP Store Protect Bits Miscompare = 1 (DRAM only)	MMP Store Protect Bits Miscompare = 1 (DRAM only)
7	CPU/IOP Single Bit Memory Error = 1	CPU/IOP Single Bit Memory Error = 1
8	IOP Multibit Memory Error = 1	IOP Multibit Memory Error = 1
9	IOP Store Protect Error = 1	IOP Store Protect Error = 1
10	Spare = 0	Spare = 0
11	CPU Multibit Error = 1	1750 Block Protect Enabled = 1
12	Spare = 0	Spare = Float
13	Spare = 0	Logic 0
14	Illegal I/O Command = 1	CMOS Memory Installed = 0
15	Spare = 0	MMP = 0; 1750 = 1

Figure 10-1. MFER/MMU Registers

11.0 AP-101S SHUTTLE INSTRUCTION SET

11.1 EFFECTIVE ADDRESS GENERATION SUMMARY CHART

	SRS, SI Formats	RS Format				
		Extended Addressing (AM=0)	Indexed Addressing (AM=1)			
			IA	I	X=000	X=$\overline{000}$
					PEA=(B)+DISP	
I|2≠11	EA=(B)+DISP	EA=(B)+DISP	00		EA=IC+PEA	EA=$(X)_{0-15}$+PEA
			01		EA=IC-PEA	EA=$(X)^{*}_{0-15}$+PEA
			10		EA=MS(PEA)	EA=MS(PEA)+$(X)_{0-15}$
			11		EA=MS(PEA)**	EA=MS(PEA)***+$(X)_{0-15}$
					PEA=DISP	
B2=11	EA=(B)+DISP	EA=DISP	00		EA=IC+PEA	EA=$(X)_{0-15}$+PEA
			01		EA=IC-PEA	EA=$(X)^{*}_{0-15}$+PEA
			10		EA=MS(PEA)	EA=MS(PEA)+$(X)_{0-15}$
			11		EA=MS(PEA)**	EA=MS(PEA)***+$(X)_{0-15}$

Definitions

EA	Effective address, main storage address of second operand
PEA	Preliminary effective address
(RN)	Contents of bits 0-15 of general register N specified by B2 or X
RN	General register "N", where N = 0 to 7
(B)	Contents of bits 0-15 of general register specified by the B2 field
B2	B field of SRS, SI, or RS format instruction
MS( )	Contents of the main storage location specified by the contents of the parenthesis
DISP	Displacement field of instruction
X	X field of RS format instruction with indexed mode of addressing
$(X)_{0-15}$	Most significant halfword (bits 0-15) of the content of index register X automatically aligned.
AM	AM (addressing mode) field of RS format instruction
IA	IA (indirect address) field of RS format instruction with the indexed mode of addressing
I	I field of RS format instruction with indexed mode of addressing
IC	Updated Instruction Counter
*	Automatic Index Modification
**	Automatic Storage Modification
***	Direct Storage Addressing with/without Post Indexing

X	INDEX VALUE	X	INDEX VALUE
000	Zero	100	(R4)
001	(R1)	101	(R5)
010	(R2)	110	(R6)
011	(R3)	111	(R7)

(This page intentionally left blank)

12.0 AP-101S INSTRUCTION REPERTOIRE

12.1 SHUTTLE INSTRUCTION SET

Name	Mnemonics	Format
Fixed Point Operations		
Add	AR,A	RR,SRS,RS
Add Halfword	AH	SRS,RS
Add Halfword Immediate	AHI	RI
Add to Storage	AST	RS
Compare	CR,C	RR,SRS,RS
Compare Between Limits	CBL	RR
Compare Halfword	CH	SRS,RS
Compare Halfword Immediate	CHI	RI
Compare Immediate with Storage	CIST	SI
Divide	DR,D	RR,SRS,RS
Exchange Upper and Lower Halfwords	XUL	RR
Insert Address Low	IAL	SRS,RS
Insert Halfword Low	IHL	RS
Load	LR,L	RR,SRS,RS
Load Address	LA	SRS,RS
Load Arithmetic Complement	LCR	RR
Load Fixed Immediate	LFXI	RR
Load Halfword	LH	SRS,RS
Load Multiple	LM	RS
Modify Storage Halfword	MSTH	SI
Multiply	MR,M	RR,SRS,RS
Multiply Halfword	MH	SRS,RS
Multiply Halfword Immediate	MHI	RI
Multiply Integer Halfword	MIH	RS
Store	ST	SRS,RS
Store Halfword	STH	SRS,RS
Store Multiple	STM	RS
Subtract	SR,S	RR,SRS,RS
Subtract from Storage	SST	RS
Subtract Halfword	SH	SRS,RS
Tally Down	TD	SRS,RS

Name	Mnemonics	Format
Branch Operations		
Branch and Link	BALR,BAL	RR,RS
Branch and Index	BIX	RS
Branch on Condition	BCR,BC	RR,RS
Branch on Condition Backward	BCB	SRS
Branch on Condition (Extended)	BCRE	RR
Branch on Condition Forward	BCF	SRS
Branch on Count	BCTR,BCT	RR,RS
Branch on Count Backward	BCTB	SRS
Branch on Overflow and Carry	BVCR,BVC	RR,RS
Branch on Overflow and Carry Forward	BVCF	SRS
Shift Operations		
Normalize and Count	NCT	RR
Shift Left Logical	SLL	SRS
Shift Left Double Logical	SLDL	SRS
Shift Right Arithmetic	SRA	SRS
Shift Right Double Arithmetic	SRDA	SRS
Shift Right Logical	SRL	SRS
Shift Right Double Logical	SRDL	SRS
Shift Right and Rotate	SRR	SRS
Shift Right Double and Rotate	SRDR	SRS
Logical Operations		
AND	NR,N	RR,SRS,RS
AND Halfword Immediate	NHI	RI
AND Immediate with Storage	NIST	SI
AND to Storage	NST	RS
Exclusive-OR	XR,X	RR,SRS,RS
Exclusive-OR Halfword Immediate	XHI	RI
Exclusive-OR Immediate with Storage	XIST	SI
Exclusive-OR to Storage	XST	RS
OR	OR,O	RR,SRS,RS
OR Halfword Immediate	OHI	RI
OR to Storage	OST	RS
Search Under Mask	SUM	RR
Set Bits	SB	SI
Set Halfword	SHW	SRS,RS
Test Bits	TB	SI
Test Register Bits	TRB	RI
Test Halfword	TH	SRS,RS
Zero Bits	ZB	SI
Zero Register Bits	ZRB	RI
Zero Halfword	ZH	SRS,RS

Name	Mnemonics	Format
Floating Point Operations		
Add (Long Operand)	AEDR,AED	RR,RS
Add (Short Operands)	AER,AE	RR,SRS,RS
Compare (Short Operand)	CER,CE	RR,RS
Compare (Long Operand)	CEDR,CED	RR,RS
Convert to Fixed Point	CVFX	RR
Convert to Floating Point	CVFL	RR
Divide (Extended Operand)	DEDR,DED	RR,RS
Divide (Short Operand)	DER,DE	RR,SRS,RS
Load (Long Operand)	LED	RS
Load (Short Operand)	LER,LE	RR,SRS,RS
Load Complement (Short Operand)	LECR	RR
Load Fixed Register	LFXR	RR
Load Floating Immediate	LFLI	RR
Load Floating Register	LFLR	RR
Midvalue Select (Short Operands)	MVS	RS
Multiply (Extended Operand)	MEDR,MED	RR,RS
Multiply (Short Operand)	MER,ME	RR,SRS,RS
Subtract (Long Operand)	SEDR,SED	RR,RS
Subtract (Short Operand)	SER,SE	RR,SRS,RS
Store (Long Operand)	STED	RS
Store (Short Operand)	STE	SRS,RS
Special Operations		
Diagnose*	-	RS
Store Extended Address	STXA	RR,RS
Store DSE Multiple	STDM	RS
Insert Storage Protect Bits*	ISPB	RS
Load Program Status*	LPS	RS
Move Halfword Operands	MVH	RR
Set Program Mask	SPM	RR
Set System Mask*	SSM	RS
Stack Call	SCAL	RS
Stack Return	SRET	RR
Load DSE Multiple	LDM	RS
Load Extended Address	LXA	RR,RS
Supervisor Call	SVC	RS
Test and Set	TS	RS
Test and Set Bits	TSB	SI
Internal Control Operations		
Internal Control*	ICR	RR
I/O Operations		
Program Controlled Input/Output*	PC	RR

*Privileged Instruction

(This page intentionally left blank)

13.0 AP-101S OP CODE ASSIGNMENTS

OP 2,3	OP0, OP1			
	00	01	11	10
	OP 04 = 1			
00	SRS SUBTRACT RR SUBTRACT RR_2 COMP BTWN LMTS RS SUBTRACT RS_2 SUB FRM STO	SRS DIVIDE RR DIVIDE RR_2 COMP FL ST RS^2DIVIDE RS_2 COMP FL ST	RR BROV & CRY RR_2 SET PROG MSK RS^2BROV & CRY RS_2 LM, LPS, STM, SVC	SBS SUB HW RR LOAD FL IMM RS SUB HW RS_2 SET SYST MASK
01	SRS LOAD RR LOAD RR COMP FL LN RS LOAD RS_2 COMP FL LN	SRS SUBTRACT FL ST RR SUBTRACT FL ST RR SUBTRACT FL IN RS^2SUBTRACT FL ST RS_2 SUBTRACT FL LN	SRS BR RELATIVE RR ICR I/O RR_2 PC RS^2BIX	SRS LOAD HW RR_2 SUM RS^2LOAD HW RS_2 MIH
11	SRS STORE FL ST RR CONV TO FXD RR_2 CONV TO FLT RS^2STORE FL ST	SRS LOAD FL ST RR LOAD FL ST RR_2 LOAD COMPL FT ST RS LOAD FL ST RS_2 LOAD FL LN	SRS REG SH DBL SRS COMPUTED SH DBL	SRS STO HW RR LOAD FX IM RS STO HW RS_2 TST & SET
10	SRS OR RR OR RR_2 LOAD FLTG REG RS^2OR RS_2 OR TO STORE	SRS DIVIDE FL ST RR DIVIDE FL ST RR MOVE HALFWORD OPS RS DIVIDE FL ST TEST 2 (LRS) RS LOAD DSE MTPL	SRS LOAD ADDRESS RR_2 LOAD ARITH COMP RS LOAD ADDRESS RS_2 INSTR PROT BITS	SRS MULTIPLY HW TEST 3 (RR_2) RS MULTIPLY HW
	OP 04 = 0			
00	SRS ADD RR ADD RR_2 XU&L HW RS^2ADD RS_2 ADD TO STORE	SRS MULTIPLY RR MULTIPLY RS MULTIPLY RS LOAD DSE RR LOAD DSE	RR BR ON COND RR_2 BR ON COND EXT RS^2BR ON COND RS_2 DIAGNOSE	SRS ADD HW RR_2 RS^2ADD HW RS_2 INSERT HW LOW
01	SRS COMP DVD FL LN RR COMP RR_2 DVD FL LONG RS^2COMP RS_2 DVD FL LN	SRS ADD FL ST BR ADD FL ST RR_2 ADD FL LN RS^2ADD FL ST RS_2 ADD FL LN	RR BR ON CT TEST 4 (RR) RS BR ON CT RS_2 STACK CALL	SRS COMP HW RR_2 STACK RTRN RS^2COMP HW RS STORE DSE MTPL
11	SRS STORE RR_2 MPY FL LN RS^2STORE RS_2 MPY FL LN	SRS XOR RR XOR RS XOR RS XOR TO STORE	SRS RG SH SING SRS COMPUTED SH SING	IEXP RR, RS R1 = OPX
10	SRS AND RR AND RR_2 LOAD FX RG RS^2AND RS_2 AND TO STORE	SRS MULTIPLY FL ST RR MULTIPLY FL ST TEST 1 (RR) RS MULTIPLY FL ST RS_2 MID VALUE SLCT	SRS INSRT ADD LO RR BR & LNK RS BR & LNK RR_2 NORM & CNT RS_2^2 INSTR ADD LO	IMPL SRS, RS R1 = OPX TEST 3 (LRS) RR STORE DSE RS STORE DSE

Notes: OP12 = 1 Causes either RR_2 or RS_2 Operations
FL ST - Floating-Point (Short Operands)
FL LN - Floating-Point (Long Operands)
HW - Halfwords

Op Code 00011 with OP12 = 1 is reserved

OP	R1=OPX	RR_2	RS_2
11001	000	Set Program Mask	Store Multiple
11001	001	Reserved	Supervisor Call
11001	010	Reserved	Reserved
11001	011	Reserved	Reserved
11001	100	Reserved	Load Multiple
11001	101	Reserved	Load Program Status
11001	110	Reserved	Reserved
11001	111	Reserved	Reserved

IMPLIED IMMEDIATE

10100	000	Tally Down	SRS,RS
10100	001	Zero Halfword	SRS,RS
10100	010	Set Halfword	SRS,RS
10100	011	Test Halfword	SRS,RS
10100	100	Reserved	
10100	101	Reserved	
10100	110	Reserved	
10100	111	Reserved	

EXPLICIT IMMEDIATE

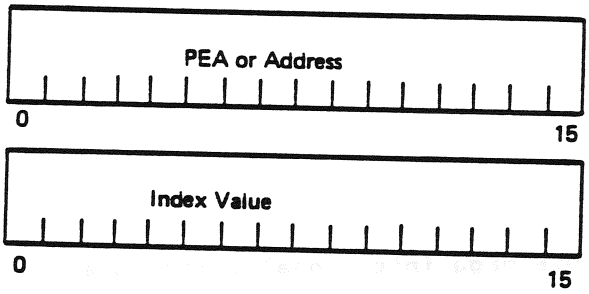
OP	R1=OPX	RR	RR_2	SRS
10110	000	Add Half Immediate	Reserved	Modify Storage Halfword
10110	001	Zero Register Bits	Reserved	Zero Bits
10110	010	OR Half Immediate	Reserved	Set Bits
10110	011	Test Register Bits	Reserved	Test Bits
10110	100	XOR Half Immediate	Reserved	XOR Immediate With Store
10110	101	Comp Half Immediate	Reserved	Compare Immediate With Store
10110	110	AND Half Immediate	Reserved	AND Immediate With Store
10110	111	Mult Half Immediate	Reserved	Test and Set Bits

14.0 AP-101S INSTRUCTION SET

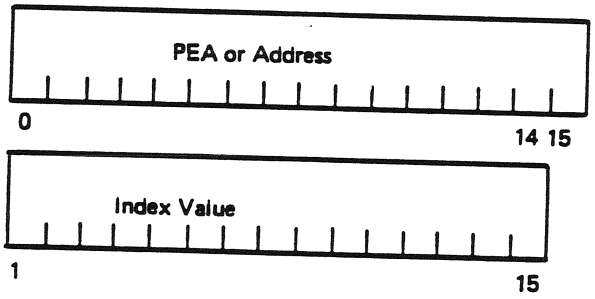
14.1 AUTOMATIC INDEX ALIGNMENT DESCRIPTION

Index alignment occurs automatically. That is, bits 0 through 15 of the general register specified by X specify entities. The identity of this entity is explicitly defined by the particular operation being executed.

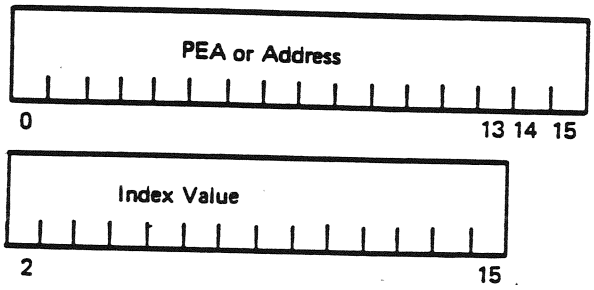
Halfword operations align index value bit 15 with the least significant bit of the PEA (described in Section 2) or the ADDRESS portion of an indirect address pointer (described in Section 2). It should be noted that the LOAD MULTIPLE, STORE MULTIPLE, LOAD PROGRAM STATUS, and INSERT STORAGE PROTECT BITS instructions are excluded from automatic index alignment and have a halfword index alignment.



Likewise, fullword operations functionally shift the index value one position to the left, prior to alignment. Note that bit 0 of the index value is lost.



Likewise, doubleword operations functionally shift the index value two positions to the left prior to alignment. Note that bits 0 and 1 of the index value are lost.



(This page intentionally left blank)

15.0 AP-101S DIAGNOSTIC FUNCTIONS

This section describes the several operations available, any special requirements, any results, and provides a timing estimate for each operation, assuming successful completion.

The purpose of the Diagnose Instruction is to provide tests of hardware that are critical to the proper operation of the Central Processor Unit (CPU) and to provide access to hardware not accessible or easily testable using the implemented instruction set. In addition, tests that might require an inordinate amount of time and/or memory are partially or wholly implemented in microcode.

All Diagnose Instructions, unless stated otherwise in the description, execute in 50 microseconds or less and do not delay I/O for more than 3 microseconds. The Read and Write System and Program Mask Diagnose Instructions do not affect the condition code. All other Diagnose Instructions set the condition code zero for pass and negative for fail.

The execution of a hardcore diagnostic test terminates at the detection of the first error for all tests except interrupt page tests. Interrupt page test errors are detected during execution of EA SCAN 5 assist. Detection of an error shall cause diagnostic data to be logged out in locations x'104' through x'12D'.

The execution of a Diagnose Instruction terminates at the detection of the first error. Detection of an error will cause diagnostic data to be logged out in locations X'104' through X'12D'.

The layout of the log out area is described below:

Location	Description
104	Unlogged error count
105	Error Flag
106-7	FA register
108-9	FB register
10A-B	FC register
10C	bits 0-7 EI register
	bits 8-15 EA register
10D	bits 0-7 EB register
	bits 8-15 X'00'
10E-11D	Local Store CPU Sector 0 registers
11E-12D	Local Store CPU Sector 7 registers

Unlogged error count contains a count of the number of errors which were not logged because the log area was occupied. If error count and error flag are both zero, the error flag is set to X'01' and error data is logged out. If error count or error flag are not zero, error count is incremented and error data is not logged out. Location X'111' (Local Store CPU Sector 0 Register 3) contains the error code.

Diagnose, unlike other instructions, does not follow the rule that programming errors are distinguished from equipment errors. Improper use of Diagnose may result in false machine-check indications or may cause actual machine malfunctions to be ignored. It may also alter other aspects of system operation, including instruction

execution and channel operation, to an extent that the operation does not comply with that specified in this manual. As a result of the improper use of Diagnose, the system may be left in such a condition that the power-on reset function must be performed.

The following descriptions define the hexadecimal value effective address (command word) required to select each of the microsequences and an estimate of the average execution time for RS Extended Addressing without a Base Register. Actual execution times will depend on pipe prefetching and conflicts, and I/O interference. The Diagnose descriptions also include any programming restrictions and a description of the data returned when an error is detected. All effective addresses not described here are reserved and shall not be used. The result of using a reserved effective address is indeterminate.

The CPU Hardcore Microcode Test is separated into nine tests to facilitate the 50 microsecond interruptibility requirement.

Hexadecimal Contents of Effective Address	Description
0000	CPU HARDCORE MICROCODE TEST 0 (31 microseconds) The CPU hardcore microcode test 0 verifies the basic operation of a portion of the CPU pages. The functions tested are: fraction ALU, fraction zero detect, fraction A, B, and C registers and controls, exponent ALU, exponent A, B, and I registers and controls. At completion of the test, the condition code shall be set zero if no error occurred. The condition code shall be set negative if an error occurred.
0100	CPU HARDCORE MICROCODE TEST 1 (27 microseconds) The CPU hardcore microcode test 1 verifies the basic operation of a portion of the CPU pages. The function tested is the two- and four-way Y-BUS micro branch hardware. At completion of the test, the condition code shall be set zero if no error occurred. The condition code shall be set negative if an error occurred.
0200	CPU HARDCORE MICROCODE TEST 2 (45 microseconds) The CPU hardcore microcode test 2 verifies the basic operation of a portion of the CPU pages. The function tested is the 16-way Y-BUS micro branch hardware. At completion of the test, the condition code shall be set zero if no error occurred. The condition code shall be set negative if an error occurred.
0300	CPU HARDCORE MICROCODE TEST 3 (46 microseconds) The CPU hardcore microcode test 3 verifies the basic operation of a portion of the CPU pages. The function tested is the two-way STAT A bit microbranch hardware. At completion of the test, the condition code shall be set zero if no error occurred. The condition code shall be set negative if an error occurred.

Hexadecimal Contents of Effective Address	Description
0400	CPU HARDCORE MICROCODE TEST 4 (50 microseconds) The CPU hardcore microcode test 4 verifies the basic operation of a portion of the CPU pages. The function tested is the four-way STAT A bit microbranch hardware. At completion of the test, the condition code shall be set zero if no error occurred. The condition code shall be set negative if an error occurred.
0401	CPU HARDCORE MICROCODE TEST 8 (16 microseconds) The CPU hardcore microcode test 8 verifies the basic operation of a portion of the CPU pages. The function tested is local store addressing. At completion of the test, the condition code shall be set zero if no error occurred. The condition code shall be set negative if an error occurred.
0500	CPU HARDCORE MICROCODE TEST 5 (5.2 milliseconds) The CPU hardcore microcode test 5 verifies the basic operation of a portion of the CPU pages. The function tested is the Constant PROM. At completion of the test, the condition code shall be set zero if no error occurred. The condition code shall be set negative if an error occurred. Note: This test will delay any pending interrupts more than 5.2 milliseconds and so should not be executed in an operational environment which cannot tolerate this delay.
0600	CPU HARDCORE MICROCODE TEST 6 (78 microseconds) The CPU hardcore microcode test 6 verifies the basic operation of a portion of the CPU, MMU, and memory pages. The functions tested are: basic memory addressing and data, and basic store protect bit hardware. At completion of the test, the condition code shall be set zero if no error occurred. The condition code shall be set negative if an error occurred. Note: This test will delay any pending interrupts more than 78 microseconds, and will delay any pending I/O operation more than 68 microseconds and so should not be executed in an operational environment which does not control I/O activity.

Hexadecimal Contents of Effective Address	Description
0700	CPU HARDCORE MICROCODE TEST 7 (14.2 microseconds) The CPU hardcore microcode test 7 verifies the basic operation of a portion of the CPU pages. The functions tested are the Real Time Communications Channel register and the Execution Unit Program Counter. The CCU need not be attached to execute this test. At completion of the test, the condition code shall be set zero if no error occurred. The condition code shall be set negative if an error occurred. Note: This Diagnose Instruction cannot be macrostepped.
1000	READ PROGRAM AND SYSTEM MASK (PSMR) (4.8 microseconds) The Read Program and System Mask reads bits 16-47 of the active PSW and places them in bits 0-31 of register R1. The condition code is not altered.
2000	LOCAL STORE CPU SECTOR TEST (LSMICRO) (40 microseconds) The Local Store Sector test shall perform a write/read test on one sector of CPU local store. Register R1 bits 1-4 shall contain the register set to be tested. The contents of the local store sector tested will not be altered. At completion of the test, the condition code shall be set zero if no error occurred. The condition code shall be set negative if an error occurred.
2001	LOCAL STORE CONSTANT SECTOR TEST (LSMICRO) (38 microseconds) The Local Store Sector test shall perform a write/read test on one sector of constant local store. Register R1 bits 1-4 shall contain the register set to be tested. The contents of the local store sector tested will not be altered. At completion of the test, the condition code shall be set zero if no error occurred. The condition code shall be set negative if an error occurred.
3000	WRITE PROGRAM AND SYSTEM MASK (PSMW) (9.2 microseconds) The Write Program and System Mask command loads bits 16-47 of the active PSW from bits 0-31 of register R1.

Hexadecimal Contents of Effective Address	Description
4000	LOCAL STORE READ ASSIST (LSREAD) (5.8 microseconds) The Local Store Read Assist allows any local store register to be read. Bits 0-15 of register R1 shall contain the address of the local store register to be read. Bits 0-6 shall be zero. Bit 7 shall be one to reference the Constant Sector and zero to reference the CPU Sector. Bits 8-11 shall indicate the sector number and bits 12-15 shall indicate the register number. The contents of the local store register shall be placed in bits 16-31 of register R1.
4100	LOCAL STORE WRITE ASSIST (LSWRITE) (13 microseconds) The Local Store Write Assist allows any local store register to be written. Bits 0-15 of register R1 shall contain the address of the local store register to be written. Bits 0-6 shall be zero. Bit 7 shall be one to reference the Constant Sector and zero to reference the CPU Sector. Bits 8-11 shall indicate the sector number and bits 12-15 shall indicate the register number. Bits 16-31 of register R1 shall be placed in the local store register referenced by bits 0-15 of register R1. After execution of this assist, the PSW will be updated and the pipe purged so that a write to the PSW portion of local store will be processed immediately.
7000	H-BUS READ (HBUSR) (6.6 microseconds) The H-BUS Read command allows any Internal I/O (IIO) command to be read. It is the responsibility of the macro programmer to ensure that only valid commands are issued. Invalid commands may cause loss of control or bus timeouts. Bits 0-15 of register R1 shall contain the Internal Bus command. Bits 16-31 of register R1 shall contain the data to be sent with the read command, if any is required. Bits 16-31 of register R1 shall contain the data read. The execution time of this Diagnose is IIO command dependent.
7001	H-BUS WRITE (HBUSW) (4.2 microseconds) The H-BUS Write command allows any Internal I/O (IIO) command to be written. It is the responsibility of the macro programmer to ensure that only valid commands are issued. Invalid commands may cause loss of control or bus timeouts. Bits 0-15 of register R1 shall contain the Internal Bus command. Bits 16-31 of register R1 shall contain the data to be written. The execution time of this Diagnose is IIO command dependent.

Hexadecimal Contents of Effective Address	Description
7100	DETECT STORES INTO IU FILE (BSTAT6) (9.6 microseconds) The Detect Stores into IU File microcode sets bit 6 of the B STAT Reg. When this status bit is set, the CPU hardware checks for conflicts within the IU file. When conflicts are detected, the file is purged.
7101	DISREGARD STORES INTO IU FILE (BSTAT6) (9.6 microseconds) The Disregard Stores into IU File microcode resets bit 6 of the B STAT Reg. When this status bit is reset, no checks for conflicts within the IU file are performed. The pipeline will not be purged.

The Interrupt Page Command PLA Test is separated into five tests to facilitate the 50 microsecond interruptibility requirement.

Hexadecimal Contents of Effective Address	Description
8000	INTERRUPT PAGE COMMAND PLA TEST 0 (CINTCMD) (32 microseconds) The Interrupt Page Command PLA test 0 verifies the operation of a portion of the interrupt page command PLA. The CPU microcode informs the interrupt page that the test will be run and then sends a sequence of MMP IIO commands to the interrupt page. The interrupt page verifies that the proper commands were received in the proper sequence. Any errors detected by the interrupt page during this test shall be saved in the interrupt page Scan Register which can be read using the EA Scan 5 Assist Diagnose. The condition code will always be zero.
8100	INTERRUPT PAGE COMMAND PLA TEST 1 (CINTCMD) (38 microseconds) The Interrupt Page Command PLA test 1 verifies the operation of a portion of the interrupt page command PLA. The CPU microcode informs the interrupt page that the test will be run and then sends a sequence of MMP IIO commands to the interrupt page. The interrupt page verifies that the proper commands were received in the proper sequence. Any errors detected by the interrupt page during this test shall be saved in the interrupt page Scan Register which can be read using the EA Scan 5 Assist Diagnose. The condition code will always be zero.
8200	INTERRUPT PAGE COMMAND PLA TEST 2 (CINTCMD) (33 microseconds) The Interrupt Page Command PLA test 2 verifies the operation of a portion of the interrupt page command PLA. The CPU microcode informs the interrupt page that the test will be run and then sends a sequence of MMP IIO commands to the interrupt page. The interrupt page verifies that the proper commands were received in the proper sequence. Any errors detected by the interrupt page during this test shall be saved in the interrupt page Scan Register which can be read using the EA Scan 5 Assist Diagnose. The condition code will always be zero.

Hexadecimal Contents of Effective Address	Description
8400	INTERRUPT PAGE COMMAND PLA TEST 4 (CINTCMD) (37 microseconds) The Interrupt Page Command PLA test 4 verifies the operation of a portion of the interrupt page command PLA. The CPU microcode informs the interrupt page that the test will be run and then sends a sequence of MMP IIO commands to the interrupt page. The interrupt page verifies that the proper commands were received in the proper sequence. Any errors detected by the interrupt page during this test shall be saved in the interrupt page Scan Register which can be read using the EA Scan 5 Assist Diagnose. The condition code will always be zero.
8500	INTERRUPT PAGE COMMAND PLA TEST 5 (CINTCMD) (34 microseconds) The Interrupt Page Command PLA test 5 verifies the operation of a portion of the interrupt page command PLA. The CPU microcode informs the interrupt page that the test will be run and then sends a sequence of MMP IIO commands to the interrupt page. The interrupt page verifies that the proper commands were received in the proper sequence. Any errors detected by the interrupt page during this test shall be saved in the interrupt page Scan Register which can be read using the EA Scan 5 Assist Diagnose. The condition code will always be zero.
9000	INTERRUPT PAGE ARITHMETIC INTERRUPT TEST (CPUARITH) (23 microseconds) The Interrupt Page Arithmetic Interrupt test verifies that the Floating Point Overflow, Fixed Point Overflow, and Floating Point Underflow CPU interrupts set the proper bits in the interrupt page Arithmetic Group Capture Register. The CPU microcode informs the interrupt page that the test will be run and then forces each interrupt in the proper sequence. The interrupt page verifies that the proper interrupts were received in the proper sequence. Any errors detected by the interrupt page during this test shall be saved in the interrupt page Scan Register which can be read using the EA Scan 5 Assist Diagnose. The condition code will always be set zero.

Hexadecimal Contents of Effective Address	Description

9100 INTERRUPT PAGE H-BUS WRAP ASSIST (IHBUSWRP)

(11.6 microseconds)

The Interrupt Page H-BUS Wrap assist verifies that data can be sent to the interrupt page on the H-BUS and received from the interrupt page on the H-BUS and INBUS correctly. The data pattern in bits 0-15 of register R1 shall be sent to the interrupt page on the H-BUS. The data received from the interrupt page on the H-BUS shall be placed in bits 16-31 of register R1, while the data received on the INBUS shall be placed in bits 0-15 of register R1+1.

A000 EU ROS PARITY ERROR MICROCODE TEST (ROSPAR)

(25 microseconds)

The EU ROS Parity Error microcode test verifies that the EU ROS microcode parity check circuit functions correctly. The test verifies that a microword with incorrect parity generates an EU ROS parity error. At completion of the test, the condition code shall be set zero if no error occurred. The condition code shall be set negative if an error occurred.

B000 DSE RAM TEST (DSETEST)

(72 microseconds)

The DSE RAM test verifies that all DSE RAM locations can be addressed and that there is no stuck bits. The DSE RAM test first will write the address of each location into its location. The DSE RAM is read, verifying that each location can be addressed. Then the DSE RAM test will write the complement of its address into its location. The DSE RAM is read, verifying no stuck data bits. If the test passes, the condition code shall be set zero. Note: The DSE RAM is set to all zeros at the end of this diagnose instruction.

Hexadecimal Contents of Effective Address	Description
C000	MONOLITHIC CHECKSUM MICROCODE ASSIST (CKSUM) (20 microseconds plus 24 microseconds for each additional halfword) The monolithic checksum assist allows the macrocode to checksum all monolithic memory locations within a specified range. The 19-bit start address shall be placed right-justified in the fullword register R1. This register shall be incremented as each halfword is checksummed. The 19-bit end address shall be placed right-justified in the fullword register R1+1. This is the address of the last word to checksum. This register will not be altered. At completion, the start address will equal the end address. The checksum shall be in bits 0-15 of register R1+2. This register should be zeroed by macrocode before calling the Diagnose to accumulate a new checksum. This register will contain the updated checksum of the halfword locations. This Diagnose is interruptible. Note: This test will delay any pending I/O operation more than 3.05 microseconds and so should not be executed in an operational environment which does not control I/O activity.

Hexadecimal Contents of Effective Address	Description

C20X MONOLITHIC MEMORY READ/WRITE TEST (MEMRWT)

(49 microseconds)

The Monolithic memory Read/Write test shall perform a read/write test on one fullword of monolithic memory. It may be called repeatedly to test all memory. This test shall test any location, regardless of store protect status. It shall test the data bits, check bits, and store protect bits. At the end of the test, the location shall be returned to its original value and protect status. Hardware errors may cause machine check interrupts for bad memory or store protect unalike. The X field of the Diagnose Command word shall be 0, 1, or 2 to select one of the three test patterns to use in the test.

Pattern (Both Halfwords)	Data	Check Bits (binary)	Store Protect
0	C3B2	010010	111
1	3C4D	010010	000
2	944C	101101	111

Register R1 shall contain the 19-bit physical address right-justified of the location to test. This must be an even fullword address. At completion of the test, the condition code shall be set to zero if the test passed, and negative if the test failed. If an error occurs, the Diagnose log out area shall contain the following data in the CPU Sector Zero local store area:

R0/R1 - Address Tested
R2 - Diagnose Command Word (X'C20X')
R3 - Error Code
X'90' - Data Bits
X'91' - Check/Store Protect Bits
R4/R5 - XOR of actual and expected data
(1 - bit in error)
R6/R7 - XOR of actual and expected Check and Store Protect bits.

Note: This test will delay any pending I/O operation more than 35 microseconds and so should not be executed in an operational environment which does not control I/O activity.

Hexadecimal Contents of Effective Address	Description

D000 EDAC SOFT ERROR TEST (DIAGEDAC)

(37 microseconds)

The EDAC Soft Error Test checks the Error Detection And Correction (EDAC) logic used to detect and correct all single bit errors in monolithic memory. Bits 0-15 of register R1 shall contain the check bit pattern to be tested and bits 16-31 of register R1 shall contain the data bit pattern to be tested. Bits 0-15 of register R1+1 shall contain the expected data bit pattern for a soft error. At completion of this Diagnose, a condition code of zero indicates the test passed and a negative condition code indicates a failure. If an error occurs, the Diagnose log out area shall contain the following data in the CPU Sector Zero local store area:

R0 - Data read back from halfword location
R1 - Not used
R2 - Diagnose Command Word (X'D000')
R3 - Error Code
  X'9A' - Soft Error expected, Hard Error bit on
  X'9B' - Soft Error expected, Soft Error bit off
  X'9C' - Soft Error expected, Data not correct
R4 - MMU Fault Extension Register (MFER) at end of test
R5 - Expected Data for Soft Errors (R1+1 bits 0-15)
R6/R7 - Data Patterns used for test (R1)

The memory address used in this test is location 0 which must not be store protected. Note: This test will delay any pending I/O operation more than 6.1 microseconds and so should not be executed in an operational environment which does not control I/O activity.

Hexadecimal Contents of Effective Address	Description

D001 EDAC HARD ERROR TEST (DIAGEDAC)

(37 microseconds)

The EDAC Hard Error Test checks the Error Detection And Correction (EDAC) logic used to detect all double bit errors and many multiple bit errors in monolithic memory. Bits 0-15 of register R1 shall contain the check bit pattern to be tested and bits 16-31 of register R1 shall contain the data bit pattern to be tested. At completion of this Diagnose, a condition code of zero indicates the test passed and a negative condition code indicates a failure. If an error occurs, the Diagnose log out area shall contain the following data in the CPU Sector Zero local store area:

R0 - Data read back from halfword location
R1 - Not used
R2 - Diagnose Command Word (X'D001')
R3 - Error Code
X'98' - Hard Error expected, Hard Error bit off
X'99' - Hard Error expected, Soft Error bit on
R4 - MMU Fault Extension Register (MFER) at end of test
R6/R7 - Data Patterns used for test (R1)

The memory address used in this test is location 0 which must not be store protected. Note: This test will delay any pending I/O operation more than 6.1 microseconds and so should not be used in an operational environment which does not control I/O activity.

Hexadecimal Contents of Effective Address	Description

D010 ADDRESSABLE EDAC SOFT ERROR TEST (DIAGEDAC)

(37 microseconds)

The Addressable EDAC Soft Error Test checks the Error Detection And Correction (EDAC) logic used to detect and correct all single bit errors in monolithic memory. It allows the user to specify the address of the location to be used for the EDAC test. This facilitates the testing of multiple monolithic memory pages. Bits 0-15 of register R1 shall contain the check bit pattern to be tested and bits 16-31 of register R1 shall contain the data bit pattern to be tested. Bits 0-15 of register R1+1 shall contain the expected data bit pattern for a soft error. The 19-bit physical address, right-justified, of the location to test, shall be contained in bits 16-31 of register R1+1 and bits 0-15 of register R1+2. At completion of this Diagnose, a condition code of zero indicates the test passed and a negative condition code indicates a failure. If an error occurs, the Diagnose log out area shall contain the following data in the CPU Sector Zero local store area:

R0 - Data read back from halfword location
R1 - Not used
R2 - Diagnose Command Word (X'D010')
R3 - Error Code
  X'9A' - Soft Error expected, Hard Error bit on
  X'9B' - Soft Error expected, Soft Error bit off
  X'9C' - Soft Error expected, Data not correct
R4 - MMU Fault Extension Register (MFER) at end of test
R5 - Expected Data for Soft Errors (R1+1 bits 0-15)
R6/R7 - Data Patterns used for test (R1)

The memory address used in this test must not be store protected. Note: This test will delay any pending I/O operation more than 6.1 microseconds and so should not be executed in an operational environment which does not control I/O activity.

Hexadecimal Contents of Effective Address	Description

D011 ADDRESSABLE EDAC HARD ERROR TEST (DIAGEDAC)

(37 microseconds)

The Addressable EDAC Hard Error Test checks the Error Detection And Correction (EDAC) logic used to detect all double bit errors and many multiple bit errors in monolithic memory. It allows the user to specify the address of the location to be used for the EDAC test. This facilitates the testing of multiple monolithic memory pages. Bits 0-15 of register R1 shall contain the check bit pattern to be tested and bits 16-31 of register R1 shall contain the data bit pattern to be tested. The 19-bit physical address, right-justified, of the location to test shall be contained in bits 16-31 of register R1+1 and bits 0-15 of register R1+2. At completion of this Diagnose, a condition code of zero indicates the test passed and a negative condition code indicates a failure. If an error occurs, the Diagnose log out area shall contain the following data in the CPU Sector Zero local store area:

R0 - Data read back from halfword location
R1 - Not used
R2 - Diagnose Command Word (X'D011')
R3 - Error Code
  X'98' - Hard Error expected,
    Hard Error bit off
  X'99' - Hard Error expected,
    Soft Error bit on
R4 - MMU Fault Extension Register
  (MFER) at end of test
R6/R7 - Data Patterns used for test (R1)

The memory address used in this test must not be store protected. Note: This test will delay any pending I/O operation more than 6.1 microseconds and so should not be used in an operational environment which does not control I/O activity.

Hexadecimal Contents of Effective Address	Description

D100 READ MONOLITHIC STORE PROTECT BITS (READSP)

(16.8 microseconds)

The Read Monolithic Store Protect Bits assist reads the store protect bits from two monolithic memory halfwords and places them in a general register. Register R1 shall contain the 19-bit physical address right-justified of the memory location from which the store protect bits are to be read. This must be an even fullword boundary. Register R1+1 shall contain the store protect bits read from the two halfwords. The definition of the bits in R1+1 is described below:

Bits 0-12	Undefined
Bits 13-15	Redundant Store Protect Bits for address in R1 (even HW)
Bits 16-21	Undefined
Bits 22-24	Redundant Store Protect Bits for address in R1 plus one (odd HW)
Bits 25-31	Undefined

The macrocode can use this Diagnose to determine which monolithic memory locations are store protected and only checksum those locations. Note: The store protect bits read are inverted. Zero signifies a protected location. One signifies an unprotected location. Note: This test will delay any pending I/O operation more than 3.8 microseconds and so should not be executed in an operational environment which does not control I/O activity.

Hexadecimal Contents of Effective Address	Description

D200 SET ECC BITS ASSIST (SETEDAC)

(13 microseconds)

The Set ECC Bits assist allows the macrocode to force data with incorrect (or correct) ECC bits in any monolithic memory location which is not store protected. Fullword register R1 shall contain the 19-bit address right-justified of the halfword location in monolithic memory in which ECC is to be changed. Register R1+1 shall contain the following:

Register R1+1 bits	Contents
0	Check bit X
1	Check bit 0
2	Check bit 1
3	Check bit 2
4	Check bit 4
5	Check bit 8
6	0
7	0
8	0
9	0
10	0
11	1
12	0
13	1
14	0
15	0
16-31	16 bit data pattern to be stored

Check bit X is the XOR of data pattern bits 1, 2, 3, 5, 8, 9, 11, and 14.
Check bit 0 is the XOR of data pattern bits 0, 1, 2, 4, 6, 8, 10, and 12.
Check bit 1 is the XNOR of data pattern bits 0, 3, 4, 7, 9, 10, 13, and 15.
Check bit 2 is the XNOR of data pattern bits 0, 1, 5, 6, 7, 11, 12, and 13.
Check bit 4 is the XOR of data pattern bits 2, 3, 4, 5, 6, 7, 14, and 15.
Check bit 8 is the XOR of data pattern bits 8, 9, 10, 11, 12, 13, 14, and 15.

Hexadecimal Contents of Effective Address	Description
D200	(cont) Note: For multibit errors, the corrected data produced by the EDAC is unspecified. This assist cannot be used to restore monolithic memory locations with incorrect ECC to correct ECC. Use the Reset ECC Bits assist to restore correct ECC to a monolithic memory location. This assist also disables scrubbing when executed. To enable scrubbing, use the H-BUS Write command with the correct IIO command and data. Note: This test will delay any pending I/O operation more than 13 microseconds and so should not be executed in an operational environment which does not control I/O activity. Note: To increase performance, this Diagnose does not purge the pipe. So any prefetched data will not be affected by this Diagnose. To detect errors on data which has been prefetched, execute a branch after this Diagnose so that the data is fetched again.
D300	RESET ECC BITS ASSIST (SETEDAC) (8 microseconds) The Reset ECC Bits assist allows the macrocode to force correct ECC bits in any monolithic memory location which is not store protected. Fullword register R1 shall contain the 19-bit address left-justified of the halfword location in monolithic memory in which ECC is to be restored. Bits 16-31 of register R1+1 shall contain the data to be written into the monolithic memory location. This assist also disables scrubbing when executed. To enable scrubbing, use the H-BUS Write command with the correct IIO command and data. The correct ECC bits shall be generated when the data is written if the MMU Mode register does not have IDO active. Note: This test will delay any pending I/O operation more than 8 microseconds and so should not be executed in an operational environment which does not control I/O activity.

EA SCAN ASSISTS

The register set used with all EA SCAN ASSISTS contains the following values in the general registers. These values will be used by the EA in computing effective addresses.

R0 - X'01060000'	R4 - X'00040040'
R1 - X'FEF90000'	R5 - X'00050050'
R2 - X'01060000'	R6 - X'00060060'
R3 - X'01060000'	R7 - X'00010000'

Only one instruction which reads store protect bits (store type) may be included in an EA SCAN buffer. The store pending hardware will be set when this instruction is decoded. But, since the instruction is not executed, only decoded, a second instruction of this type will cause the EA to stop until the first instruction has been executed. This will cause Diagnose to hang and result in an Endop Timeout Machine Check Interrupt. The Instruction Address in the Old PSW for a Machine Check Interrupt which occurs during an EA SCAN ASSIST may not be correct. The instructions placed in an EA SCAN buffer must be selected and arranged with care. Any register conflicts, operand conflicts, or pending stores will alter the results produced by the EA SCAN ASSISTS.

Hexadecimal Contents of Effective Address	Description
E000	EA SCAN 1 ASSIST (EASCAN) (21 microseconds) The EA Scan 1 Assist allows the IU and EA to be stepped and the selected scan register to be read. The IU and EA can each be stepped from 0-15 times before scanning. Bits 0-15 of register R1 contains the virtual address of the buffer of instructions to be used by the assist. Register R1+1 contains the scan command. The bits of the scan command are defined below.

Bits	Contents
0-3	IU Step Count
4-7	EA Step Count
8-15	Interrupt Page Scan Data
16-31	Interrupt Page Scan Command

The results of this scan assist can be read by EA Scan 5 Assist. The execution time of this Diagnose is instruction buffer dependent.

Hexadecimal Contents of Effective Address	Description
E100	EA SCAN 2 ASSIST (EASCAN2)

(22 microseconds)

The EA Scan 2 Assist allows the IU to run and the EA to be stepped. For each EA step which produces a valid decoded opcode, the decoded opcode, the halfword operand, the fullword operand, the left local store, and the right local store are checksummed. Bits 0-15 of register R1 contain the count of the number of valid decoded opcodes to step. Bits 16-31 of register R1 contain the virtual address of the buffer of instructions to be used by the assist. The checksums produced by this assist are placed in the following CPU sector 7 registers:

Register	Checksum
0,1	Fullword Operand
8	Decoded Opcode
9	Halfword Operand
10	Left Local Store
11	Right Local Store

These checksums can be read using the Local Store Read Diagnose. The execution time of this Diagnose is instruction buffer dependent.

Hexadecimal Contents of Effective Address	Description

E200 EA SCAN 3 ASSIST (EASCAN)

(45 microseconds)

The EA Scan 3 Assist allows an IU or EA scan register to be read. Then the IU and EA are stepped and the selected scan register is read. The IU and EA can each be stepped from 0-15 times. The results of the first scan are placed in local store CPU sector 7 registers 14 and 15. This data can be read using the Local Store Read Diagnose. Bits 0-15 of register R1 contain the virtual address of the buffer of instructions to be used by the assist. Register R1+1 contains the second scan command. The bits of the scan command are defined below:

Bits	Contents
0-3	IU Step Count
4-7	EA Step Count
8-15	Interrupt Page Scan Data
16-31	Interrupt Page Scan Command

Bits 0-15 of register R1+2 contain bits 0-15 of the first scan command. However, only bits 8-15 are used. Bits 16-31 of the first scan command are the same as for the second scan command and are contained in bits 16-31 of register R1+1. The results of the second scan can be read by EA Scan 5 Assist. The execution time of this Diagnose is instruction buffer dependent.

E300 EA SCAN 4 ASSIST (EASCAN2)

(21 microseconds)

The EA Scan 4 Assist allows the IU to run and the EA to be stepped. For each EA step which produces a valid decoded opcode, the fullword operand is checksummed. Bits 0-15 of register R1 contain the count of the number of instructions to step. Bits 16-31 of register R1 contain the virtual address of the buffer of instructions to be used by the assist. The fullword operand checksum is placed in local store CPU sector 7 registers 8 and 9. The checksum can be read using the Local Store Read Diagnose. The execution time of this Diagnose is instruction buffer dependent.

Hexadecimal Contents of Effective Address	Description

E301 EA SCAN 5 ASSIST (CPUSREAD)

(18 microseconds)

The EA Scan 5 Assist allows the 32 bits of EA scan data to be read from the Interrupt page. Register R1 will contain the 32 bits of scan data at completion. The Interrupt page scan register is used for scan data and also as the Interrupt page Diagnose Error register. This register is cleared after reading. It should be read before executing any Interrupt page self-test commands to clear the register and after executing the self-test to see if any errors were detected. The bits in the Interrupt Page Diagnose Error Register are defined below:

Bit	Description
0	Diagnose Register Fault
1	GP Register Fault
2	Unconditional Branch Test Fault
3	ALU Function Test Fault
4	Local Store Data Test Fault
5	Local Store Addressing Test Fault
6	Conditional Branch Test Fault
7	POR Fault
8	Capture Register - High Group Fault
9	Capture Register - CPU Group Fault
10	Capture Register - Memory Group Fault
11	Capture Register - I/O Group Fault
12	Capture Register - External Group Fault
13	0 (spare)
14	0 (spare)
15	Bad Parity Test Fault
16	Priority PLA Fault (Input = X'FF' Fails)
17	Priority PLA Fault (Input = X'7F' Fails)
18	Priority PLA Fault (Input = X'3F' Fails)
19	Priority PLA Fault (Input = X'1F' Fails)
20	Priority PLA Fault (Input = X'0F' Fails)
21	Priority PLA Fault (Input = X'07' Fails)
22	Priority PLA Fault (Input = X'03' Fails)
23	Priority PLA Fault (Input = X'01' Fails)
24	Command PLA Fault
25	0 (spare)
26	Priority PLA Test 2 Fault
27	Priority PLA Test 3 Fault
28	Floating Point Overflow Fault
29	Fixed Point Overflow Fault
30	0 (spare)
31	Floating Point Underflow Fault

Hexadecimal Contents of Effective Address	Description

F100 ENDOP TIMER TEST (ENDOPINT)

(300 microseconds)

The ENDOP Timer test verifies the proper operation of the ENDOP timer. The ENDOP timer is reset and microcode waits to see if the correct micro interrupt is generated when the timer times out. If the timer does not time out in the proper time (200us) or the correct micro interrupt is not generated, the condition code shall be set negative. If the test passes the condition code shall be set zero. Note: This test will delay any pending interrupts more than 300 microseconds and so should not be executed in an operational environment which can not tolerate this delay. Also, this Diagnose Instruction cannot be macrostepped.

F200 WAIT MICROCODE ASSIST (WAITMICR)

The Wait microcode assist causes all CPU memory operations to stop until any macro or micro level interrupt occurs. Note: This Diagnose Instruction cannot be macrostepped.

F300 FORCE ROS PARITY ERROR ASSIST (FORCEROS)

(3.5 microseconds)

The Force ROS Parity Error Assist allows macrocode to force a ROS parity error. The ROS parity error will only be forced if bits 0-15 of register R1 contain X'0001'. The condition code will be set zero. The ROS parity error machine check interrupt can be masked by PSW bit 45. If it is masked, it will not remain pending and the computer will not be reset. The reset of the computer on detection of the ROS parity error can be inhibited by placing the interrupt page in Diagnose Mode. In Diagnose Mode the machine check interrupt will not be generated.

The following descriptions define the hexadecimal value for the H-BUS IIO command required to select each of the micro sequences. Only the diagnostic IIO commands are described here. These commands are used by the H-BUS Read and Write commands.

H-BUS IIO Command — Description

1002 READ INTERRUPT PAGE LOCAL STORE REGISTER (READ)

(12.8 microseconds)

The read interrupt page local store register IIO command allows macrocode to read the interrupt page local store registers. Each register is eight bits in length. The command reads two consecutive registers. The table below describes the data to be sent with the command and the registers which are read and placed in the general register.

H-BUS Data	Register Read bits 0-7	bits 8-15
0000	R0	R1
0001	R1	R2
0002	R2	R3
0003	R3	R4
0004	R4	R5
0005	R5	R6
0006	R6	R7
0007	R7	R8
0008	R8	R9
0009	R9	R10
000A	R10	R11
000B	R11	R12
000C	R12	R13
000D	R13	R14
000E	R14	R15
000F	R15	R0

9011 INTERRUPT PAGE SELF-TEST (WRITE)

The interrupt page self-test IIO command allows macrocode to perform various self-test microcode functions on the interrupt page. The results of the self-test are placed in the scan register. This register can be read by the macrocode via the EA SCAN 5 ASSIST Diagnose. The register should be read to clear it before executing the self-test. The table below describes the data to be sent with the command and the function performed. The self-test is divided into many small tests to meet the 50 microsecond interruptibility requirement.

H-BUS Data	Description
0000	This subcommand verifies unconditional branching, ALU functions and local store register 15 data integrity. (34 microseconds)
0001	This subcommand verifies local store register 0 and 1 data integrity. (38 microseconds)
0002	This subcommand verifies local store register 2 and 3 data integrity. (38 microseconds)
0003	This subcommand verifies local store register 4 and 5 data integrity. (38 microseconds)
0004	This subcommand verifies local store register 6 and 7 data integrity. (38 microseconds)
0005	This subcommand verifies local store register 8 and 9 data integrity. (38 microseconds)
0006	This subcommand verifies local store register 0 addressing. (24 microseconds)
0007	This subcommand verifies local store register 1 addressing. (22 microseconds)
0008	This subcommand verifies local store register 2 addressing. (39 microseconds)
0009	This subcommand verifies local store register 3 addressing. (36 microseconds)
000A	This subcommand verifies local store register 4 addressing. (33 microseconds)
000B	This subcommand verifies local store register 5 and 6 addressing. (41 microseconds)

H-BUS Data	Description
000C	This subcommand verifies local store register 7 and 8 addressing. (33 microseconds)
000D	This subcommand verifies local store register 9, 10, and 11 addressing. (36 microseconds)
000E	This subcommand verifies local store register 12 and 13 addressing. (16.2 microseconds)
000F	This subcommand verifies the conditional branching function. (32 microseconds)
8000	This subcommand verifies the parity circuits on the command PLA, the priority PLA, and the control store. This subcommand also verifies the operation of a portion of the priority PLA. (27 microseconds)
8100	This subcommand verifies the operation of the capture register. Any pending interrupts will be lost when this subcommand is executed. Note: Since any pending interrupts are lost, it may not be possible to macrostop when executing this Diagnose. (335 microseconds)
8200	This subcommand verifies the operation of a portion of the priority PLA. (29 microseconds)
8C00	This subcommand verifies local store register 10 and 11 data integrity. (38 microseconds)
8D00	This subcommand verifies local store register 12 and 13 data integrity. (38 microseconds)
8E00	This subcommand verifies local store register 14 data integrity. (23 microseconds)

H-BUS Data	Description
8F00	This subcommand verifies the operation of a portion of the priority PLA. (28 microseconds)

H-BUS IIO Command	Description

9013 SET/RESET INTERRUPT PAGE DIAGNOSE MODE (WRITE)

(6.6 microseconds)

The set/reset interrupt page Diagnose mode IIO command allows macrocode to place the interrupt page in, or remove it from diagnose mode. When in diagnose mode, the interrupt page will not reset the computer when it detects a crash interrupt condition. Also, the ROS parity error, and the Endop Timeout machine check interrupts will not be generated. If the data sent with the command is nonzero, the page will be placed in diagnose mode. If the data sent with the command is zero, the page will be removed from diagnose mode.

9014 START INTERRUPT PRIORITY MICROCODE TEST (WRITE)

The Start Interrupt Priority Test IIO command sets all of the valid interrupts in the External Pending Interrupt Register. Also, the two interval timers are set pending. Interrupt processing will then proceed in the normal manner. Any pending interrupts will be lost when this command is executed.

The following interrupts are set in the External Pending Interrupt Register:

External 0
External 1
External 2
External 3
External 4
AGE

The following interrupts are set in I/O Interrupt Register:

Timer A
Timer B

Note: Timer A and B interrupts only become macro interrupts if location B0 and B1, respectively, equal zero.

H-BUS IIO Command	Description

9407 LOAD MMU MODE REGISTER (WRITE)

(4.3 microseconds)

The Load MMU Mode Register IIO command allows the operation of the MMU and the various memory options to be selected. The definition of the bits in the MMU Mode Register is presented below:

Bit	Description
6	Inhibit memory accesses from all sources except EX
7	Inhibit all memory error interrupts
8	BSE Disable
9	Insert store protect bits
10	Transmit Disable
11	System IPL
12	Passthrough mode
13	Gate syndrome/check bits to data bus
14	Code ID0
15	Disable scrubbing

Bits 6-15 of the data word sent with this command will be loaded into the MMU Mode register.

ERROR CODE	DESCRIPTION	CLD SHEET		TEST NAME
01	RTCC Register 5's Pattern Fail	UD34		RTCC
02	RTCC Register A's Pattern Fail	UD34		RTCC
03	Constant Prom Checksum Fail	UD25		CPROM
04	2way Branch On YBUS Bit 11 Fail	UD51		XWAYBR
05	2way Branch On YBUS Bit 12 Fail	UD51		XWAYBR
06	2way Branch On YBUS Bit 13 Fail	UD52		XWAYBR
07	2way Branch On YBUS Bit 14 Fail	UD52		XWAYBR
08	2way Branch On YBUS Bit 15 Fail	UD52		XWAYBR
09	4way Branch On YBUS Bits 12,13	UD53		XWAYBR
0A	4way Branch On YBUS Bits 14,15	UD53		XWAYBR
0B	16way Branch On YBUS Bits 8-11	UD54		XWAYBR
0C	16way Branch On YBUS Bits 12-15	UD54		XWAYBR
0D	Local Store Adr Forced CPUSEC0	UD95		LSADR
0E	Local Store Adr Forced CONSEC0	UD96		LSADR
0F	Local Store Adr Direct CONSEC15	UD96		LSADR
10	Local Store Adr Indirct CONSEC15	UD96		LSADR
11	Ls Adr CPUSEC Direct ≠ Forced	UD97		LSADR
12	Ls Adr CONSEC Direct ≠ Forced	UD98		LSADR
13	Even Adr FW Write HW Read	UE55		HCMEMTST
14	Even Adr FW Write Odd HW Read	UE55		HCMEMTST
15	Two HW Writes FW Read Even Adr	UE56		HCMEMTST
19	FW Write Read Odd Adr	UE56		HCMEMTST
1A	Two HW Writes FW Read Odd Adr	UE60		HCMEMTST
1B	2way Branch On ASTAT3 Fail	UE18		STATBR
1C	2way Branch On ASTAT2 Fail	UE18		STATBR
1D	2way Branch On ASTAT1 Fail	UE19		STATBR
1E	2way Branch On ASTAT0 Fail	UE19		STATBR
1F	4way Branch On ASTAT45 Fail	UE21		STATBR
20	4way Branch On ASTAT67 Fail	UE21		STATBR
21	Logical/Physical Adr Error	UE61		HCMEMTST
22	Store Protect Bits Incorrect	UE62		HCMEMTST
30	EI ≠ EPEMIT Or EXZERO Fail	UE27		EXHARD
31	EI=EI+1 Or EXZERO Fail	UE27		EXHARD
32	EI=EI+EPEMIT Or EXSIGN Fail	UE27		EXHARD
33	EXCRY Fail	UE27		EXHARD
34	EXSIGN Fail	UE28		EXHARD
35	EXCRY Fail	UE28		EXHARD
36	EA=EI,EB=EI,ES=EA+EB Or Reg Fail	UE28		EXHARD
37	EA=EPEMIT Or EA=EA+EPEMIT Fail	UE29		EXHARD
38	EA carry Or ES=EI&EPEMIT Fail	UE29		EXHARD
39	EA Effects EI	UE29		EXHARD

ERROR CODE	DESCRIPTION	CLD SHEET		TEST NAME
3A	EA Effects EB	UE30		EXHARD
3B	EB=EB-1 Fail	UE30		EXHARD
3C	EB=F00-07 Fail	UE30		EXHARD
3D	EB Effects EA	UE31		EXHARD
3F	EA=F00-07 Fail	UE31		EXHARD
40	EI=EI&EPEMIT Fail	UE32		EXHARD
41	EA=EA-EB Fail	UE32		EXHARD
42	EA=F24-31;INV Fail	UE32		EXHARD
43	EB=f24-31;INV Fail	UE33		EXHARD
44	EI=EA-EB Fail	UE33		EXHARD
45	EA=EA(L1) Fail	UE33		EXHARD
46	EI=EI+2 Fail	UE33		EXHARD
47	EI=EI-2 Fail	UE34		EXHARD
48	EI=EA Fail	UE34		EXHARD
49	EI=EPEMIT Of EB=F00-07;EI=EPEMIT	UE34		EXHARD
4A	EB=F00-07 Of EB=F00-07;EI=EPEMIT	UE34		EXHARD
4B	EI=EI-1 Fail	UE28		EXHARD
4C	EI=INB0815 Fail	UE34		EXHARD
4D	WLS=Y(FW, EXALU, MMP/1750) Fail	UE34		EXHARD
50	FA ≠ 0 Or FZERO Fail	UE35		FRXHARD
51	FB ≠ 0	UE35		FRXHARD
52	FC ≠ 0	UE35		FRXHARD
53	FA ≠ F,LS Addr, AI+BI Or FCRY	UE36		FRXHARD
54	FA Effects FB	UE36		FRXHARD
55	FA Effects FC	UE36		FRXHARD
56	FB ≠ F	UE36		FRXHARD
57	FB Effects FA Or FC	UE36		FRXHARD
58	FSIGN1 Fail	UE37		FRXHARD
59	FC Effects FA Or FB	UE37		FRXHARD
5A	YOV Fail	UE37		FRXHARD
5B	FZERO With Overflow Fail	UE37		FRXHARD
5C	AI+AI Or Guard Bit Fail	UE37		FRXHARD
5D	Shift Right 2 Or FNORM Fail	UE37		FRXHARD
5E	FANORM Fail	UE38		FRXHARD
5F	FA ≠ F HW Fail	UE38		FRXHARD
60	Bits 16-31 ≠ 0 HW	UE38		FRXHARD
61	Y MUX CONTROL Or ENABLE Fail	UE38		FRXHARD
62	Y=(N,N,N,N,I,I) Fail	UE39		FRXHARD
63	FC Shift/Rotate Fail	UE39		FRXHARD
64	FA31=FC00 Of FA31=FC00;FC31=FC32	UE39		FRXHARD
65	FC31=FC32 Of FA31=FC00;FC31=FC32	UE39		FRXHARD

ERROR CODE	DESCRIPTION	CLD SHEET		TEST NAME
66	FA00=FA-1 Of FA00=FA-1;FC00=FA31	UE40		FRXHARD
67	FC00=FA31 Of FA00=FA-1;FC00=FA31	UE40		FRXHARD
68	FA=Y Of FA=Y;FC=FCI Fail	UE40		FRXHARD
69	FC=FCI Of FA=Y;FC=FCI Fail	UE40		FRXHARD
6A	RDEXPALU Or BI=INBUS Fail	UE41		FRXHARD
6B	FIO, FL8, FB=FB(R1) Fail	UE41		FRXHARD
6C	FA=(Z,Z); FC=FCI Fail	UE41		FRXHARD
6D	FCI=FC(R2) Fail	UE41		FRXHARD
70	EDAC Error During Reset	UD71		EDACTEST
80	Local Store R/W LLS Pat 1	UD46,48		LSMICRO
81	Local Store R/W LLS≠RLS Pat 1	UD46,48		LSMICRO
82	Local Store R/W RLS Pat 2	UD46,48		LSMICRO
83	Local Store R/W RLS≠LLS Pat 2	UD47,49		LSMICRO
84	EU PC Fail On X'5555' Pattern	UD40		EUPCTST
85	EU PC Fail On x'AAAA' Pattern	UD40		EUPCTST
86	EU PC Increment Fail	UD40		EUPCTST
90	Memory R/W Data Pattern 1	UD65		WRCCORE
91	Memory R/W Check/SP Pattern 1	UD65		WRCCORE
98	EDAC Hard Error Bit Not On	UD74		EDACTEST
99	EDAC Unexpected Soft Error	UD74		EDACTEST
9A	EDAC Unexpected Hard Error	UD74		EDACTEST
9B	EDAC Soft Error Bit Not On	UD75		EDACTEST
9C	EDAC Soft Error Not Corrected	UD75		EDACTEST
A0	Memory Protect RAM, Pattern 1	UE09		DIAGMPPR
A1	Memory Protect RAM, Pattern 2	UE09		DIAGMPPR
A2	Inst-Page Reg RAM, Pattern 1	UE09		DIAGMPPR
A3	Inst-Page Reg RAM, Pattern 2	UE09		DIAGMPPR
A4	Opnd-Page Reg RAM, Pattern 1	UE09		DIAGMPPR
A5	Opnd-Page Reg RAM, Pattern 2	UE09		DIAGMPPR
A7	BSR/DSR - PS/AS Test Failed	UE23		DIAGBSRP
B0	Interval Timer A Test -SIB	UD94		ITMATST
F6	Interrupt Page Failure - FC reg is Int Page Diag Err Reg	UE49		HCSREAD
F7	ROS Parity Test Failure	UE03		INTCRASH
F8	ENDOP Timer Test Failure	UE03		INTCRASH
F9	DSE Test Failure	UE71		DSETEST
FB	Intr Page HBUS Wrap (INBUS) Fail	UD87		IHBUSWRP

ERROR CODE	DESCRIPTION	CLD SHEET		TEST NAME
FC	Intr Page HBUS Wrap (HBUS) Fail	UD86		IHBUSWRP
FD	Operand (FW) Checksum Fail	UD39		HEASCAN2
FE	LLS & RLS Checksum Fail	UD39		HEASCAN2
FF	Opcode & Operand Checksum Fail	UD39		HEASCAN2

16.0 PIPELINE TIMING CONSIDERATIONS

The AP-101S computer is a pipelined machine which exhibits significant throughput improvement over nonpipelined sequential machines. The pipeline which is involved is based on prefetching both instructions and operands from memory. Instructions and operands are prefetched assuming sequential instruction execution. This means that as long as the sequence of instruction execution is not altered, all prefetched information will be used.

Some branch instructions alter the sequence of execution, and therefore nullify any prefetched information. The time required to restart the pipeline in this case may be directly attributed to the branch instruction. Instruction execution times for branch instructions include all overhead required to restart the pipeline, if the order of execution is altered.

Other factors also exist which have an impact on the throughput of the pipeline. These factors may not be attributed directly to any one instruction in general, rather they are a function of the order and relationship of instruction execution. Three factors may be classified as follows:

Register conflict	Modification of base or index register needed to prefetch an operand
Store conflict	Modification of prefetched operand
I unit hazard	Modification of prefetched instruction

Instruction execution times do not include any overhead due to these factors. Any penalty in execution time must be considered independent of instruction execution time. The total time required to execute a given sequence of instructions must include any applicable penalty due to these factors.

It is for this reason that a separate description of conflicts and hazards is presented. Not only will this description explain the various conflicts and hazards as previously mentioned, it will also discuss how the conflicts and hazards are resolved and what the execution time impact is associated with these events. Furthermore, numerous conditions, such as branching and store instructions, will be discussed with an emphasis on pipeline operation. Instructions of this type change the nature of pipeline processing near that instruction, but are not a conflict or hazard. In order to aid understanding of the AP-101S computer and the pipeline, these instructions have been included in this discussion. Any execution time impacts due to the pipeline have already been included in the stated instruction execution times.

16.1 INSTRUCTION EXECUTION - PIPELINE BASICS

Every instruction requires at least four stages in order to execute. First, the instruction must be read from memory during the instruction fetch stage. Second, the instruction must be decoded both in terms of what type of operation is specified (add, multiply, shift, etc.) and the effective address of the second operand must be

computed. Next, the second operand is read from memory using the effective address (EA) during the operand fetch stage. Finally, the instruction may be executed, generally resulting in modification of the general purpose registers. In the case of the AP-101S computer, two additional stages are required in support of the memory references. Since the AP-101S utilizes expanded addressing, an additional stage of address translation is required for every memory operation. Therefore, an instruction address translation stage and operand address translation stage are required. Figure 16-1 shows the relationship between all six stages of the AP-101S computer.

Each stage represents a specific function which is relatively independent of the other functions, except for the given time relationship. It is this independence and the timing sequence which permits the construction of a six stage pipeline. Within the pipeline, each function, or stage, is contained and controlled completely by an independent hardware element. The timing relationship between an instruction and each hardware element is shown in Figure 16-2.

The advantage of using a pipelined organization is obvious when considering the execution of three simple instructions. Figure 16-3 indicates that a total of 18 machine cycles would be required for a sequential machine to execute just three instructions, assuming that each stage of the instruction could be completed in a single machine cycle. Each hardware element is capable of independent operation, which permits pipeline operation as shown in the figure. Notice that a total of 8 machine cycles are required to execute three instructions. Considering pipeline operation for a sequence of a single type of instruction yields the mean time required to execute that instruction. The example shown is for an RS format instruction. If the example were extended indefinitely, the execution time would average to 2 cycles per instruction. Completing a similar pipeline chart for SRS instructions would indicate 1.5 cycles per instruction, and 1 cycle for RR format instructions. For the AP-101S computer, the pipeline cycle time is 0.250 microseconds.

16.2 LONG INSTRUCTIONS - NON-SINGLE-CYCLE EXECUTION

Not all instructions may be executed by the execution unit within a single pipeline cycle. These instructions, referred to as long instructions, force the pipeline to stop while execution proceeds, as indicated in Figure 16-4. This is actually accomplished by postponing further EA calculations until the last machine cycle of the long instruction. Instruction execution times as indicated include any effects of long instructions, as necessary. Notice that even though the pipeline waits for a number of cycles, there are no unused cycles in the execution unit.

16.3 BRANCH INSTRUCTIONS - RESTART THE PIPELINE

Branch instructions, as previously discussed, cause any prefetched information to be discarded and the pipeline must be restarted. The branch instruction shown in Figure 16-5 indicates that 3 machine cycles within the execution unit are unused during the pipeline restart. Also, notice that the target instruction has

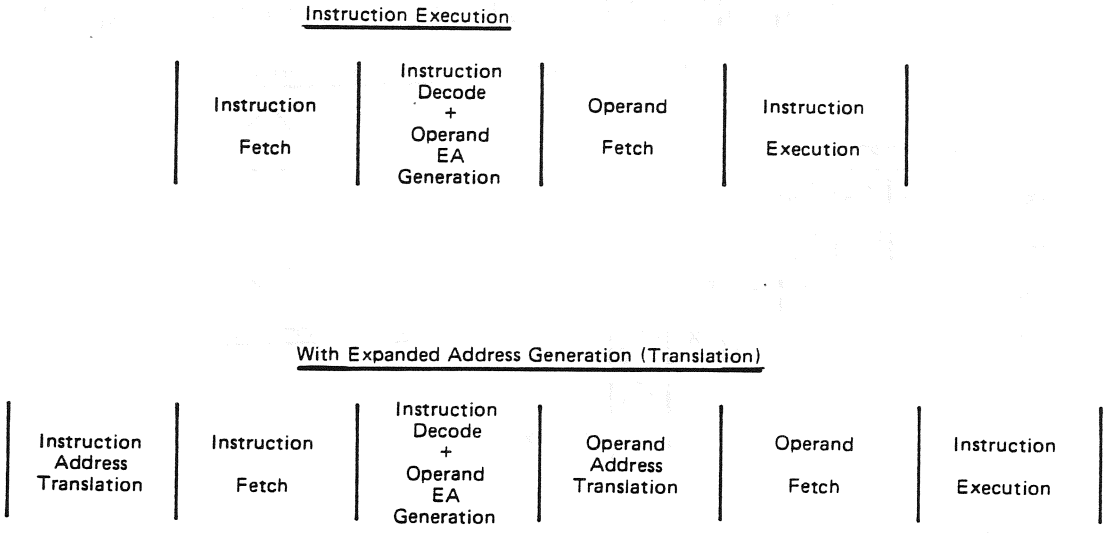


Figure 16-1. Dissection of Instruction

- o Sequence of 6 functions →6 stage pipeline
- o Independent hardware per stage/function

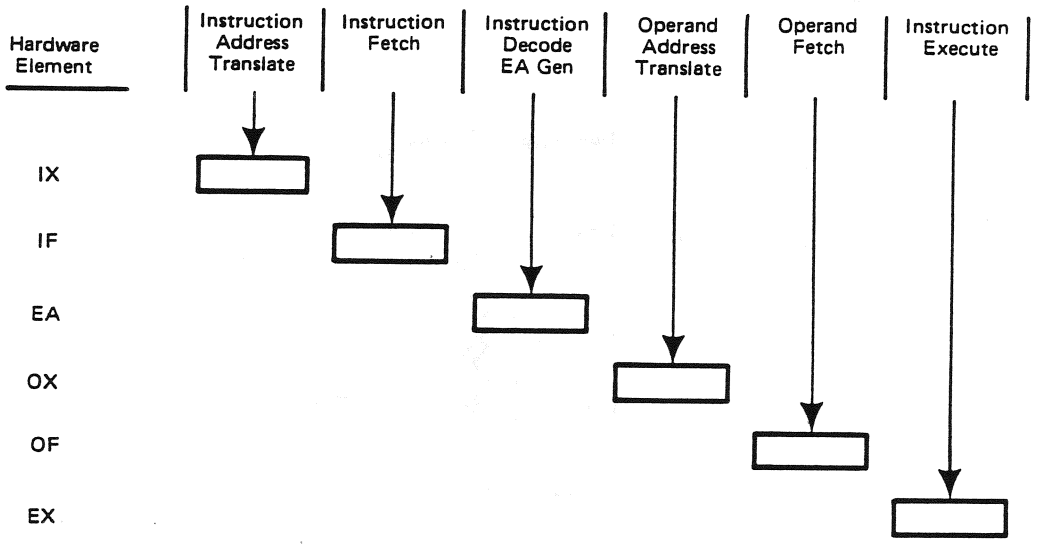
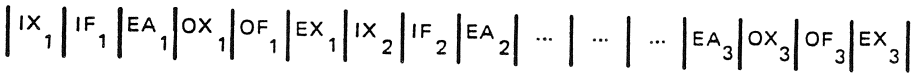


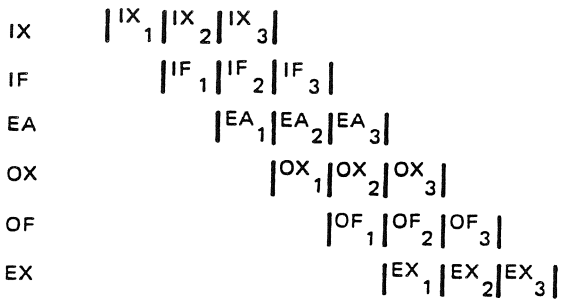
Figure 16-2. Pipeline Hardware Elements

- Consider the instruction sequence 1,2,3
- Sequential machine operation is:



6 cycles x 3 instructions = 18 cycles to complete 3 instructions

- Pipeline machine execution is:



8 cycles to complete 3 instructions

Therefore, over a period of time, pipelined instructions would average:

2 cycles / RS instruction
1.5 cycles / SRS instruction
1 cycle / RR instruction

Figure 16-3. Pipeline Advantage

- Not a hazard or conflict
- Instructions which require more than 1 pipeline cycle to execute
- Postpones EA calculations until end of instruction

LOC	INSTR	
L	AE	
L+2	--	(SHORT FLT PU ADD)
L+4	--	
L+6	--	

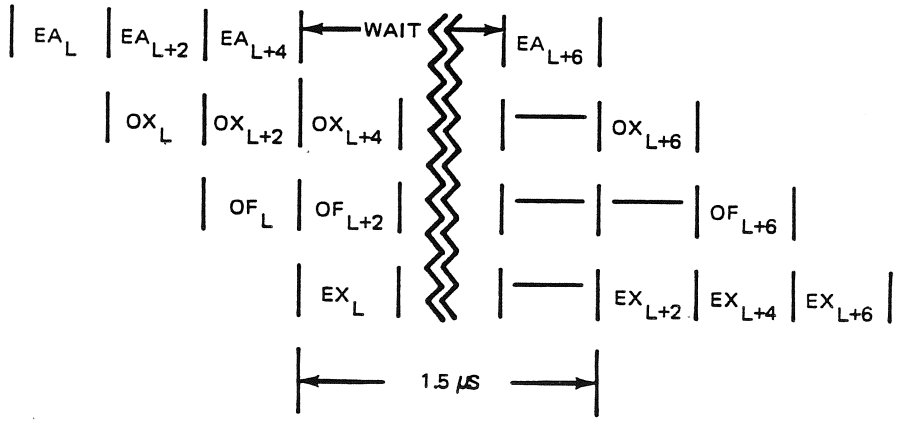


Figure 16-4. Long Instruction

- o Not a hazard or conflict
- o Harmful to pipeline throughput — 3 cycles to restart
- o Example:

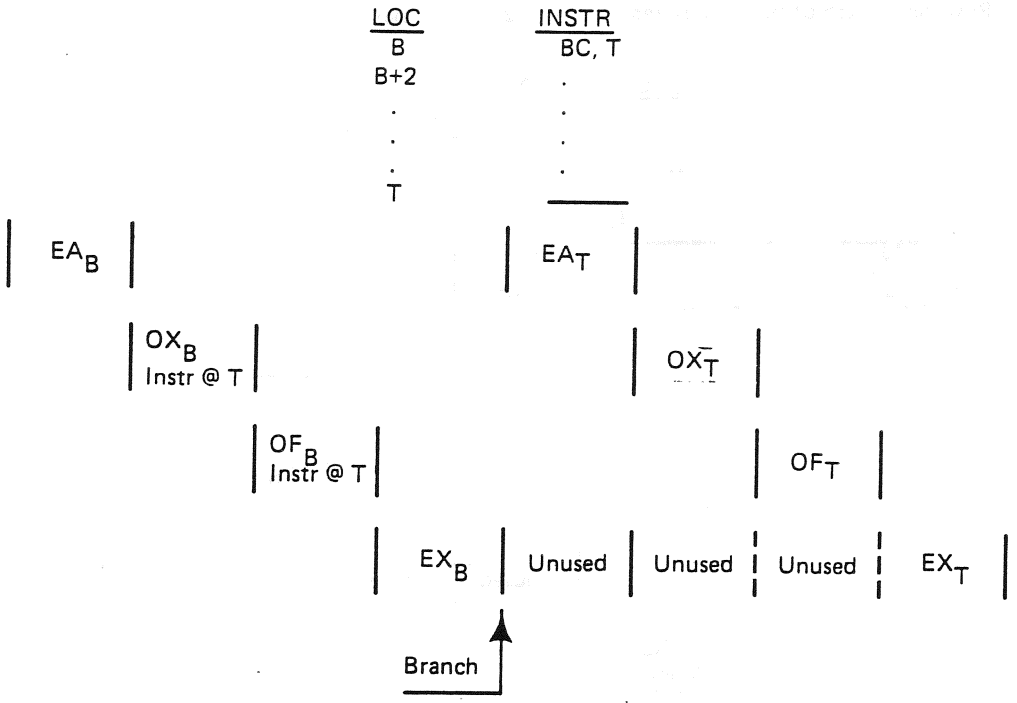


Figure 16-5. Branch Taken

previously been prefetched by the EA unit in order to minimize the restart time. If a conditional branch is not taken, then the pipeline is not restarted. Indicated instruction execution times include all effects of restarting the pipeline.

16.4 REGISTER CONFLICT - MODIFY BASE OR INDEX REGISTER

Register conflicts can only occur for instructions which use either a base or an index register to compute the effective address of a memory operand. A conflict arises if a preceding instruction (within three instructions) modifies the contents of the register which is used for the base or index value. In order to minimize the penalty involved, register conflicts are detected and totally controlled by hardware resources. EA unit operation is postponed, as shown in Figure 16-6, until the register involved has been loaded with the correct value. At most, three machine cycles will be unused by the EA unit while waiting for valid register data. This results in three unused machine cycles in the execution unit hardware. This penalty will decrease, depending upon the number of instructions between the register-modifying instruction and the register-using instruction. Any penalty involved with register conflicts has not been included with the stated instruction execution times, and must be evaluated separately if necessary.

- Caused by loading & using a base/index register within 3 instructions
- Detected and handled by hardware
- Forces sequential instruction execution within pipeline
- Postpones fetch of base/index register by 1, 2, or 3 cycles
- Example:

```
LOC     INSTR
 R       AHR R3, R5
R+1       M  R1, ADDR(R3)
R+3      ---
```

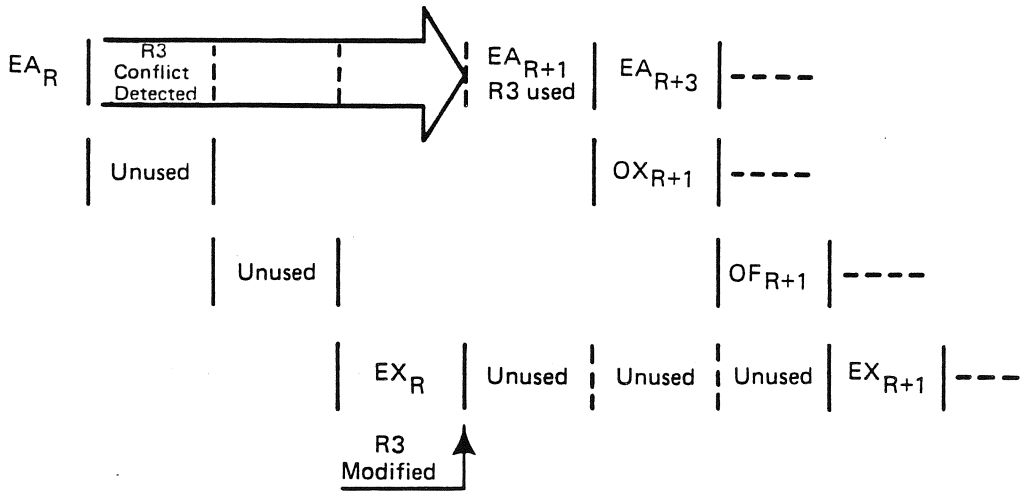


Figure 16-6. Register Conflict

16.5 STORE INSTRUCTIONS - MULTIPLE MEMORY CYCLES

The pipeline structure has been implemented to maximize performance for memory read operations. Memory write operations do not fit into the same pipeline structure as read operations and, as a result, the pipeline is disturbed in the area of a store instruction. Figure 16-7 indicates that two additional memory cycles are needed to perform the actual memory write operation. Also notice that the EA unit performs a pre-read of the memory location in order to assist the memory management unit in storage protection error detection. At most, two cycles will be unavailable for instruction execution due to this pipeline disturbance. The actual number of cycles lost is dependent upon the nature of the instruction following the store instruction. Therefore, the instruction execution time presented for store instructions is a typical value. The corresponding note for applicable store instructions indicates some criteria for determining the exact time required to execute a specific store instruction. Only simple store instructions operate in this fashion. These are; ST, STH, STE and STB.

16.6 STORE CONFLICT - MODIFY PREFETCHED MEMORY OPERAND

Store conflicts are a result of prefetching operands from memory. An operand

- o Not a hazard or conflict
- o Causes additional 2 cycle delay due to memory — total execution .75 → .25 us
- o Example:

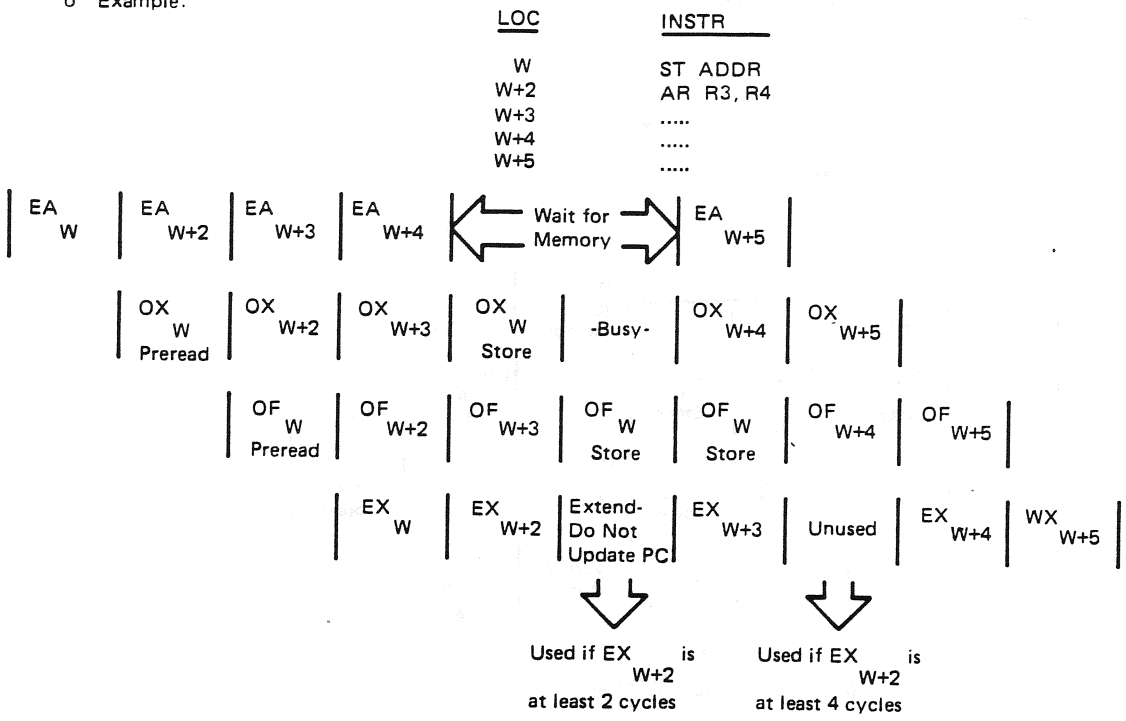


Figure 16-7. Store Instruction

prefetch for a load instruction will actually occur before the memory write is done for a store instruction which precedes the load. If the load and store instructions involve the same memory address, then the operand prefetch for the load instruction must be postponed until the memory write is completed, as shown in Figure 16-8. (The operand fetch actually occurs, however, the data is discarded). In order to minimize the penalty involved, store conflicts are detected and totally controlled by hardware resources. Any penalty involved with store conflicts has not been included with the stated instruction times, and must be evaluated separately if necessary. Store conflicts are applicable for simple store instructions only.

The store conflict hardware has been simplified somewhat by assuming that all memory operations involve two locations, or 32 bits. Therefore, the conflicting instructions only need to deal with memory locations which are within one location of each other in order to cause the detection of a store conflict. Furthermore, store conflicts are detected on the 16 bit logical address, and not the 19 bit physical address. In order to guarantee proper operation with expanded memory addressing, store conflicts are detected on the 15 least significant bits of the logical address. Addresses 7FFF and 0000 are considered to be contiguous, as are addresses FFFF and 8000. At most, two machine cycles will be lost while the operand fetch is postponed. This penalty will decrease to one machine cycle if one other instruction is executed between the conflicting instructions. No conflict will exist if there are two or more intervening instructions.

- o Caused by store with successive load from memory within 2 instructions
- o Detected and handled by hardware
- o Example:

LOC	INSTR
W	ST ADDR
W+2	L ADDR
W+4	

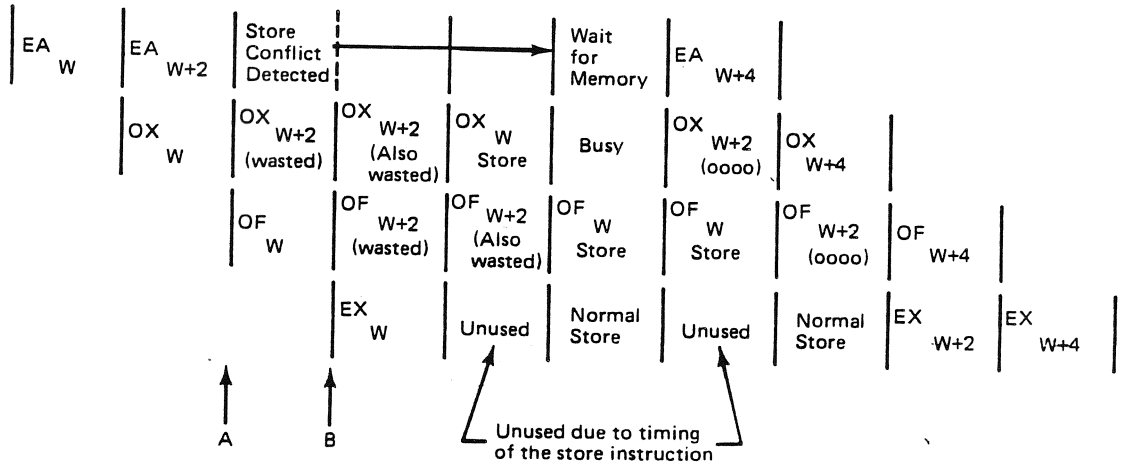


Figure 16-8. Store Conflict

16.7 SUCCESSIVE STORES - BACK-TO-BACK STORES

The execution unit of the CPU contains a store pending register which holds the memory address for simple store instructions. Since only one register exists, only one store instruction can reside in the pipeline at one time. Figure 16-9 indicates that processing by the EA unit for the second store instruction is postponed until the memory write for the first store instruction has been initiated. This situation is not a conflict or hazard, it is only a limitation of the hardware. The guidelines associated with store instruction execution times includes a case for a successive store condition. A penalty of 2 machine cycles has been included with the execution time of the first store instruction. This penalty will decrease to one machine cycle if one other instruction is executed between the store instructions. No penalty exists if there are two or more intervening instructions. The penalty for successive stores is applicable only for simple store instructions.

16.8 I UNIT HAZARD - MODIFICATION OF PREFETCHED INSTRUCTION

An I unit (instruction fetch unit) hazard is the result of a store instruction which modifies memory in the immediate area of the current instruction. The I unit can be at most 22 memory locations ahead of the current instruction. If a store

- o Caused by consecutive store instructions within 2 instructions
- o Detected & handled by hardware
- o Due to existence of one address register for CPU stores

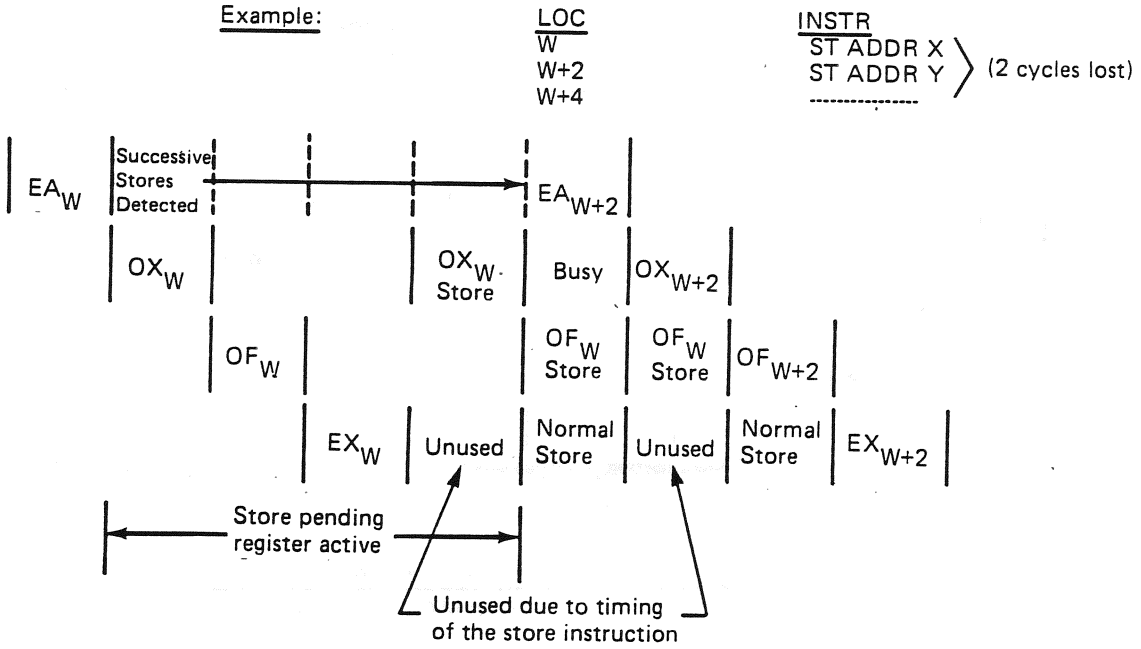


Figure 16-9. Successive Stores

instruction writes to memory in the area from which the I unit may have already prefetched instructions, then an I unit hazard exists. The actual detection circuitry uses the range of IC-1 to IC+23 in order to indicate an I unit hazard. Once a hazard has been detected, the entire pipeline is discarded and restarted from the location following the current instruction, as indicated in Figure 16-10.

I unit hazards are detected on the 16 bit logical address, and not the 19 bit physical address. In order to guarantee proper operation with expanded memory addressing, I unit hazards are detected on the 15 least significant bits of the logical address. Addresses 7FFF and 0000 are considered to be contiguous, as are addresses FFFF and 8000.

The I unit hazard circuitry is provided in order to guard against self-modifying code. This circuitry forces a restart of the pipeline to guarantee that the proper instructions, including modified instructions, are executed. However, it is possible to modify a data location at the end of a program segment and cause an I unit hazard. The I unit hazard circuitry cannot distinguish between memory used for instructions as opposed to data. Therefore, any store within the indicated range will cause an I unit hazard condition whether it is real or not.

16.9 CONFLICT/HAZARD SUMMARY

All effects of the pipeline on instruction execution times have been included with the indicated times except for register conflicts, store conflicts, and I unit

- o Caused by a store into memory within the immediate area of the current instruction (-1 ↔ PC ↔ +23)
- o Forces a restart of the I-unit and pipeline
- o Detected by hardware. Handled by microcode

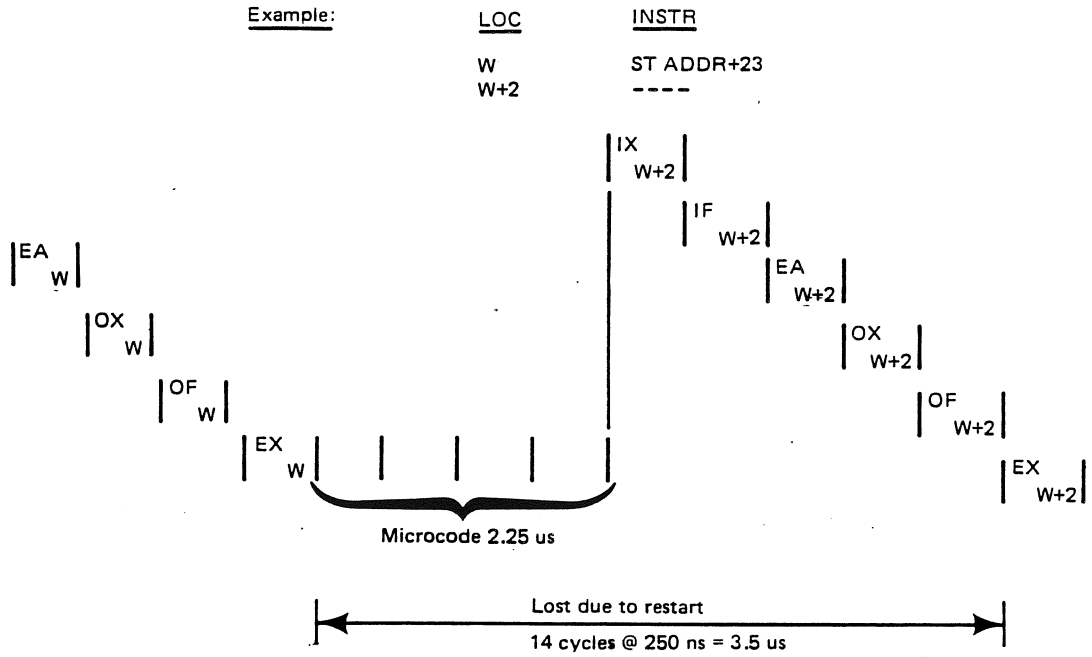


Figure 16-10. I Unit Hazard

hazards. Below is a summary of the penalties involved with each.

	Number of Intervening Instructions		
	0 instr	1 instr	2 instr
Register Conflict	.75 us	.50 us	.25 us
Store Conflict	.50 us	.25 us	----

	Independent of Intervening Instructions
I Unit Hazard	3.50 us

17.0 AP-101S INSTRUCTION EXECUTION TIMES

All floating point execution times have been rounded up to the nearest multiple of 250 nanoseconds and are based on the following assumptions:

- Neither operand is zero, and for the long (64-bit) instructions neither hi or low words of an operand is zero.
- All results will require normalization of 8 bits (2 hex digits).
- All operands are normalized, hence prenormalization of the divisor in the divide instructions is unnecessary.
- For instructions requiring prealignment (Add, Subtract, Compare) the difference in exponents will be 4.
- Operands will not be the same signs (except for the COMPARE instructions in which operands will have identical signs).

MMP INSTRUCTION			INSTRUCTION EXECUTION TIME IN US						
			NORMAL ADDRESSING MODES	DOUBLE INDIRECTION XC=0 C =0	DOUBLE INDIRECTION XC=0 C =1	DOUBLE INDIRECTION XC=1 C =0	DOUBLE INDIRECTION XC=1 C =1	AUTO STORAGE MODIFICATION	AUTO INDEXING
A	RS		.250	4.5	4.25	4.25	4.25	5.5	7.25
A	SRS		.250	—	—	—	—	—	—
AE	RS		2.50	6.75	6.5	6.5	6.5	7.5	9.0
AE	SRS		2.50	—	—	—	—	—	—
AED	RS		6.50	10.5	10.25	10.25	10.25	11.5	13.25
AEDR	RR		6.25	—	—	—	—	—	—
AER	RR		2.25	—	—	—	—	—	—
AH	RS		.250	4.50	4.25	4.25	4.25	5.50	7.0
AH	SRS		.250	—	—	—	—	—	—
AHI	RI		.250	—	—	—	—	—	—
AR	RR		.250	—	—	—	—	—	—
AST	RS		.750	6.0	7.0	5.75	7.0	8.25	10.25
BAL	RS		3.75	7.0	10.0	6.75	10.0	8.0	9.5
BALR	RR		BT=3.50; BNT=4.50	—	—	—	—	—	—
BC	RS		BT=1.25; BNT=.250	4.25	7.25	4.0	7.25	5.25	6.25
BCB	SRS		.250	—	—	—	—	—	—
BCF	SRS		.250	—	—	—	—	—	—
BCR	RR		.250	—	—	—	—	—	—
BCRE	RR		BT=5.75; BNT=.50	—	—	—	—	—	—
BCT	RS		BT=1.75; BNT=.750	4.5	7.5	4.25	7.5	5.5	7.0
BCTB	SRS		BT=1.75; BNT=.750	—	—	—	—	—	—
BCTR	RR		BT=1.75; BNT=.750	—	—	—	—	—	—
BIX	RS		BT=2.5; BNT=1.5	5.75	8.7	5.5	8.75	6.75	8.25
BVC	RS		BT=1.25; BNT=.50	4.0	7.0	3.75	7.0	5.0	6.5
BVCF	SRS		BT=1.25; BNT=.50	—	—	—	—	—	—
BVCR	RR		BT=1.25; BNT=.50	—	—	—	—	—	—
C	RS		.250	4.5	4.25	4.25	4.25	5.5	7.25
C	SRS		.250	—	—	—	—	—	—
CBL	RR		AVG. = 5.0	—	—	—	—	—	—
CE	RS		1.75	6.0	5.75	5.75	5.75	6.75	8.5
CED	RS		5.75	9.75	9.5	9.5	9.5	10.75	12.5
CEDR	RR		5.50	—	—	—	—	—	—
CER	RR		1.50	—	—	—	—	—	—
CH	RS		.250	4.50	4.25	4.25	4.25	5.50	7.0
CH	SRS		.250	—	—	—	—	—	—
CHI	RI		.250	—	—	—	—	—	—
CIST	SI		1.5	—	—	—	—	—	—
CR	RR		.250	—	—	—	—	—	—
CVFL	RR		1.75	—	—	—	—	—	—
CVFX	RR		2.25	—	—	—	—	—	—
D	RS	(R1 EVEN)	AVG. = 4.925	9.05	8.8	8.8	8.8	10.05	11.8
D	RS	(R1 ODD)	AVG. = 4.675	8.8	7.55	7.55	7.55	9.8	10.05
D	SRS	(R1 EVEN)	AVG. = 4.925	—	—	—	—	—	—
D	SRS	(R1 ODD)	AVG. = 4.675	—	—	—	—	—	—
DE	RS		7.50	12	11.5	11.5	11.5	12.75	15.25
DE	SRS		7.50	—	—	—	—	—	—
DED	RS		23.00	27.75	27.75	27.75	27.75	28.75	29.75
DEDR	RR		22.75	—	—	—	—	—	—
DER	RR		7.25	—	—	—	—	—	—
DIAG	RS		SEE POO	—	—	—	—	—	—
DR	RR	(R1 EVEN)	AVG. = 4.925	—	—	—	—	—	—
DR	RR	(R1 ODD)	AVG. = 4.675	—	—	—	—	—	—
IAL	RS		.50	4.0	5.0	3.75	5.0	6.25	8.0
IAL	SRS		.50	—	—	—	—	—	—
ICR	RR		COMMAND DEPENDENT	—	—	—	—	—	—
IHL	RS		.50	4.75	4.50	4.50	4.50	5.75	7.25
ISPB	RS	(R1 = 0)	5.625	8.0	9.0	7.75	9.0	10.25	12.0
ISPB	RS	(R1 = 1)	5.625	8.0	9.0	7.75	9.0	10.25	12.0
ISPB	RS	(R1 = 2)	5.625	8.0	9.0	7.75	9.0	10.25	12.0
ISPB	RS	(R1 = 3)	5.625	8.0	9.0	7.75	9.0	10.25	12.0

MMP INSTRUCTION			INSTRUCTION EXECUTION TIME IN US						
			NORMAL ADDRESSING MODES	DOUBLE INDIRECTION XC=0 C =0	DOUBLE INDIRECTION XC=0 C =1	DOUBLE INDIRECTION XC=1 C =0	DOUBLE INDIRECTION XC=1 C =1	AUTO STORAGE MODIFICATION	AUTO INDEXING
ISPB	RS	(R1 = 5)	.125	—	—	—	—	—	—
ISPB	RS	(R1 = 6)	.125	—	—	—	—	—	—
ISPB	RS	(R1 = 7)	.125	—	—	—	—	—	—
L	RS		.250	4.5	4.25	4.25	4.25	5.5	7.25
L	SRS		.250	—	—	—	—	—	—
LA	RS		.250	4.0	5.0	3.75	5.0	6.25	8.0
LA	SRS		.250	—	—	—	—	—	—
LCR	RR		.50	—	—	—	—	—	—
LDM	RS		6.75	10.0	10.0	10.0	10.0	10.25	10.25
LE	RS		1.20	5.0	4.75	4.75	4.75	5.75	8.5
LE	SRS		1.20	—	—	—	—	—	—
LECR	RR		1.00	—	—	—	—	—	—
LED	RS		1.50	5.5	5.0	5.0	5.0	6.25	8.75
LER	RR		1.00	—	—	—	—	—	—
LFLI	RR		.750	—	—	—	—	—	—
LFLR	RR		.750	—	—	—	—	—	—
LFXI	RR		.750	—	—	—	—	—	—
LFXR	RR		.750	—	—	—	—	—	—
LH	RS		.250	4.50	4.25	4.25	4.25	5.50	7.0
LH	SRS		.250	—	—	—	—	—	—
LM	RS		8.5	12.25	13.25	12.0	13.25	14.5	16.25
LPS	RS		10.25	13.25	14.25	13.0	14.25	15.5	17.25
LR	RR		.250	—	—	—	—	—	—
LXA	RR		3.50 (-1.25 for early out)	—	—	—	—	—	—
LXA	RS		3.50 (-1.25 for early out)	6.50	6.25	6.25	6.25	6.50	5.25
M	RS	(R1 EVEN)	2.40	6.53	7.53	6.28	7.53	8.78	10.53
M	RS	(R1 ODD)	2.15	6.28	7.28	6.03	7.28	8.53	10.28
M	SRS	(R1 EVEN)	2.40	—	—	—	—	—	—
M	SRS	(R1 ODD)	2.15	—	—	—	—	—	—
ME	RS	(R1 EVEN)	6.25	10.5	10.25	10.25	10.25	11.5	13.25
ME	RS	(R1 ODD)	5.75	10.0	9.75	9.75	9.75	11.0	12.75
ME	SRS	(R1 EVEN)	5.75	—	—	—	—	—	—
ME	SRS	(R1 ODD)	5.75	—	—	—	—	—	—
MED	RS		19.00	22.5	22.25	22.25	22.25	24.25	25.75
MEDR	RR		18.50	—	—	—	—	—	—
MER	RR	(R1 EVEN)	6.00	—	—	—	—	—	—
MER	RR	(R1 ODD)	5.50	—	—	—	—	—	—
MH	RS		1.35	5.48	5.23	5.23	5.23	6.48	7.98
MH	SRS		1.35	—	—	—	—	—	—
MHI	RI		1.35	—	—	—	—	—	—
MIH	RS		AVG. = 1.7	5.83	5.58	5.58	5.58	6.825	8.025
MR	RR	(R1 EVEN)	2.40	—	—	—	—	—	—
MR	RR	(R1 ODD)	2.15	—	—	—	—	—	—
MSTH	SI		3.0	—	—	—	—	—	—
MVH	RR	(SRC-DEST=1)	9.5+1.75*N (-2.25 FOR DSR)	—	—	—	—	—	—
MVH	RR	(COUNT EVEN)	10.25+.875*N (-2.25 FOR DSR)	—	—	—	—	—	—
MVH	RR	(COUNT ODD)	12.0+.875*(N-1) (-2.25; DSR)	—	—	—	—	—	—
MVH	RR	(COUNT NEG)	7.5 (-2.25 FOR DSR)	—	—	—	—	—	—
MVH	RR	(COUNT ZERO)	7.75 (-2.25 FOR DSR)	—	—	—	—	—	—
MVS	RS		4.75	9.25	9.0	9.0	9.0	10.5	11.75
N	RS		.250	4.75	4.5	4.5	4.5	5.75	6.5
N	SRS		.250	—	—	—	—	—	—
NCT	RR		1.05 + (.075 * N)	—	—	—	—	—	—
NHI	RI		.250	—	—	—	—	—	—
NIST	SI		3.0	—	—	—	—	—	—
NR	RR		.250	—	—	—	—	—	—
NST	RS		.750	6.0	7.0	5.75	7.0	8.25	10.25
O	RS		.250	4.75	4.5	4.5	4.5	5.75	6.5
O	SRS		.250	—	—	—	—	—	—
OHI	RI		.250	—	—	—	—	—	—
OR	RR		.250	—	—	—	—	—	—

MMP INSTRUCTION		INSTRUCTION EXECUTION TIME IN US						
		NORMAL	DOUBLE INDIRECTION				AUTO	AUTO
		ADDRESSING MODES	XC=0 C =0	XC=0 C =1	XC=1 C =0	XC=1 C =1	STORAGE MODIFICATION	INDEXING
OST	RS	.750	6.0	7.0	5.75	7.0	8.25	10.25
PC	RR	>4.25 BUT <22.5 (NO CUR DMA)	—	—	—	—	—	—
S	RS	.250	4.5	4.25	4.25	4.25	5.5	7.25
S	SRS	.250	—	—	—	—	—	—
SB	SI	3.0	—	—	—	—	—	—
SCAL	RS	18.125	21.5	24.5	21.25	24.5	22.5	24
SE	RS	2.50	4.75	4.5	4.5	4.5	4.5	9.5
SE	SRS	2.50	—	—	—	—	—	—
SED	RS	6.50	10.75	10.5	10.5	10.5	11.5	13.5
SEDR	RR	6.25	—	—	—	—	—	—
SER	RR	2.25	—	—	—	—	—	—
SH	RS	.250	4.50	4.25	4.25	4.25	5.75	7.25
SH	SRS	.250	—	—	—	—	—	—
SHW	RS	1.50	4.50	5.50	4.25	5.50	6.75	8.50
SHW	SRS	1.50	—	—	—	—	—	—
SLDL	SRS	1.0 + (0.25 * N); N>0	—	—	—	—	—	—
SLL	SRS	.675 + (0.1 * N); N>1	—	—	—	—	—	—
SPM	RR	5.25	—	—	—	—	—	—
SR	RR	.250	—	—	—	—	—	—
SRA	SRS	.650 + (0.1 * N); N>0	—	—	—	—	—	—
SRDA	SRS	1.0 + (0.25 * N); N>0	—	—	—	—	—	—
SRDL	SRS	1.0 + (0.1 * N); N>0	—	—	—	—	—	—
SRDR	SRS	2.0 + (0.5 * N); N<32	—	—	—	—	—	—
SRDR	SRS	2.0 + (0.5 * (N-32)); N>=32	—	—	—	—	—	—
SRET	RR	17.50	—	—	—	—	—	—
SRL	SRS	.650 + (0.1 * N); N>0	—	—	—	—	—	—
SRR	SRS	.650 + (0.1 * N); N>0	—	—	—	—	—	—
SSM	RS	7.75	10.63	11.63	10.38	11.63	12.875	14.625
SST	RS	1.0	—	—	—	—	—	—
ST	RS	0.50	4.75	5.75	4.5	5.75	7.0	9.0
ST	SRS	0.50	—	—	—	—	—	—
STDM	RS	2.25	5.25	6.75	5.0	5.25	7.0	7.5
STE	RS	.500	4.75	4.5	4.5	4.5	4.5	7.5
STE	SRS	.500	—	—	—	—	—	—
STED	RS	1.00	5.25	5.0	5.0	5.0	5.0	7.5
STH	RS	.50	4.50	5.50	4.25	5.50	6.75	8.50
STH	SRS	.50	—	—	—	—	—	—
STM	RS	7.25	10.25	11.25	10.0	11.25	12.5	14.25
STXA	RR	2.50	—	—	—	—	—	—
STXA	RS	2.50	6.50	8.0	6.25	8.0	8.25	8.75
SUM	RR	2.5 * (# ELEMENTS TESTED)	—	—	—	—	—	—
SVC	RS	20.25	22.75	23.75	22.5	23.75	25.0	26.75
TB	SI	2.0	—	—	—	—	—	—
TD	RS	3.0	5.75	5.50	5.50	5.50	6.75	8.25
TD	SRS	3.0	—	—	—	—	—	—
TH	RS	1.75	5.25	5.0	5.0	5.0	6.25	7.75
TH	SRS	1.75	—	—	—	—	—	—
TRB	RI	1.0	—	—	—	—	—	—
TS	RS	3.75	6.50	6.25	6.25	6.25	7.50	9.0
TSB	SI	3.0	—	—	—	—	—	—
X	RS	.250	4.75	4.50	4.50	4.50	5.75	7.50
X	SRS	.250	—	—	—	—	—	—
XHI	RI	.250	—	—	—	—	—	—
XIST	SI	3.0	—	—	—	—	—	—
XR	RR	.250	—	—	—	—	—	—
XST	RS	.750	6.0	7.0	5.75	7.0	8.25	10.25
XUL	RR	1.0	—	—	—	—	—	—
ZB	SI	3.25	—	—	—	—	—	—
ZH	RS	1.50	4.50	5.50	4.25	5.50	6.75	8.50
ZH	SRS	1.50	—	—	—	—	—	—
ZRB	RI	.250	—	—	—	—	—	—

Appendix I

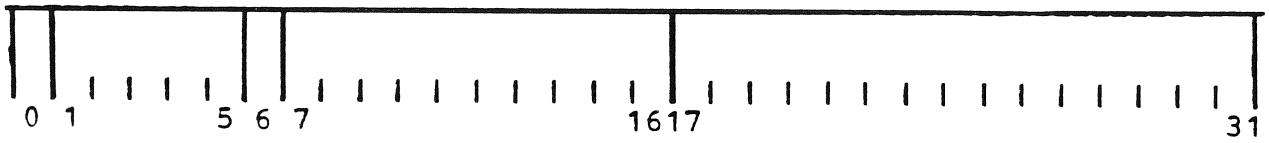
Input/Output Processor (IOP) — Principles of Operation for Program-Controlled Inputs & Outputs

TABLE OF CONTENTS

PCI/PCO PRINCIPLES OF OPERATION

This document identifies the specific Program Controlled Input and Output commands available to the Space Shuttle AP-101S computer user.

PCI/PCO COMMAND WORD FORMAT



BIT 0; COMMAND ID FIELD

1	=	Program Controlled Output (CPU Output)
0	=	Program Controlled Input (CPU Input)

BITS 1 Through 5; Subsystem SELECT FIELD

0 0 0 0 1	=	Control/Monitor (CM)
0 0 0 1 0	=	Redundancy Management (RM)
0 0 1 0 0	=	Data Flow (DF)
0 1 0 0 0	=	Local Store (LS)
1 0 0 0 0	=	Channel Control (CC)

BIT 6; HANDSHAKE CONTROL FIELD

0	=	No handshake required
1	=	Handshake required

BIT 7 through 16; DATA SELECT FIELD

(See Separate section)

BITS 17 through 31; IGNORED.

NOTES:

1. The five-bit Subsystem Select Field must contain only the bit specified in the format description for the desired subsystem selected. Additional bits will cause the PCI/PCO data word to be written to all the subsystems designated with associated loss of IOP control. No attempts to use the hardware configuration should be made since driving circuits are not sized to drive multiple loads and will not operate reliably.

2. Handshaking for a PCI/PCO is required for several operations to allow the subsystem selected to complete an operation before the PCI/PCO command function is implemented. The added operation is accomplished by the IOP and requires no special programming of the CPU. The handshaking operation prevents loss of control of the IOP software because of possible configuration changes during the PCI/PCO implementation.

PCO COMMAND WORD FORMAT SUMMARY

OCTAL	HEX	
301010 00000	C104 0000	DMA BURST ENABLE
300100 00000	C004 0000	DMA BURST INHIBIT
301400 00000	C180 0000	FORCE OCTAL MIA BAD PARITY
301200 00000	C140 0000	FORCE DMA ADD/DATA BAD PARITY
301020 00000	C108 0000	FORCE QUEUE CONTROL BAD PARITY
301004 00000	C102 0000	FORCE IOP H-BUS BAD PARITY
300002 00000	C001 0000	DISABLE PARITY CHECK
301002 00000	C101 0000	ENABLE PARITY CHECK
205010 00000	8504 0000	MIA XMTR ENABLE
204010 00000	8404 0000	MIA XMTR DISABLE
205020 00000	8508 0000	MIA RCVR ENABLE
204020 00000	8408 0000	MIA RCVR DISABLE
205040 00000	8510 0000	DISCRETE OUTPUT SET
204040 00000	8410 0000	DISCRETE OUTPUT RESET
206100 00000	8620 0000	PROCESSOR HALT
207100 00000	8720 0000	PROCESSOR ENABLE
204200 00000	8440 0000	MASTER RESET
210010 00000	8804 0000	LOAD GO/NO-GO TIMER
210011 00000	8804 8000	TEST GO/NO-GO TIMER
210020 00000	8808 0000	TERM. LATCH CONTROL
210040 00000	8810 0000	LOAD TEST REGISTER
210050 00000	8814 0000	ENABLE INTERRUPTS
210060 00000	8818 0000	TEST ALL INTERRUPTS
222000 00000	9200 0000	RESET STATUS 1(GO/NO-GO)
222010 00000	9204 0000	LOAD MSC BUSY
(SEE WITHIN)		LOAD LOCAL STORE
301001 00000	C100 8000	TEST DMA TIMER

PCI COMMAND WORD FORMAT SUMMARY

OCTAL	HEX	
004000 00000	0400 0000	MIA XMTR STATUS
004010 00000	0404 0000	MIA RCVR STATUS .
004020 00000	0408 0000	READ DISC. OUTPUT STATUS
004030 00000	040C 0000	PROCESSOR HALT STATUS
010000 00000	0800 0000	INTERRUPT REG. A
010010 00000	0804 0000	INTERRUPT REG. B
010020 00000	0808 0000	INTERRUPT REG. C
010030 00000	080C 0000	INTERRUPT REG. D
010040 00000	0810 0000	INTERRUPT REG. E
010050 00000	0814 0000	RM STATUS REG.
010060 00000	0818 0000	D.I.A. (1-32)
010070 00000	081C 0000	D.I.B. (33-40)
020000 00000	1000 0000	READ STATUS 1 (GO/NO-GO)
020010 00000	1004 0000	READ STATUS 4 (B/W)
(SEE WITHIN)		READ LOCAL STORE

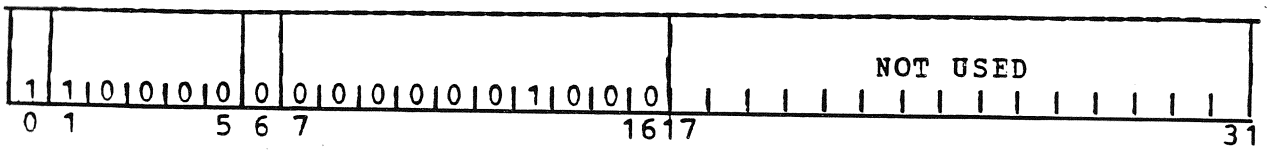
PCO FORMATS

DMA BURST

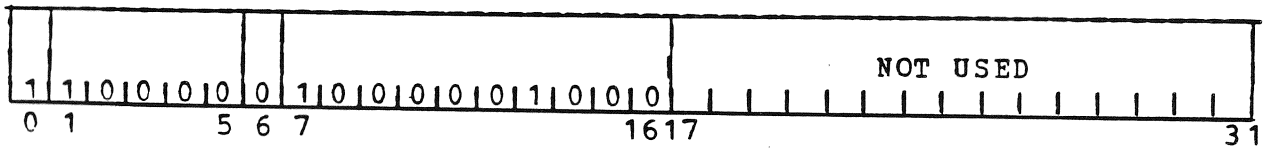
COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
DMA BURST INHIBIT	30001000000	C0040000	CC
ENABLE	30101000000	C1040000	

INHIBIT



ENABLE



DATA WORD

BITS	
0	NOT USED. THESE PCO REQUIRE NO DATA WORD.
1	.
.	.
.	.
.	.
31	.

These command words provide control of the Direct Memory Access capability to CPU main memory. When inhibited, the IOP will not access memory using Burst Mode. The commands are provided to allow CPU control of memory operations.

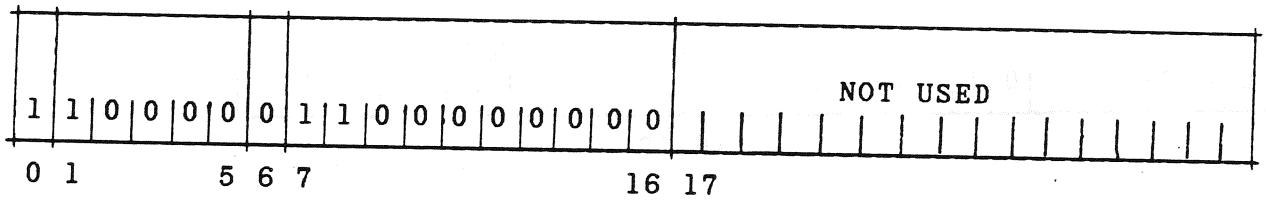
PCO FORMATS

FORCE OCTAL MIA BAD PARITY

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
FORCE BAD PARITY TO OCTAL MIA PAGES	30140000000	C1800000	CC

ENABLE



DATA WORD

BITS	
0	NOT USED. THIS PCO REQUIRES NO DATA WORD.
.	
.	
.	
31	

This command forces bad parity on all data transmitted from the IOP to the OCTAL MIA pages. The MIA page checks parity on all incoming command and data words.

This command can be reset by either Power on Reset, System Reset, or by issuing the PCO command "Disable Flow Parity Check".

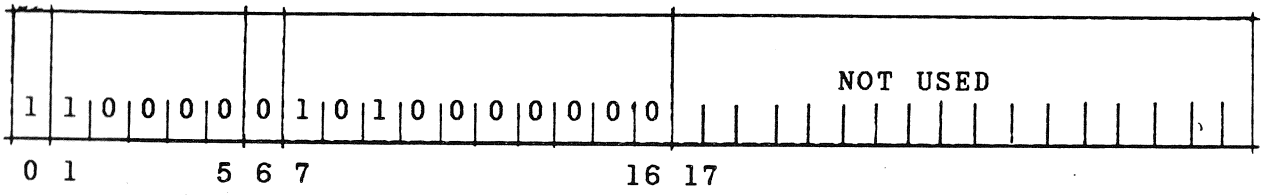
PCO FORMATS

FORCE DMA ADDRESS/DATA BAD PARITY

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
FORCE BAD PARITY ON DMA ADD./DATA	30120000000	C1400000	CC

ENABLE



DATA WORD

BITS

0 NOT USED. THIS PCO REQUIRES NO DATA WORD.

.

.

.

31

This command allows the operator to force bad parity on the DMA address or data bits individually, in order to check out each of the two parity checkers. To force bad parity on the DMA address only, a data word containing an odd number of 1's must be written to an odd parity address location with the above PCO active. To force bad parity on the DMA data only, a data word containing an even number of 1's must be written to an even parity address location with the above PCO active.

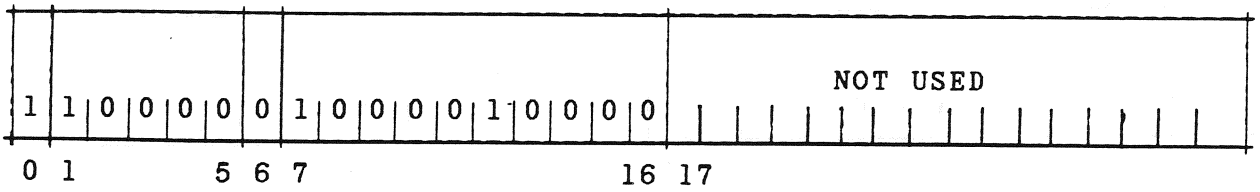
This command can be reset by either Power on Reset, System Reset, or by issuing the PCO command "Disable Flow Parity Check".

FORCE QUEUE CONTROL BAD PARITY

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
FORCE BAD PARITY ON THE QUEUE CONTROL BITS	30102000000	C1080000	CC

ENABLE



DATA WORD

BITS	
0	NOT USED. THIS PCO REQUIRES NO DATA WORD.
.	
.	
.	
31	

This command forces bad parity on the local store address and queue control bits.

This command can be reset by either Power on Reset, System Reset, or by issuing the PCO command "Disable Flow Parity Check".

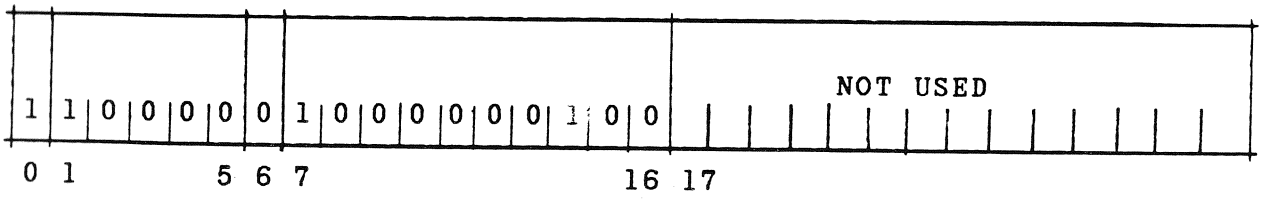
PCO FORMATS

FORCE IOP H-BUS BAD PARITY

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
FORCE BAD PARITY ON IOP H-BUS RECEIVED DATA	30100400000	C1020000	CC

ENABLE



DATA WORD

BITS	
0	NOT USED. THIS PCO REQUIRES NO DATA WORD.
.	
.	
.	
31	

This command forces bad parity on all data coming to the IOP via the H-Bus (PCO's or DMA's).

This command can be reset by either Power on Reset, System Reset, or by issuing the PCO command "Disable Flow Parity Check".

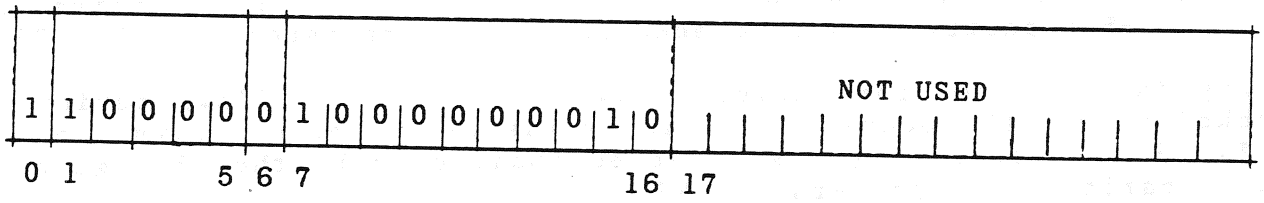
PCO FORMATS

DATA FLOW PARITY CHECK

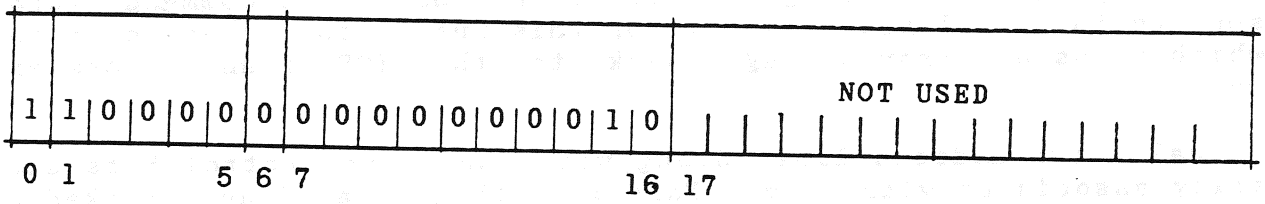
COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
ENABLE FLOW PARITY CHECK	30100200000	C1010000	CC
DISABLE FLOW PARITY CHECK	30000200000	C0010000	CC

ENABLE



DISABLE



DATA WORD

BITS	
0	NOT USED. THIS PCO REQUIRES NO DATA WORD.
.	
.	
.	
31	

The Enable Flow Parity Check PCO command is necessary to start the parity checking in the data flow following any event that disables parity checking. Events that disable parity checking include Power On, System Reset, and Disable Flow Parity Check PCO command. IOP Master Reset does not alter the state of the parity checkers.

The Disable Flow Parity Check PCO command disables the parity checkers. It also resets any parity generator which is forcing bad parity in response to one of the "force bad parity" PCOs.

Parity is generated in four locations in the IOP in order to detect single bit errors. Each of the four generators has its corresponding checker. In some cases because of the IOP bus structure, a single checker will check two different data paths. All generators and checkers work on odd parity and can be enabled and disabled under PCO control. All four checkers can be individually checked with the PCO commands to force bad parity. If a parity error does occur in the IOP, an external 1 interrupt is issued to the CPU and all BCE's and the MSC are halted, all transmitter and receiver enables are disabled and the discrete outputs are reset. The cause of this interrupt can be determined by reading the IOP interrupt register B.

All incoming H-BUS transfers from the CPU such as DMA's and PCI/O's have odd parity generated once it is received by the IOP. This data is then checked in two locations. The SI page checks the data for correct parity directly off the 'DEV OUT DATA BUS' and the IB page indirectly checks the H-BUS parity when it checks parity for registers R1, R2, R3.

Parity is generated for registers R4, R5, R6 which allows parity to be checked on all DMA address and data words before being driven to the CPU. R4, R5, R6 parity is also checked indirectly by the IB page when it checks parity for registers R1, R2, R3.

On the IB page parity is generated for all data and command words being sent to the octal MIA. Parity for this bus is then checked on the MIA's which sends an error message back to the IOP if any errors are detected.

The Local Store address lines along with the Queue control bits also have parity associated with them. This is both generated and checked on the MC page. The data flow parity checkers, and bad parity generators can be reset by either issuing the PCO command "Disable Flow Parity Check" or by Power on Reset.

Data flow parity checking must be enabled for the self test processor to set an external 1 interrupt if it detects an error. If parity is disabled no error indication is made and the self test processor continues.

IOP Local Store requires initialization before parity can be turned on. This is done on power on initialization, or after an IPL. Once a parity error occurs, R2 bit 0 is forced (as with an interface parity error in the AP101B), the MSC and BCEs are halted, all transmitters and receivers are disabled, and the discrete outputs are reset. These are the mechanisms used to inhibit further transmissions on any BCE. Any parity error causes an External 1 (level B) interrupt to the CPU.

To resume MSC and BCE operation one of the following must occur:

1) Cycle power on the unit.
2) IPL the unit.

NOTE: For testing purposes it is possible to force errors and resume if local store locations with bad parity are corrected and a Master Reset PCO is issued to reset the IOP.

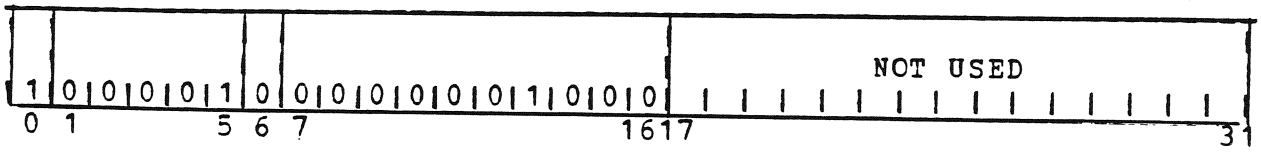
PCO FORMAT

MIA TRANSMITTER

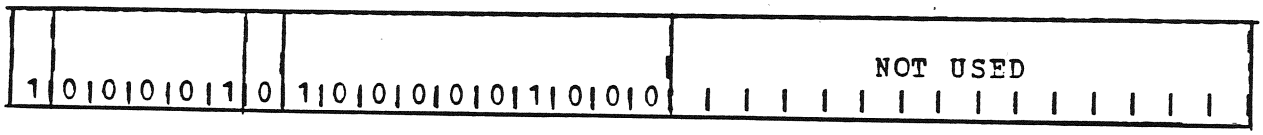
COMMAND WORD

FUNCTION		OCTAL	HEX	DEVICE
MIA TRANSMITTER	DISABLE	20401000000	84040000	C/M
	ENABLE	20501000000	85040000	

DISABLE



ENABLE



DATA WORD

0 = No change of condition. Hardware does not respond.

1 = Enable individual MIA transmitter if command word was enable.

1 = Disable individual MIA transmitter if command word was disable.

These words are to control individual MIA transmitters to provide IOP output control of system data buses. The data word is used as a mask for configuring MIA transmitters.

BIT	
0	NOT USED.
1	CHANNEL NO. 1 MIA TRANSMITTER
.	.
.	.
24	CHANNEL NO. 24 MIA TRANSMITTER
25 . . . 31	NOT USED

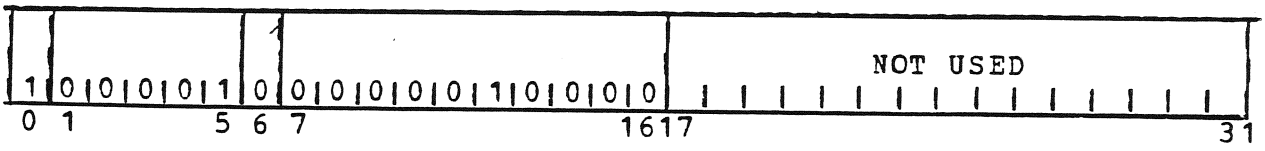
PCO FORMAT

MIA RECEIVER

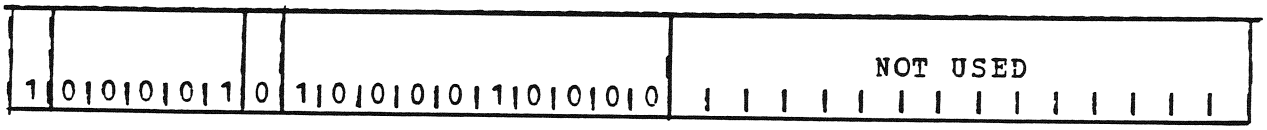
COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
MIA RECEIVER DISABLE	20402000000	84080000	C/M
ENABLE	20502000000	85080000	

DISABLE



ENABLE



DATA WORD

BIT

0	NOT USED. BIT IGNORED
1	CHANNEL NO. 1 MIA RECEIVER
2	CHANNEL NO. 2 MIA RECEIVER
.	.
.	.
23	No. 23
24	No. 24
25	NOT PRESENTLY USED. BITS IGNORED
.	.
.	.
31	.

0 = No Change of status

1 = Enable receiver if with enable command word*

1 = Disable receiver if with disable command word

*Should only be used when the associated BCE is in the Halt State.

These command and data words provide capability to control condition of the MIA receivers for purposes of channel control and data input. The data word is used as a mask for MIA configuration control.

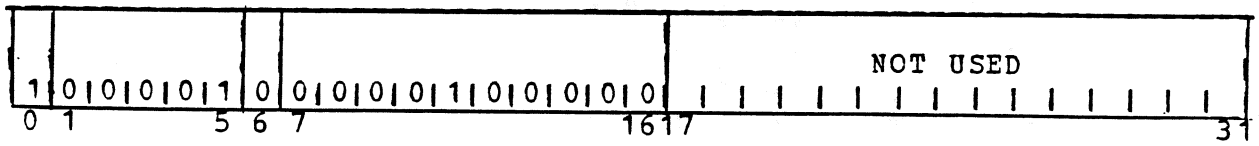
PCO FORMAT

DISCRETE OUTPUT

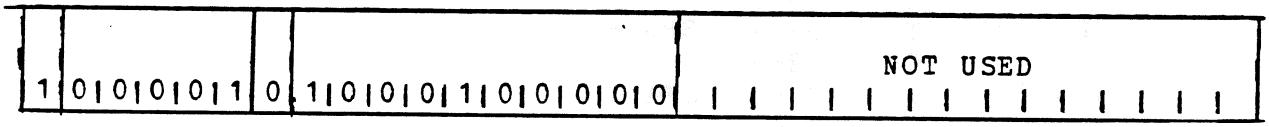
COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
DISCRETE OUTPUT RESET	20404000000	84100000	C/M
SET	20504000000	85100000	

RESET



SET



DATA WORD

* = Hardwired to IOP internal circuitry. These DO's are not available for software use.

0 = No change

1 = Set discrete bit if command word is set.

1 = Reset discrete bit if command word is reset.

The data word is used as a mask to control the discrete outputs. The discrete outputs are configured as differential drivers. The pin connections to the IOP are indicated for the true (T) and complement (C) outputs.

BIT				
0	(D0-0)	T	J3-51	SPARE
		C	J3-63	
1	(D0-1)	T	J3-18	SPARE
		C	J3-29	
2	(D0-2)	T	J3-28	SPARE
		C	J3-40	
3	(D0-3)	T	J3-30	SPARE

		C	J3-19	
4	(D0-4)	T	J3-41	SPARE
		C	J3-52	
5	(D0-5)	T	J3-59	SPARE
		C	J3-60	
6	(D0-6)	T	J3-49	SPARE
		C	J3-50	
7	(D0-7)	T	J3-36	I/O ACTIVE TALKBACK
		C	J3-48	
8	(D0-8)	T	J3-62	SPARE
		C	J3-61	
9	(D0-9)	T	J3-25	GPC READY TALKBACK
		C	J3-37	
10	(D0-10)	T	J3-15	SPARE
		C	J3-26	
11	(D0-11)	T	J3-38	SPARE
		C	J3-39	
12	(D0-12)	T	J3-8	MM1 RESET
		C	J3-16	
13	(D0-13)	T	J3-17	MM2 RESET
		C	J3-27	
14	(D0-14)	T	J3-1	SPARE
		C	J3-2	
15	(D0-15)	T	J3-3	SPARE
		C	J3-9	
16	(D0-16)	T	J3-4	SPARE
		C	J3-10	
17	(D0-17)	T	J3-11	SPARE
		C	J3-5	
#18	(D0-18)	T	J3-12	SPARE
		C	J3-6	
19	(D0-19)	T	J3-7	SPARE
		C	J3-13	
#20	(D0-20)	T	J3-44	GPC SELF SYNC 1
		C	J3-43	
21	(D0-21)	T	J3-22	SPARE
		C	J3-14	
#22	(D0-22)	T	J3-45	BFS RUN
		C	J3-33	
23	(D0-23)	T	J3-24	SPARE
		C	J3-23	
#24	(D0-24)	T	J3-34	GPC SELF SYNC 2
		C	J3-35	
25	(D0-25)	T	J3-56	SPARE
		C	J3-55	
#26	(D0-26)	T	J3-21	SPARE
		C	J3-32	
27	(D0-27)	T	J3-20	SPARE
		C	J3-31	
#28	(D0-28)	T	J3-46	GPC SELF SYNC 3
		C	J3-47	
29	(D0-29)	T	J3-42	SPARE
		C	J3-53	

*30	GPC ID (D0-30)	T	J3-58
		C	J3-57

This bit is hardwired to a positive output source and will provide a source voltage for GPC identification in the system.

*31	IPL (D0-31)	T	J3-68
		C	J3-67

This bit when set (1) indicates that the IPL routine is in progress. When reset (0), the bit indicates that the IPL routine has not been requested or that it is complete depending upon the time/event sequence.

* See I-13

High Speed Discretes

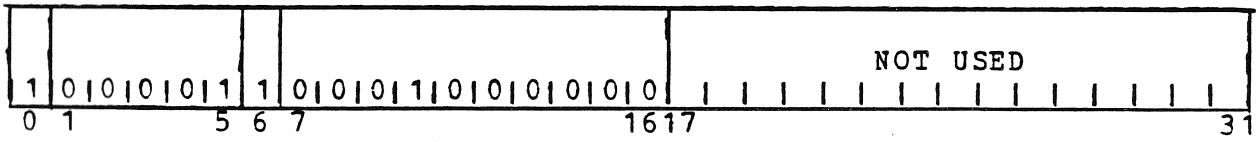
PCO FORMAT

CONFIGURE PROCESSORS

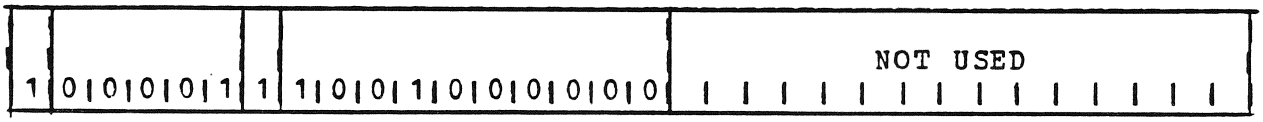
COMMAND WORD

FUNCTION		OCTAL	HEX	DEVICE
CONFIGURE PROCESSORS	HALT	20610000000	86200000	C/M
	ENABLE	20710000000	87200000	

HALT



ENABLE



DATA WORD

These PCO's provide capability to force individual MSC and BCE to the halt or operate states under GPC control. The data word is used as a mask to control the appropriate BCE/MSC processor.

0 = No change

1 = HALT if accompanied by the HALT command word.

1 = ENABLE if accompanied by the ENABLE command word.

BIT	
0	MSC
1	BCE NO. 1
2	2
.	.
.	.
23	23
24	24
25	SELF TEST
26	NOT USED. BITS IGNORED
.	.
.	.
31	.

The 25th BCE processor also called the self-test processor, is not associated with an I/O system bus. This processor executes diagnostic microcode that can detect certain faults in the IOP. One major purpose of the diagnostic microcode is to verify data flow paths in areas of the IOP where parity is not present. Another is to test basic microcode operations that normally occur during execution of the BCE #MOUT instruction.

Unlike the other BCE processors, the self-test processor can be in only one of two states, halt or enable. The halt state is entered after any system reset, or after a 'processor halt' PCO with the self-test bit of the data word set. In this state the error detection capabilities of the self-test processor are disabled and the processor is reset to a known condition. Exit from the halt state to the enable state can only be accomplished by a 'processor enable' PCO with the self-test bit of the data word set. In this state the error detection capabilities of the self-test processor are enabled. The current state of the self-test processor can be determined by the state of the self-test bit in the stat5 (halt/enable) register. This register can be read with a 'read processor halt status' PCI. Since there are only two states for the self-test processor, the stat4 (busy/wait) register has no meaning for processor 25 and the MSC has no control over processor 25's state.

If the self-test processor detects an error, and the Data Flow Parity Checking is enabled, an external 1 interrupt is sent to the CPU, the MSC and all BCEs are halted, all transmitters and receivers are disabled, and the discrete outputs are reset. The cause of this interrupt can be determined by reading IOP interrupt register B. Both the self-test bit of the statl (GO/NOGO) register, and processor 25's status register have no meaning for the self-test processor.

The diagnostic microcode uses 5 full word memory locations during execution. Location A4 is used to verify that the IOP can properly store data into the main memory. Therefore this location must not be store protected. Before the self-test can be enabled, locations A6-AC must be initialized to the full word constants shown in the table below:

Memory Locations Used by the Self-Test Processor

ADDRESS	DATA	
000A4	XXXXXXXX	*This location must not be store protected
000A6	33333333	
000A8	0F0F0F0F	
000AA	00FF00FF	
000AC	0000FFFF	

Data from the five memory locations will be continuously fetched from memory by the diagnostic microcode while the self-test processor is enabled. Therefore these memory locations must not be altered.

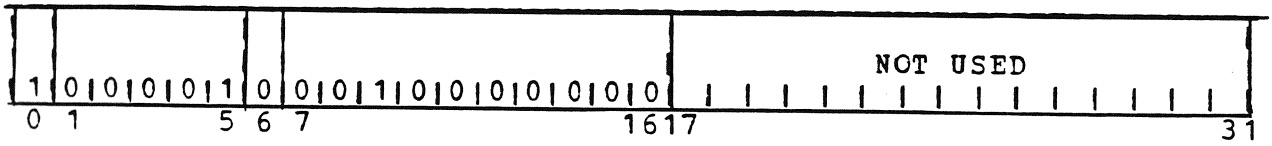
Note: Diagnostic processor 25 should not be enabled while the macro instruction "MSC self-test" is executing. MSC self-test modifies Proc 25's locations in Local Store which results in IOP diagnostic errors.

PCO FORMAT

MASTER RESET

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
MASTER RESET	20420000000	84400000	C/M



DATA WORD

There is no data word associated with this command word. The following table indicates the hardware reset by this command word and the resulting condition.

REGISTER	MASTER RESET FUNCTION
STAT1 (GO/NOGO)	RST=GO
STAT4 (BUSY/WAIT)	RST-WAIT
STAT5 (HALT/NO HALT)	RST=HALT MSC/BCE
XMIT ENABLE	RST=DISABLE
RCVR ENABLE	RST-DISABLE
CHANNEL	NO CHANGE
FAIL VOTE	RST=NO FAIL
TERMINATE CONTROL LATCHES	RST=NO TERMINATION
VOTER TEST	RST=OPERATIONAL DATA
FAIL LATCH	NO CHANGE
TIME OUT LATCH	NO CHANGE
DISCRETE OUTPUTS	RST=INACTIVE

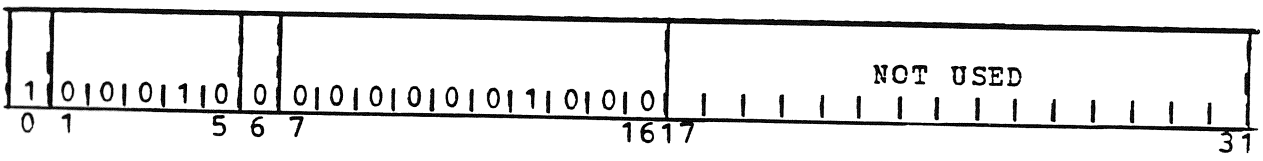
INTERRUPT	
-C/M IDLE	SET
-IOP FAIL LTCH	NO CHANGE
-TIME OUT LTCH	NO CHANGE
-ROS PAR	RESET
-IOP FAULT	RESET
-ALL OTHER INTERRUPTS	RESET
WATCHDOG TIMER	RST=ZERO COUNTER AND INHIBIT COUNTING

PCO FORMAT

LOAD GO/NO-GO TIMER

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
LOAD GO/NO-GO TIMER	21001000000	88040000	RM



DATA WORD

BIT	
0	NOT USED. BITS IF SET ARE IGNORED.
.	.
.	.
19	.
20	GO/NO-GO TIMER BIT 0 MSB
21	1
22	2
23	3
24	4
25	5
26	6
27	7
28	8
29	9
30	10
31	11 LSB = 0.768 msec.

Data word used to load the Go/NO-GO timer in normal system operation. The timer scaling permits 3.145728 seconds maximum to timeout. Timeout sets a timeout latch which is used to drive the Computer Fail output. Once the timer has been reset, the counter will not operate until loaded via this PCO.

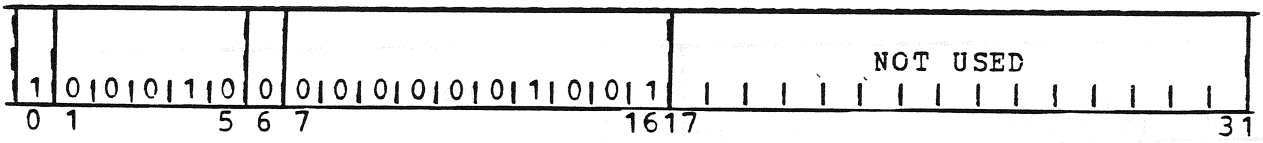
The timer is a countup device. The data loaded must be the twos complement of the desired time. The leading bit positions are ignored and may be set as a result of the complement operation if desired. A data word of all zeros causes an interrupt after a full count (3.145728 sec). This PCO also resets the timeout latch.

PCO FORMAT

LOAD GO/NO-GO TIMER TEST

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
LCAD GO/NO-GO TIMER TEST	21001100000	88048000	RM



DATA WORD

BIT		
0	NOT USED. BITS IGNORED	
.	.	
.	.	
19	.	
20	GO/NO-GO TIMER BIT 0	MSB
21	1	
22	2	
23	3	
24	4	
25	5	
26	6	
27	7	
28	8	
29	9	
30	10	
31	11	LSB = 0.768 msec.

This PCO is used to load the Go/No-Go Timer with any chosen value which is incremented by one low order bit and read with a PCI (READ STATUS REGISTER) to determine the operating status of the timer.

The Terminate Output driver is permanently inhibited by the hardware.

The Terminate Output latch is reset after the time out interrupt has been generated.

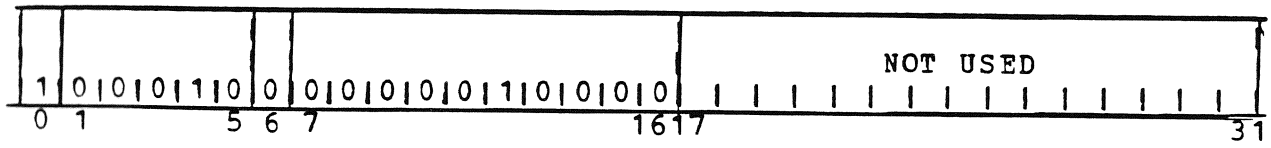
The timer is a countup device. The desired code loaded in the data word must be the twos complement of the desired timeout value. A data word of all zeros causes an interrupt after a full count. This PCO also resets the timeout latch.

PCO FORMAT

CONFIGURE TERMINATION CONTROL LATCHES

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
CONFIGURE TERMINATION CONTROL LATCHES	21002000000	88080000	RM



DATA WORD

BIT	
0	NOT USED. BITS IGNORED
.	.
.	.
29	.
30	TIMEOUT TERMINATION LATCH
31	VOTER TERMINATION CONTROL LATCH

0 = Reset
1 = Set

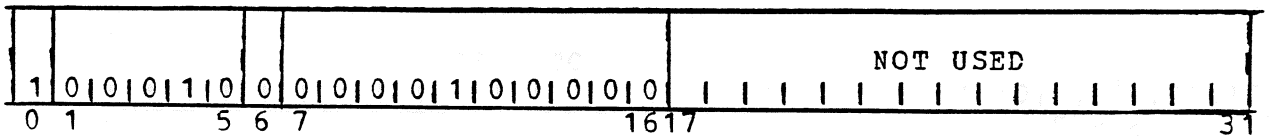
These two bits permit control of the associated RM latches by CPU software (PCO) for purposes of testing and to force error indications (Computer Fail and IOP Transmission Termination) for CPU detected faults. The error indications may be inhibited for self testing.

PCO FORMAT

LOAD TEST REGISTER

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
LOAD TEST REGISTER	21004000000	88100000	RM



DATA WORD

BIT

0 NOT USED. BITS IGNORED
.
.
26

27 VOTER TEST CONTROL
0 = Permit normal operation
1 = Inhibit normal inputs (perform test)

This bit inhibits the normal voter inputs (when set) from the other IOP's and inhibits driving of the Computer Fail latch and IOP Transmissions Termination logic. This is used to test the RM voter logic.

28 VOTER TEST INPUT 1
29 2
30 3
31 4

0 = Not failed test input
1 = Failed test input

These bits are used to load a register to provide test inputs to the voter logic for test purposes. Bit 27 (above) must be set to prevent erroneous system failure indications.

The IOP Voter test hardware will respond to these test inputs (Bits 28-31) or to the normal operational inputs. The Test Inputs and Operational Inputs are logically OR'ed.

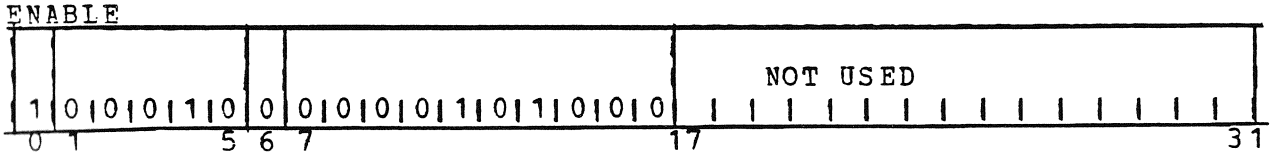
PCO FORMAT

INTERRUPTS

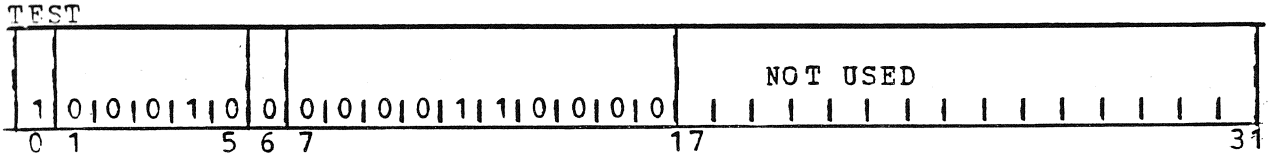
COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
TEST INTERRUPTS	21006000000	88180000	RM
ENABLE INTERRUPTS	21005000000	88140000	RM

ENABLE



TEST



DATA WORD

BIT	
0 . . . 31	NOT USED. THESE PCO REQUIRE NO DATA WORD.

The TEST command word forces interrupt Registers A, B, D, and E to set all interrupts as follows:

REG A	BITS 0-5	(FC00 0000)
REG B	BITS 4&5	(0C00 0000)
REG D	BIT 0	(8000 0000)
REG E	BIT 0	(8000 0000)

The interrupts will stay set until the action described below is taken. This permits self-testing of the interrupt detection circuitry. The interrupt registers will not be reset by reading of the registers as in normal operation.

The ENABLE command must be issued after the TEST command word to remove the test interrupt. After issuing this PCO, each of the four registers must be read to reset them and to permit normal operation.

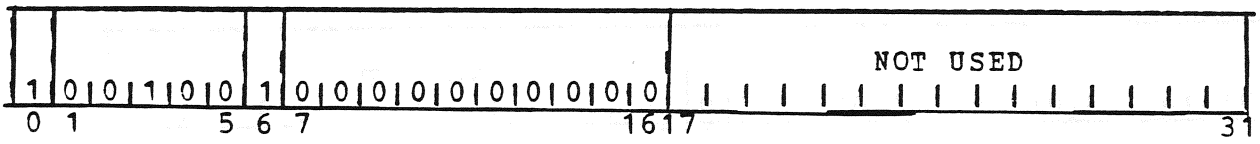
Note: Test and Enable do not force the new parity interrupts.

PCO FORMAT

RESET STATUS 1 (GO/NO-GO)

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
RESET STATUS1(GO/NO-GO)	22200000000	92000000	DF



DATA WORD

These PCO's provide the capability (data Word is used as Mask) to reset Status Register 1 to the normal or GO indicator. The entire word must be configured for each application.

0 = No Change
1 = Reset Status

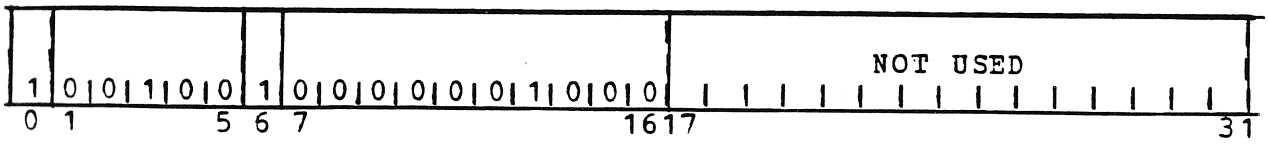
BIT		
0	MSC	
1	BCE NO. 1	
2	2	
.	.	
.	.	
23	23	
24	24	
25	NOT USED	BITS IGNORED
26	.	
.	.	
.	.	
31	.	

PCO FORMAT

LOAD MSC BUSY

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
LOAD MSC BUSY	2220 1000000	92040000	DF



DATA WORD

No data word required.

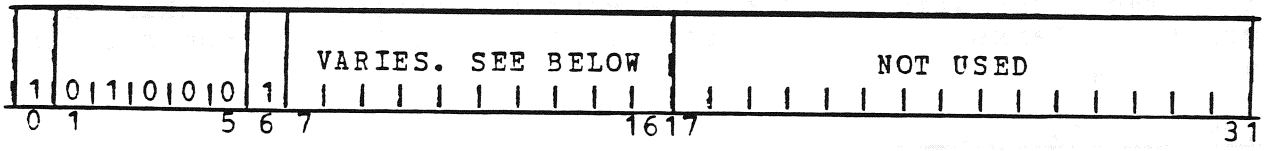
This PCO command word forces a BUSY condition to the MSC. It is required to start the micro-processing of MSC/BCE operations after system power-up and initialization.

PCO FORMAT

LOAD LOCAL STORE

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
LOCAL STORE			LS



The 10(10) bits (7 through 16) associated with the device select field for this command word varies depending upon the word (location) loaded in Local Store. The command word provides access to the Local Store area associated with the MSC functions, 24 BCE functions, and Self-Testing of the IOP. The specific data stored in each location is discussed in the MSC and BCE Principals of Operation documents and involve that data required for initializing and operating the MSC and BCE's.

Bits 7 through 11

This 5-bit field is used to designate the MSC or BCE to which the data word must be transferred as shown below:

BITS 7	8	9	10	11	
0	0	0	0	0	MSC REGION
0	0	0	0	1	BCE NO. 1 REGION
0	0	0	1	0	BCE NO. 2 REGION
		.			.
		.			.
1	1	0	0	0	BCE NO. 24 REGION
1	1	0	0	1	SELF TEST

Bits 12 and 13

This 2-bit field identifies the bank (A, B, or C) in Local Store for the above defined MSC or BCE area to which the data word must be transferred as indicated below.

BITS	12	13	
	0	0	BANK A
	0	1	BANK B
	1	0	BANK C

Bits 14,15 and 16

This 3-bit field is used to identify which word in a Bank is loaded. The resolution to a single word is achieved when this field is completed. Banks A and B use only Bits 15 and 16 to control the four words required.

BITS	14	15	16	
	0	0	0	LOCATION 0
	0	0	1	LOCATION 1
		.		.
		.		.
	1	1	1	LOCATION 7

DATA WORD

The single data word for this command must contain the data word (18(10) bits) scaled to the LSB portion of the 32-bit data word as indicated below.

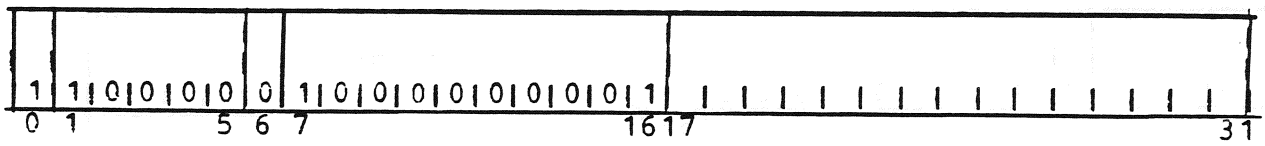
PCO DATA WORD BIT	LOCAL STORAGE WORD BIT	
0	NOT USED	
1	.	
.	.	
.	.	
13	.	
14	0	MSB
15	1	
.	.	
.	.	
31	17	LSB

PCO FORMAT

TEST DMA 8 MICRO SECOND TIMER

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
INHIBIT COMPLETION OF A DMA CYCLE	30100100000	C1008000	CHANNEL



DATA WORD

BIT	
0 . . . 31	NOT USED. THIS PCO REQUIRES NO DATA WORD

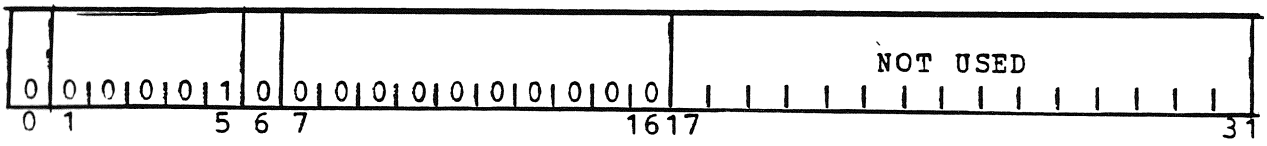
The Test DMA 8 Microsecond Timer command word causes the channel to inhibit completion of a DMA cycle. This permits self-testing of the DMA 8 microsecond timer and associated interrupt logic. Only the first DMA cycle occurring after reception of the Test DMA 8 Microsecond Timer command word will be inhibited.

PCI FORMAT

READ MIA TRANSMITTER STATUS

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
READ MIA TRANSMITTER STATUS	00400000000	04000000	C/M



DATA WORD

BIT

0	CHANNEL NO. 1 MIA TRANSMITTER
1	CHANNEL NO. 2 MIA TRANSMITTER
.	.
.	.
23	CHANNEL NO. 24 MIA TRANSMITTER
24	NOT USED. BITS, IF SET, ARE INVALID.
.	.
.	.
31	.

0 = Transmitter disabled
1 = Transmitter enabled

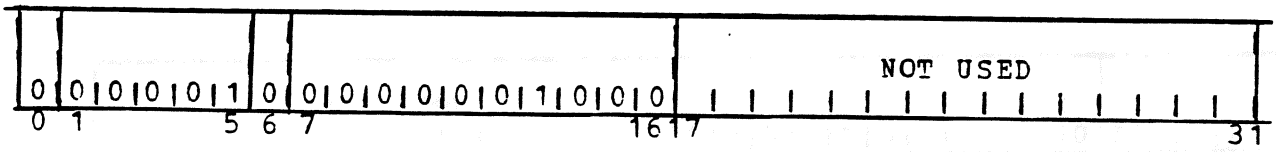
The PCI results in transfer, via the data word, of the configuration of the MIA enable register.

PCI FORMAT

READ MIA RECEIVER STATUS

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
READ MIA RECEIVER STATUS	00401000000	04040000	C/M



DATA WORD

BIT	
0	MIA CHANNEL NO. 1 RECEIVER
1	2
.	.
.	.
.	.
23	24
24	NOT USED. BITS, IF SET, ARE INVALID.
.	.
.	.
31	.

0 = Receiver disabled
1 = Receiver enabled

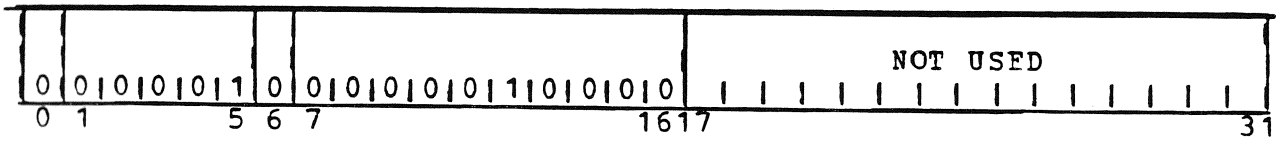
The PCI results in filling the data word with the contents of the MIA receiver control register.

PCI FORMAT

READ DISCRETE OUTPUT STATUS

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
READ DISCRETE OUTPUT STATUS	00402000000	04080000	C/M



DATA WORD

BIT	
0	DISCRETE OUTPUT NO. 1
1	2
.	.
.	.
31	32

0 = Discrete Output Disabled (OFF)
1 = Discrete Output Enabled (ON)

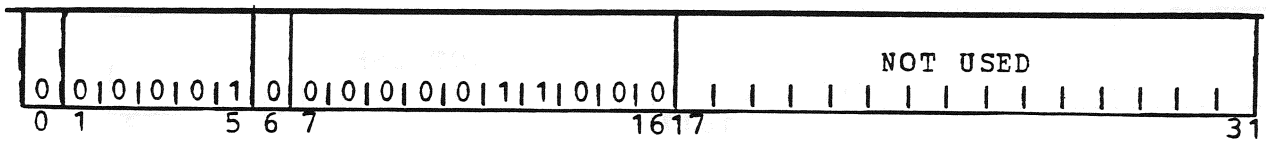
The PCI provides the capability to read the D.O. register.

PCI FORMAT

READ PROCESSOR HALT STATUS

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
READ PROCESSOR HALT STATUS	00403000000	040C0000	C/M



DATA WORD

BIT	
0	MSC
1	BCE No. 1
2	BCE No. 2
.	.
.	.
24	BCE No. 24
25	SELF TEST PROCESSOR
26	NOT PRESENTLY USED. BITS, IF SET, ARE INVALID.
.	.
.	.
31	.

0 = Processor (MSC or BCE) Disabled
1 = Processor Enabled

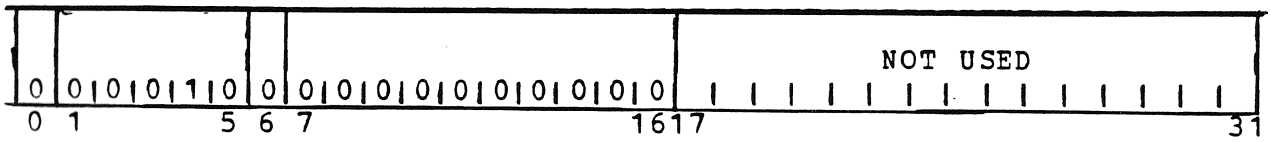
The PCI provides access to Status Register 5 (The Halt Register). The data indicates the status (enabled or disabled) of each BCE (25) and the MSC.

PCI FORMAT

READ INTERRUPT REGISTER A/GROUP 1

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
READ INTERRUPT REGISTER A	01000000000	08000000	RM



DATA WORD

Only the first 6 bits of this data word are valid since the interrupt register contains 6 bits. The remainder of the data will be zeros.

All bits in the interrupt register are set to zero when data is transferred for this PCI command or when the CPU issues an ICR "reset channel".

0 = No interrupt
1 = Interrupt

BIT

0 GO/NO-GO TIMER
Timer has timed out and generated the interrupt.

1 IOP FAIL LATCH

The IOP has detected a failure (Computer Fail) that effects capability of the machine. Signal originates in the RM Voter logic and not from the IOP transmission termination.

2 C/M IDLE

The IOP Control/Monitor logic is in the Idle mode and available for further operations.

3 ROS PARITY ERROR

A parity error has occurred during transfer from IOP Read Only Storage (ROS).

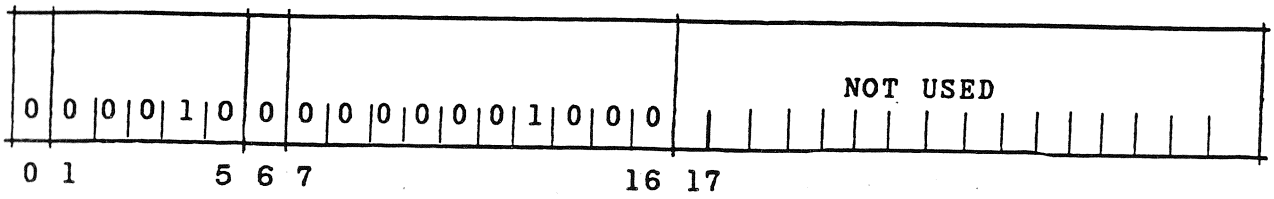
4	IOP FAULT This bit is set when a fault in the IOP timing is detected.
5	SPARE
6	NOT USED. UNDEFINED
.	.
.	.
31	.

READ INTERRUPT REGISTER B/GROUP 2

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
READ INTERRUPT REGISTER B	01001000000	08040000	RM

ENABLE



DATA WORD

The interrupt register read by this PCI contains 6 bits. Therefore, the 6 MSB's are valid and the remainder should be zeros. Reading the interrupt reg for this data word causes the bits to be reset to zero.

0 = No interrupt
1 = interrupt

BITS

0 NOT USED
1, 2, 3 BITS 1,2,3 ARE PRIORITY ENCODED AS FOLLOWS: IF MULTIPLE ERRORS OCCUR, ONLY THE HIGHEST PRIORITY EVENT WILL BE ANNUNCIATED.

BIT 1 2 3		ERROR CONDITION
0 0 0		No error
0 0 1	(Lowest Priority)	Dev out data parity error
0 1 0		R1,R2,R3, parity error
0 1 1		FB DMA add. or data parity error
1 0 0		MC queue control parity error
1 0 1		MIA parity error
1 1 0	(Highest Priority)	Diagnostic Proc 25 error
1 1 1		Not used
4		Queue overflow, the overflow circuitry

has detected more than 64 requests in the queue.

5 DMA 8 μ sec timeout, the 8 μ sec timer has detected a DMA that has been in process for more than 8 μ sec.

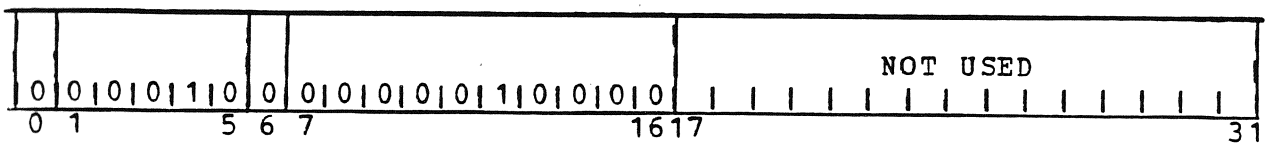
6 . . 31 NOT USED. UNDEFINED.

PCI FORMAT

READ INTERRUPT REGISTER C/GROUP 3

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
READ INTERRUPT REGISTER C	01002000000	08080000	RM



DATA WORD

The bits indicate which of the MSC generated interrupts are present. The PCI transfer causes the bits in Register C to be reset.

0 = No interrupt
1 = interrupt

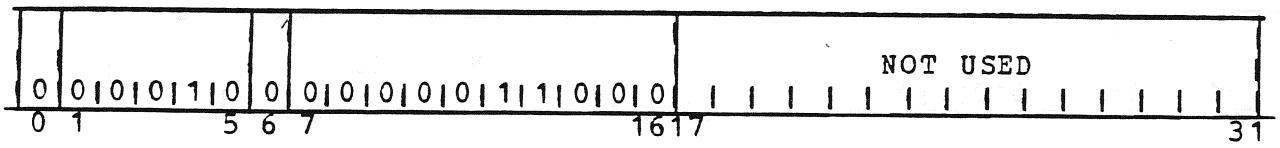
BIT	
0	MSC PROGRAM INTERRUPT 1
1	.
.	.
11	12
12	NOT USED (ZEROS) UNAVAILABLE.
.	.
.	.
31	.

PCI FORMAT

READ INTERRUPT REGISTER D/GROUP 4

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
READ INTERRUPT REGISTER D	01003000000	080C0000	RM



DATA WORD

This PCI reads a four-bit register and the bits are reset as a result of the PCI.

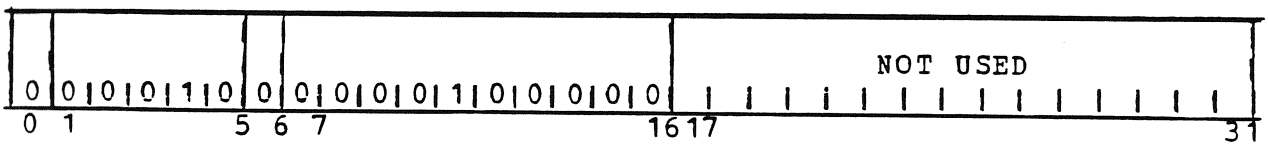
BIT	
0	SPARE
1	NOT USED (ZEROS) UNAVAILABLE.
2	.
3	.
4	.
.	.
.	.
31	.

PCI FORMAT

READ INTERRUPT REGISTER E/GROUP 5

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
READ INTERRUPT REGISTER E	01004000000	08100000	RM



DATA WORD

This is the read out of a four-bit register.

BIT

0	SPARE	
1	NOT USED (ZEROS) UNAVAILABLE.	
2	.	
3	P. S. OVER TEMPERATURE	0 = GOOD 1 = BAD
4	CHARGER STATUS	
5	BATTERY STATUS	
6 . . 31	NOT USED. UNDEFINED.	

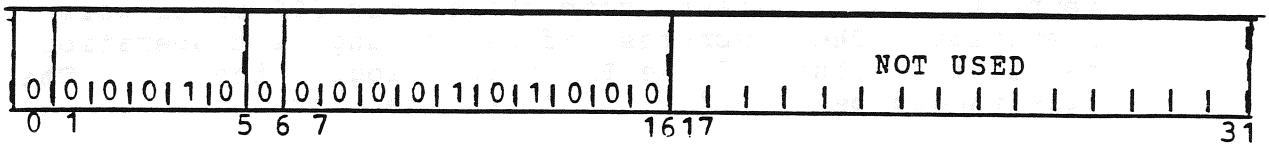
Note: These errors do not generate interrupts and reading the interrupt register does not reset the bits. They are used for status only.

PCI FORMAT

READ RM STATUS REGISTER

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
READ RM STATUS REGISTER	01005000000	08140000	RM



DATA WORD

Filled from IOP RM status register to provide status of IOP hardware voting logic and data output termination logic. This PCI does not cause reconfiguration of the Status Register.

BIT

0 FAIL OR TIMEOUT LATCH

0 = NO FAILURE, NORMAL OPERATION

1 = FAILURE INDICATOR IS SET.

Indicates that RM has detected a failure and set the failure latch or that the watchdog timer has timed out forcing the fail latch. Also set during test modes.

1 PCO INHIBIT FAIL VOTE INPUTS FOR TEST

0 = NORMAL OPERATION

1 = PCO inhibiting voter inputs while IOP performs RM logic self testing.

2 MSC INHIBIT FAIL VOTE OUTPUTS FOR TEST

0 = Normal operation

1 = MSC inhibiting fail voter output while IOP performs self testing.

3 FAILURE VOTE 1 INPUT

4 FAILURE VOTE 2 INPUT

5 FAILURE VOTE 3 INPUT

6 FAILURE VOTE 4 INPUT

0 = NO FAILURE

1 = FAILURE INDICATES

These bits represent the inputs from other IOP's each representing a failure vote while this IOP is in normal operation. The discretes, if set during test operation, represent failure of the RM voter logic inputs. The bits should be zero during RM self testing.

7 FAILURE VOTE 1 OUTPUT

8 FAILURE VOTE 2 OUTPUT

9 FAILURE VOTE 3 OUTPUT

10 FAILURE VOTE 4 OUTPUT

0 = NO FAILURE

1 = FAILURE VOTE

These bits represent the output (before drivers and output inhibit latches) of the RM software logic to the other IOP's. The register for these bits is set upon PCO from the CPU by the MSC. The data represents output to the other IOP's when in normal system operation.

11 TEST FAILURE VOTE 1 INPUT

12 TEST FAILURE VOTE 2 INPUT

13 TEST FAILURE VOTE 3 INPUT

14 TEST FAILURE VOTE 4 INPUT

0 = NO FAILURE

1 = FAILURE INDICATED (TEST)

These bits represents the RM hardware voter inputs while self testing. A bit set during normal system operation will result in one invalid FAILED VOTE input.

15 VOTER FAIL LATCH

0 = NO FAILURE

1 = FAILURE INDICATED

This bit represents the latched output of the RM hardware voter and in normal operation would indicate Computer Failure. The bit is also set during test when the hardware voter receives at least two of four inputs. During test, this voter output is inhibited before reaching the line driver.

16 TIMEOUT LATCH

0 = NO TIME OUT

1 = TIMEOUT REACHED

This bit indicates that the Watchdog Timer reached a timeout condition. If in normal operation, the bit would trigger the Computer Fail Latch. In test mode, the signal is inhibited prior to the Computer Fail Driver.

17 VOTER TERMINATION CONTROL LATCH

18 TIMER TERMINATION CONTROL LATCH

0 = RESET

1 = SET

These bits represent the output of the respective latches which are set and reset via PCI/PCO from CPU software. The termination output driver is permanently inhibited by the hardware.

19 IOP TRANSMISSION TERMINATION

0 = NORMAL

1 = TERMINATE

This bit indicates the status of the IOP transmission termination logic. In the AP-101S IOP, the logic does not disable discrete or databus outputs.

20 GO/NO-GO TIMER Bit 0 (MSB)
21 1
22 2

23	3	
24	4	
25	5	
26	6	
27	7	
28	8	
29	9	
30	10	
31	11	(LSB) = 0.768 msec

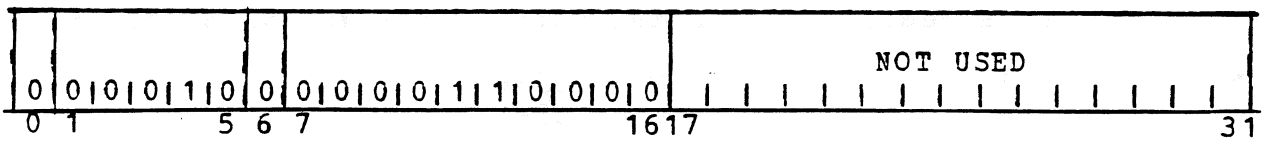
Bits 20-31 represent the configuration of the Watchdog (GO/NO-GO) Timer. The data is available in normal system operation and during self testing.

--

READ DISCRETE INPUT A

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
READ DISCRETE INPUT A (1-32)	01006000000	08180000	DF



DATA WORD

In the following discrete definitions, GPC N is the GPC ID of GPC self. This PCI provides means to transfer data from the IOP Discrete Input register (32 bits) to the CPU.

0 = D.I. RESET

1 = D.I. SET

NOTE: All the DI's are software readable. Some also configure hardware. Bit positions designated by * indicate those discrete inputs which are hardwired to IOP functions and must not be changed from the designated function due to internal or external wiring changes. The discrete inputs are implemented with differential receivers. The input pins are listed with each DI. The T represents the true input pin and C indicates the complement inputs.

BIT				
*0	HALT (DI-0)	T	J5-1	
		C	J5-2	

The setting of this bit indicates that the crew panel switch has been set to 'HALT'. Receipt of this DI causes the IOP to configure all processors to Halt thereby prohibiting IOP operation. The CPU is held in system reset by this discrete.

*1	STANDBY (DI-1)	T	J5-3
		C	J5-4

This bit is set from a crew panel switch.
The IOP response TBD.

*2 RUN (DI-2) T J5-5
C J5-6

The bit is set from a crew panel switch.
The IOP response is TBD.

*3 IPL (DI-3) T J5-7
C J5-14

Bit is set by crew panel switch. The IOP response is to perform the Initial Program Loading using data from the Mass Memory Unit as indicated below (DI-5, 6, 7, and 8).

*4 MM1 IPL (DI-4) T J5-8
C J5-9

The discrete is driven from the Orbiter systems network and when set indicates that MM1 is to be used as source for IPL.

*5 MM2 IPL (DI-5) T J5-10
C J5-11

Same as bit 4 above except applies to No. 2 Mass Memory Unit.

*6 MM1 READY (DI-6) T J5-12
C J5-13

This signal originates in No. 1 Mass Memory Unit and indicates, when set, that No. 1 MMU is available for use.

*7 MM2 READY (DI-7) T J5-15
C J5-16

Same as bit 6 above except applies to No. 2 MMU.

8 # (DI-8) T J5-17 GPC N+1 IS BFS RUN GPC
C J5-18

9 # (DI-9) T J5-19 GPC N+2 IS BFS RUN GPC
C J5-20

10 # (DI-10) T J5-21 GPC N+3 IS BFS RUN GPC
C J5-22

11 # (DI-11) T J5-23 GPC N+4 IS BFS RUN GPC
C J5-24

12		I/O TERMINATE A 10-13 (DI-12)	T	J5-25	
			C	J5-26	

Receipt of this DI causes the IOP to inhibit the MIA transmitters thereby prohibiting the output of data on Channels 10-13 (MIA's 10-13).

*13		INHIBIT CHANS. 14-17 AND 20-23 (DI-13)	T	J5-27	
			C	J5-28	

Also called I/O terminate B discrete.
Receipt of this DI causes the IOP to inhibit MIA transmitters 14-17 and 20-23.

14		(DI-14)	T	J5-29	SPARE
			C	J5-30	
15		HISAM DUMP	T	J3-66	SET BY ORBITER SWITCH TO INDICATE GPC DUMP REQUESTED
			C	J3-77	
16		(DI-16)	T	J3-89	SPARE
			C	J3-88	
17		(DI-17)	T	J3-69	SPARE
			C	J3-70	
18		(DI-18)	T	J3-79	SPARE
			C	J3-78	
19		(DI-19)	T	J3-81	SPARE
			C	J3-93	
20	#	(DI-20)	T	J3-80	GPC N+1 DISCRETE OUTPUT BIT 20 (SYNC 1)
			C	J3-91	
21	#	(DI-21)	T	J3-92	GPC N+2 DISCRETE OUTPUT BIT 20 (SYNC 1)
			C	J3-104	
22	#	(DI-22)	T	J3-114	GPC N+3 DISCRETE OUTPUT BIT 20 (SYNC 1)
			C	J3-103	
23	#	(DI-23)	T	J3-90	GPC N+4 DISCRETE OUTPUT BIT 20 (SYNC 1)
			C	J3-102	
24	#	(DI-24)	T	J3-113	GPC N+1 DISCRETE OUTPUT BIT 24 (SYNC 2)
			C	J3-121	
25	#	(DI-25)	T	J3-128	GPC N+2 DISCRETE OUTPUT BIT 24 (SYNC 2)
			C	J3-112	
26	#	(DI-26)	T	J3-111	GPC N+3 DISCRETE OUTPUT BIT 24 (SYNC 2)
			C	J3-101	

27	#	(DI-27)	T	J3-120 GPC N+4 DISCRETE OUTPUT BIT 24 (SYNC 2)
			C	J3-127
28	#	(DI-28)	T	J3-126 GPC N+1 DISCRETE OUTPUT BIT 28 (SYNC 3)
			C	J3-119
29	#	(DI-29)	T	J3-99 GPC N+2 DISCRETE OUTPUT BIT 28 (SYNC 3)
			C	J3-109
30	#	(DI-30)	T	J3-118 GPC N+3 DISCRETE OUTPUT BIT 28 (SYNC 3)
			C	J3-125
31	#	(DI-31)	T	J3-124 GPC N+4 DISCRETE OUTPUT BIT 28 (SYNC 3)
			C	J3-117

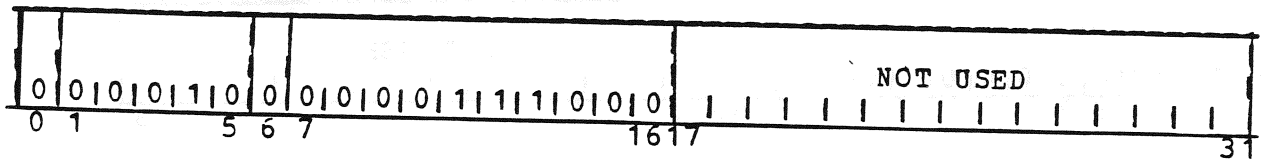
When N+ displacement > 5, subtract 5 from the result.

PCI FORMAT

READ DISCRETE INPUTS B

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
READ DISCRETE INPUTS (33-40)	01007000000	081C0000	RM



DATA WORD

This PCI provides the transfer of Discrete Inputs 33-40 to the CPU.

0 = D.I. RESET

1 = D.I. SET

BIT

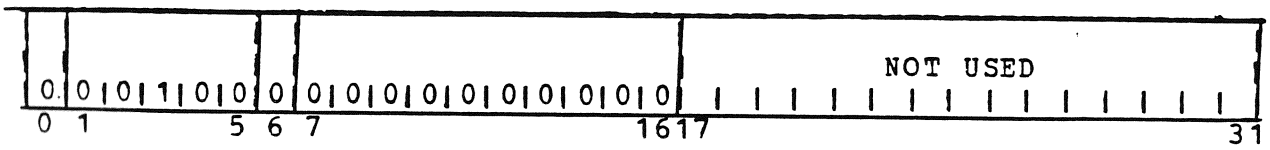
0	(DI-32)	T	J3-74	0-2 GPC SELFS ID
		C	J3-75	
1	(DI-33)	T	J3-116	
		C	J3-123	
2	(DI-34)	T	J3-122	
		C	J3-107	
3	(DI-35)	T	J3-96	BFS ENGAGE 1. SET BY ORBITER BFS CONTROLLER WHEN BFS ENGAGE PUSH-BUTTON IS DEPRESSED.
		C	J3-97	
4	(DI-36)	T	J3-106	BFS ENGAGE 2
		C	J3-115	
5	(DI-37)	T	J3-85	BFS ENGAGE 3
		C	J3-84	
6	(DI-38)	T	J3-95	6 AND 7. BFS CRT SELECT A AND B. INDICATES CURRENT SETTING OF ORBITER BFC CRT SELECT SWITCH, IF BFC CRT DISPLAY SWITCH IS ON.
		C	J3-105	
7	(DI-39)	T	J3-94	
		C	J3-83	
8	NOT USED.	UNDEFINED		
.		.		
.		.		
31		.		

PCI FORMAT

READ STATUS 1 (GO/NO-GO)

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
READ STATUS1(GO/NO-GO)	02000000000	10000000	DF



DATA WORD

The data word contains the GO/NO-GO status for each BCE and MSC as determined by IOP BITE.

0 = NO-GO (ERROR)
1 = GO

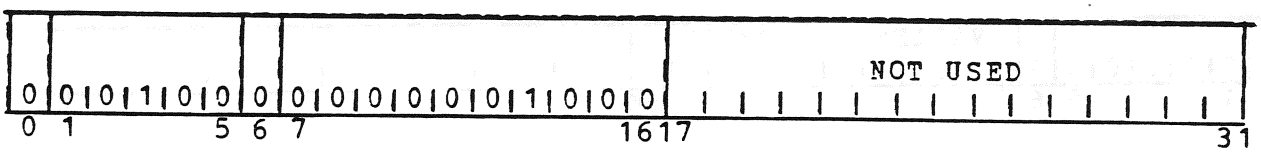
BIT	
0	MSC
1	BCE NO. 1
2	BCE NO. 2
.	.
.	.
24	24
25	NOT USED. UNDEFINED.
.	.
.	.
.	.
31	.

PCI FORMAT

READ STATUS 4 (BUSY/WAIT)

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
READ STATUS4 (BUSY/WAIT)	02001000000	10040000	DF



DATA WORD

The data word transfers the contents of Status Register 4 to the CPU. The action provides indication of the MSC and BCE's operating states.

0 = WAIT
1 = BUSY

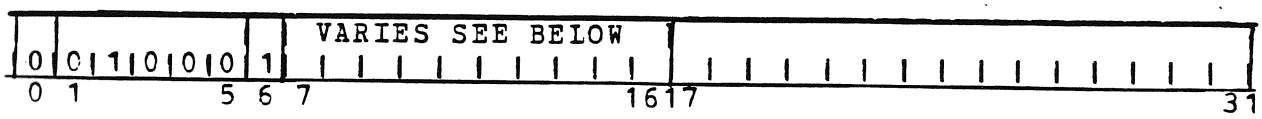
BIT	
0	MSC
1	BCE NO. 1
.	.
.	.
24	24
25	NOT USED. UNDEFINED.
.	.
.	.
.	.
31	.

PCI FORMAT

READ LOCAL STORE

COMMAND WORD

FUNCTION	OCTAL	HEX	DEVICE
READ LOCAL STORE			LS



The 10(10) bits (7 through 16) associated with the subsystem select field for this command word varies depending upon the word (location) loaded in Local Store. The command word provides access to the Local Store area associated with the MSC function and the 24 BCE functions of the IOP. The specific data stored in each location is discussed in the MSC and BCE Principals of Operation documents and involve that data required for initializing and operating the MSC and BCE's.

Bits 7 through 11

This 5-bit field is used to designate the MSC or BCE from which the data word must be transferred as shown below:

BITS 7	8	9	10	11	
0	0	0	0	0	MSC REGION
0	0	0	0	1	BCE NO. 1 REGION
0	0	0	1	0	BCE NO. 2 REGION
		.			.
		.			.
1	1	0	0	0	BCE NO. 24 REGION
1	1	0	0	1	SELF TEST PROCESSOR

Bits 12 and 13

This 2-bit field identifies the bank (A, B, or C) in Local Store for the above defined MSC or BCE area from which the data word must be transferred as indicated below.

BITS 12	13	
0	0	BANK A
0	1	BANK B
1	0	BANK C

Bits 14, 15 and 16

This 3-bit field is used to identify which word in a Bank is read. The resolution to a single word is achieved when this field is completed. Banks A and B use only Bits 15 and 16 to identify the four words required.

BITS 14	15	16	
0	0	0	LOCATION 0
0	0	1	LOCATION 1
	.		.
	.		.
1	1	1	LOCATION 7

DATA WORD

The single data word for this command contains the data word (18(10) bits) scaled to the LSB portion of the 32-bit data word as indicated below.

PCO DATA WORD BIT	LOCAL STORAGE WORD BIT	
0	UNDEFINED	
1	.	
.	.	
.	.	
13	.	
14	0	MSB
15	1	
.	.	
.	.	
31	17	LSB

Appendix II

Input/Output Processor (IOP) — Principles of Operation for Master Sequence Controller

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 MASTER SEQUENCE CONTROLLER

The Master Sequence Controller (MSC) is a micro programmed computer specifically tailored for I/O Management within the Space Shuttle General Purpose Computer (GPC). As such, it has extensive and programmable capabilities for monitoring and controlling the basic I/O operations performed by upwards to 24 Bus Control Elements (BCE's) which are implemented in the baseline GPC. These capabilities include setting up, scheduling, and initiating BCE programs, monitoring the status of BCE operations, and communicating overall completion of these operations to the CPU.

Table 1.1 summarizes the characteristics of the MSC.

TABLE 1.1

MSC CHARACTERISTICS

TYPE -	Single Accumulator I/O Management Computer
CONTROL STRUCTURE -	Microprogrammed
PROGRAMMABLE REGISTERS -	
	32 Bit ACCUMULATOR (ACC)
	18 Bit INDEX REGISTER (X)
	18 Bit PROGRAM COUNTER (PC)
PROGRAM VISIBLE REGISTERS	18 Bit STATUS REGISTER
	25 Bit PROGRAM EXCEPTION REGISTER
	25 Bit BUSY/WAIT REGISTER
	24 Bit BCE-MSC INDICATORS
	5 Bit FAIL DISCRETES
	12 Bit IOP PROGRAMMABLE INTERRUPT REGISTER
	18 Bit EXTERNAL CALL REGISTER
INSTRUCTION FORMATS -	16 Bit SHORT/32 Bit LONG
INSTRUCTION REPERTOIRE -	47 SHORT FORMAT/10 LONG FORMAT (NOT COUNTING ADDRESSING MODES)
ADDRESSING SPACE	131,072 32 Bit FULLWORDS/262,144 16 Bit HALFWORDS
ADDRESSING MODES	IMMEDIATE, ABSOLUTE, INDEXED, PC RELATIVE
DATA FORMAT	SIGNED, TWO'S COMPLEMENT INTEGER
SPECIAL FEATURES	INITIALIZE AND MONITOR BCEs. RESPOND TO CPU REQUESTS TO CHANGE PROGRAM (EXTERNAL CALL).

1.1 ADDRESSING, DATA AND INSTRUCTION FORMATS

The MSC may directly address up to 262,144 16-bit halfwords or 131,072 32-bit fullwords. To achieve this, all main memory addresses computed by the MSC are represented as 18-bit absolute numbers, as pictured in Figure 1.1. The upper 17-bits (bits 0 through 16) represent the fullword location, and the lowest bit (bit 17) the halfword portion of the addressed fullword. A 0 in this lower bit refers to bits 0-15 of the 32-bit fullword; a 1 refers to bits 16-31. When used as a fullword address, bit 17 is ignored. Thus, H'276' and H'277' refer to the same fullword.

All data referenced as numbers by the MSC are treated as signed two's complement binary integers.

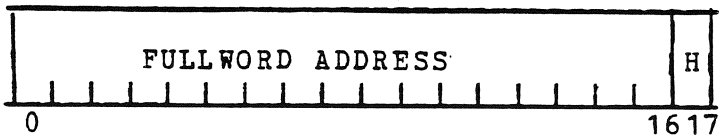
There are three basic instruction formats used by the MSC and they are depicted in Figure 1.2.

Short format 1 is used primarily by instructions dealing with memory and the MSC accumulator. It has the following fields:

Field	Field Description
OP	This 4-bit field defines the basic operation to be performed by the MSC
I	This bit serves either as an opcode extension or as an index mode specification in address generation
DISP	This 11-bit field serves either as immediate data or as a PC relative address displacement.

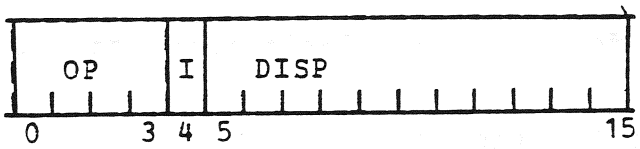
Short format 2 is used by the register operation, register immediate, repeat, and conditional branching instruction classes. It has the following fields:

Field	Field Description
OP	This 4-bit field defines the basic operation class to be performed by the MSC
I	This bit serves either as an opcode extension or as an index mode specification in address generation
OPX	This field is an extension of OP
DISP	This field serves as either immediate data or as a PC relative address displacement.



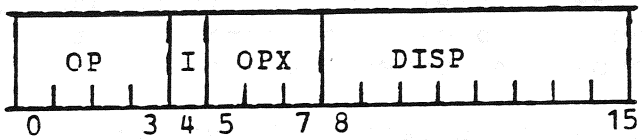
H = Halfword Selector

Figure 1.1. MSC-Computed Main Memory Addresses.



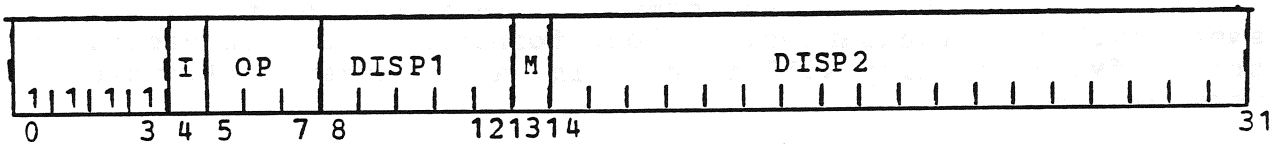
I = Index Specification or OPX

(a) Short Format 1



I = Index Specification or OPX

(b) Short Format 2



(c) Long Format

Figure 1.2. Basic MSC Instruction Formats.

Format 3 is used for all long 32-bit instructions, and has the following fields:

Field	Field Description
OP	This 3-bit field defines the basic operation to be performed by the MSC
DISP1	This 5-bit field defines such things as BCE numbers, short displacements, etc.
DISP2	This 18-bit field is used in conjunction with I and M to generate both 18 bit main memory addresses and immediate data.
I,M	These two bits define the way that DISP2 is to be used in formation of values used by the instruction for main memory addresses and immediate data.

Table 1.2 summarizes the use of I,M and DISP2.

Note that all long format instructions must be located on even fullword boundaries. A long format instruction beginning on an odd halfword boundary will result in a "Boundary Alignment" error termination.

TABLE 1.2

MSC LONG FORMAT MODES

I	M	Effective Value	Mode
0	0	DISP2*	IMMEDIATE
1	0	X+DISP2*	INDEXED,IMMEDIATE
0	1	(DISP2)	DIRECT
1	1	(X+DISP2)	INDEXED,DIRECT

Parenthesis used around a quantity indicate that it is to be treated as an 18-bit main memory fullword address (least-significant bit ignored) and that the effective value is found in fullword.

NOTES:

* When the effective value is used as immediate data, the most-significant bit is sign-extended 14 places to the left

1.2 MSC REGISTERS

1.2.1 MSC PROGRAMMABLE REGISTER

The MSC contains 3 registers under direct program control. They are:

ACC - A 32-bit accumulator

X - An 18-bit index register

PC - An 18-bit Program Counter

The ACC is a 32-bit register capable of accumulating a fullword of data from memory. MSC instructions are available to load, modify, test and store this register. The ACC is also used to contain bit masks for status and polling applications.

Data representation of numbers held by the ACC is a signed, two's complement, 32-bit integer, with the sign bit in bit 0 and the binary point to the right of bit 31 (Figure 1.3).

The Index register may be used both in generating 18-bit main memory addresses and as a holding register for signed two's complement 18-bit integers. When treated as an address, it is formatted as in Figure 1.1. When treated as a number, it is formatted as in Figure 1.4.

The Program Counter is an 18-bit register that indicates the main memory halfword or fullword location of the MSC instruction presently being executed. It is formatted as a standard 18-bit address as in Figure 1.1.

1.2.2 MSC STATUS REGISTER

The status of the MSC is kept in the following three registers:

1) MSC Busy/Wait Bit - Bit 0 of the IOP Busy/Wait Register. A 1 in this bit indicates that the MSC is currently executing a program located in main memory. A 0 indicates it is not busy. This bit is set via a PCO from the CPU to activate the MSC, and reset by the MSC upon completion of a program.

2) MSC Program Exception Bit - bit 0 of the IOP Program Exception Register A 0 indicates that the MSC has encountered some problem in the execution of an MSC program. A 1 indicates that no problems were encountered. This bit is maintained by the MSC microcode.

3) MSC Status Register - An 18-bit register found in register C7 in the MSC segment of Local Store (See Figure 1.5). This register contains copies of the MSC Busy/Wait bit and the MSC Program Exception bit. If the latter bit is set, this register also indicates the exact cause of the exception, as indicated in Table 1.3. It is set automatically upon the detection of an exception or as part of MSC execution of an @REC instruction, and may be read by the MSC with a @LMS instruction.

PROGRAMMING NOTE

The macro programmer can:

1) Reset STAT1 (Hardware register) via a PCO command.

2) Clear or change the MSC status register via a PCO command.

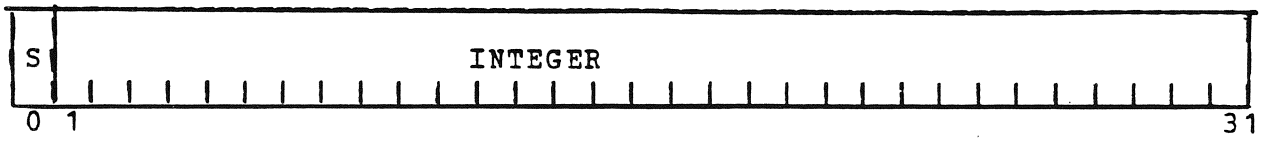
3) Reload the MSC status register using an @REC instruction.

Execution of any of the three cases above will alter the MSC Status Register. To guard against their possible interference with system operation the following principles should be adhered to:

1) While the MSC is busy do not attempt to alter the STAT1 or STAT4 Registers by using PCO commands.

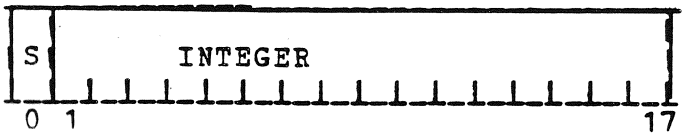
2) While the MSC is busy do not attempt to change the MSC Status Register via PCO's.

3) Be aware that when the MSC returns from an external call (@REC) the MSC Status Register is loaded.



S = Sign

Figure 1.3. Number Representation in MSC Accumulator



S = Sign

Figure 1.4. Number Representation in MSC Index Register.

TABLE 1.3

MSC STATUS REGISTER

RESERVED	STF		SIO ERR	LBP ERR	LBB ERR	ALIGN	ILL OP	ERR	BUSY/ WAIT
0	89	10	11	12	13	14	15	16	17

Bit 17 STAT4 bit 0 - The Busy/Wait bit for the MSC. It will always be 1 while instructions are being executed.

Bit 16 STAT1 bit 0 - The program exception bit for the MSC. This bit will be set to one only if the execution of some previous instructions resulted in an error of some kind. Bits 15 through 8 catalog the exact error. This bit is the inverse of the MSC program exception bit.

Bit 15* Illegal opcode. If set, this bit indicates that an illegal opcode was encountered at some point in the past.

Bit 14* Boundary alignment error. If set, this bit indicates that a long format instruction was encountered at an odd halfword boundary.

Bit 13 LBB error. The BCE that was specified by a previous @LBB instruction was busy or halted at the time the instruction was executed, and consequently its Base Register was not loaded.

Bit 12 LBP error. The BCE that was specified by a previous @LBP was busy or halted at the time the instruction was executed, and consequently its Program Counter was not loaded.

Bit 11 SIO error. A previous execution of an @SIO instruction tried to set at least one BCE to busy that was already busy.

Bit 10,9 Reserved.

Bit 8 Self test failure. The execution of a previous MSC self test instruction detected a fault in the MSC.

*Note that the only way this bit could be set when an @LMS is executed is if this error condition was detected

in the past, and the CPU restarted the MSC without clearing the status word first.

1.2.3 OTHER MSC VISIBLE REGISTERS

In addition to the register described above, there are several other IOP registers that are accessible by an MSC program. These registers include the:

- Program Exception Register (STAT1 Register) - a 25-bit register containing one bit per processor. A 0 in bit position i indicates that processor i (0=MSC,1-24=BCE) has encountered some sort of exceptional condition. An exact description of the cause of the exception may be found in processor(i)'s status register.

- Busy/Wait Register - a 25-bit register containing one bit per processor. A 1 in bit position i indicates that processor(i) is presently executing a program. A 0 indicates that the processor is inactive.

- BCE-MSC Indicator Register - a 24-bit register containing one bit per BCE. BCE(i) may set or reset bit (i) under BCE program control (See #SIB, #RIB). Various BCE errors will also set this bit to 1. The MSC @RBI instruction may reset any of these bits. There is no hardware defined meaning for these values. They may be used freely to indicate any programmer specified convention.

- Fail Discrete Register - these discretes are grouped as a single register, and are used in the redundancy management process to indicate failed GPC units.

- IOP Programmable Interrupt Register - this 12-bit register may be set by the MSC to specify any combination of 12 CPU interrupts. The exact meaning of each interrupt is determined by convention between the MSC program and the CPU interrupt handling program.

- External Call Register - this 18-bit register may be set by the CPU to point to a program that it wishes the MSC to execute. The MSC resets the register to zero when it has started the program. The Sample For External Call (@SEC) instruction and section 4.0 should be referenced for additional detail.

The Program Exception register may only be read. The Busy/Wait and Fail discrete registers may be both read and set. The BCE-MSC Indicator register may be read or reset. The IOP Programmable Interrupt Register may only be set. The External Call register may be set by the CPU and reset by the MSC.

1.3 MSC IMPLEMENTATION

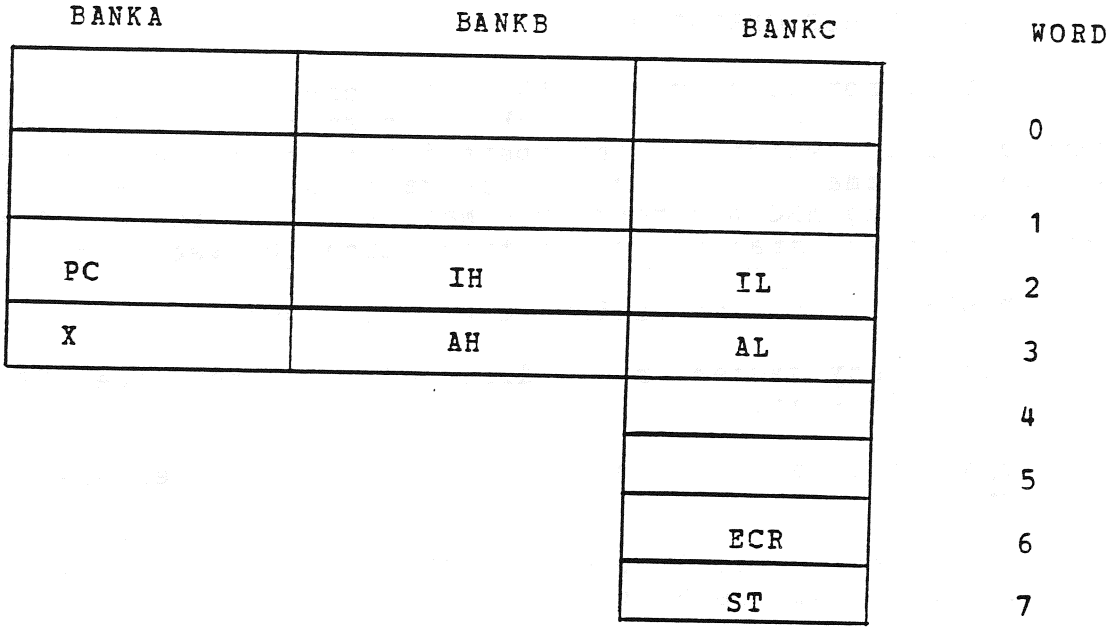
The MSC as implemented within the IOP consists of:

- Segment 0 of Banks A,B and C of Local Store. This contains the ACC, X, PC, Status Register, and all other working registers needed to implement the MSC instruction set.
- Bit 0 of the IOP Busy/Wait Register.
- Bit 0 of the IOP Program Exception Register.
- Bit 0 of the Halt Register
- Microprogram Counter 0. This contains the ROS address of the next micro-instruction to be executed as part of the execution of the present MSC instruction.

As described in the AP-101S Design Workbook (IBM No. 85-C67-005), the operation of the MSC is time-shared with operations for the BCE's. This time-sharing is performed at the micro-instruction level, where the IOP executes one MSC micro-instruction from ROS (The one indicated by the MSC micro-program counter) then a micro-instruction for several of the BCE's, followed by an MSC micro-instruction, etc. MSC micro-instructions normally occur at 2 microsecond intervals with the exception of every eighth MSC micro-instruction which occurs 2.5 microseconds after the proceding instruction. The only MSC-related operation carried on during non-MSC microcycles are memory operations, previously requested by MSC, and CPU-directed PCI/O.

The general markup of the Local Store segment dedicated to the MSC is shown in Figure 1.5. All unlabelled words are reserved as temporary working registers during the execution of a single MSC instruction. After each instruction, however, only the labelled contents are assumed to be properly maintained.

The PCI/O operations of reading or writing to any of these locations may occur at any time, although writing into MSC local store while the MSC is busy will generally result in unpredictable MSC behavior.



PC - MSC Program Counter
X - MSC Index Register
IH - Upper 16 bits of present MSC instructions (Right justified with Bits 0 and 1 ignored)
IL - Lower 18 bits of last instruction word that was read from memory. (Right justified with Bits 0 and 1 ignored)
AH - Bits 0-15 of MSC accumulator (right justified into local store. Bits 0,1 ignored).
AL - Bits 16-31 of MSC accumulator (right justified as for AH)
ECR - External Call Register
ST - Bits 0-17 of MSC Status Register (Right justified)

Figure 1.5. MSC Local Store Useage

2.0 GENERAL MSC OPERATION

During normal operation the MSC can be in one of three states: Halt, Wait and Busy. In the Halt state the MSC is physically restrained from performing any operations; in the Wait state the MSC is awaiting a command to execute a program; and in the Busy state the MSC is executing MSC programs from main store. Figure 2.1 and Table 2.1 summarize these states and the transitions between them.

Typical state transitions are as follows:

1) During any system or CPU-directed MSC reset, the MSC is in the Halt state.

2) Upon release from the Halt state, the MSC enters the Wait state

3) A sequence of PCO's from the CPU initializes the MSC, and places it in the Busy state.

4) In the Busy state, the MSC is executing a program located in main memory. It exits the Busy state only upon execution of a Wait instruction, detection of an invalid instruction, or some MSC reset signal. In all but the latter case, the MSC re-enters the Wait state; in the latter case it is forced to the Halt state.

The two bits that indicate the current state of the MSC are its Halt bit (Bit 0 of the IOP Halt register) and its Busy/Wait bit (Bit 0 of the IOP Busy/Wait register). In addition, the MSC Program Exception bit indicates if the MSC has found an error during its last period of time in the Busy state. If it has found an error, the MSC Status Register contains a record of the error.

The following sections describe each state in detail.

2.1 HALT STATE

The primary purpose of the Halt state is two-fold:

1) Allow the external world to reset MSC operation to a known condition.

2) Upon the detection of very serious IOP faults (such as failure in the microstore) to isolate the MSC and prevent it from performing potentially erroneous operations.

Entry and exit from the Halt state are controlled by the value of a single "MSC Halt" status bit which is part of the IOP Halt Register. As long as this bit is set, the MSC microprogram counter is

forced to point to a micro-instruction that performs no operation other than clear the MSC Busy/Wait bit. The Halt bit may be set at any time, and effectively terminates anything that the MSC is doing.

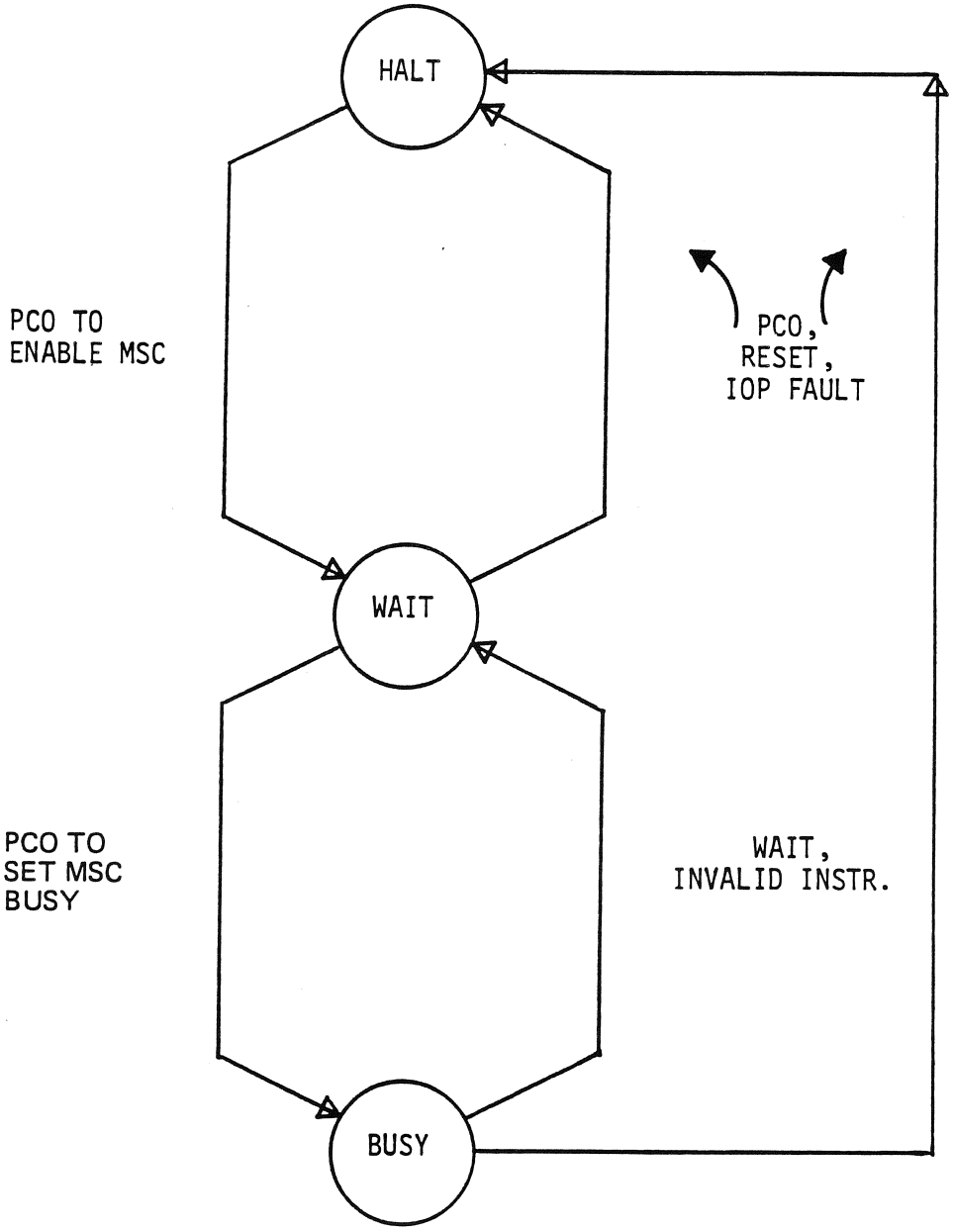


Figure 2.1 MSC States

TABLE 2.1

MSC STATES

STATE	ENTERED FROM STATE/WHEN		ACTION TAKEN IN STATE	STATE INDICATORS HALT	BUSY/WAIT
HALT	ANY STATE	- MASTER RESET	HARDWARE FORCED MICRO BRANCH TO DO NOTHING	0	0
		- FAIL LATCH SET FOR THIS GPC	MICRO INSTRUCTION.		
		- PCO SPECIFYING MSC HALT	BUSY/WAIT, STATUS, EXTERNAL CALL REGISTER, AND PROGRAM EXCEPTION BIT CLEARED UPON EXIT.		
WAIT	HALT	- PCO TO CLEAR BIT	MONITOR BUSY/WAIT BIT FOR TRANSITION TO BUSY	1	0
	BUSY	- EXECUTION OF "WAIT" INSTR - ILLEGAL OPCODE - BOUNDARY ALIGNMENT ERROR			
BUSY	WAIT	- PCO SETS BUSY/WAIT BIT	EXECUTE MSC PROGRAM SET PROGRAM EXCEPTION BIT AND STATUS WORD UPON DETECTION OF ANY PROBLEM	1	1

The signals that set the MSC Halt bit to 0 (halt MSC) include:

1) BCE/MSC disable discretes or discrete input 0.

2) Power-up/down signal.

3) IOP detected serious errors such as ROM parity error or timing failure.

4) A "Master Reset" PCO from the CPU.

5) A "Halt Processor" PCO from the CPU with a 1 in bit position 0.

The first four signal classes also halt all BCE's.

Exit from this state to the Wait state occurs only when the following occurs:

1) "BCE Disable" discrete is reset.

2) Power-up sequence is complete.

and a CPU "Enable Processors" PCO with bit 0=1 is present.

Upon any exit from the Halt state, the MSC Program Exception bit is set to 1 and the MSC Status Register is cleared to 0 to indicate that no errors exist. The MSC also clears the External Call register at this time.

2.2 WAIT STATE

In the Wait state, the MSC is prepared to be told to perform an MSC program. This Wait loop is implemented by a micro-routine that monitors the status of the MSC Busy/Wait bit. The MSC remains in the Wait state as long as this bit is reset to the "wait" value. Setting this bit to a "busy" condition causes the MSC to transit to the Busy state. This transition consists of using the present value of the MSC's Program Counter as the starting address of an MSC program in main store.

Entry to the Wait state occurs either upon exit from the Halt state, as described above, or by a transition from the Busy state. The MSC performs this latter transition when any of the following occur:

1) A "Wait" instruction is executed.

2) An instruction with an illegal opcode is encountered.

3) A valid long format instruction is found starting on an odd halfword boundary.

In the latter two cases, the following actions are performed prior to entering the Wait state:

1) The MSC Program Exception bit is set to 0.

2) The MSC Status Register is set to indicate the exact cause of the error.

3) The MSC Program Counter is left pointing to the offending instruction.

The Program Exception bit and Status Registers may also be set when other errors are detected while the MSC is in the Busy state; in these cases, however, the MSC continues execution of the program.

Exit from the Wait state is normally to the Busy state. This is done by the CPU via a sequence of PCO's; these typically include:

1) A "Load Local store" PCO to clear the MSC Status Register (if set).

2) A "Reset Program Exception Register" PCO with a 1 in bit 0 (again used only when MSC has indicated an error).

3) A "Load Local Store" PCO to point to the start of the desired MSC program in main store.

4) A "Start MSC" PCO to set the MSC Busy/Wait bit.

Note that the first two are needed only to recover from an MSC error, and the third is needed only if the MSC program counter must be reset. They may not always be necessary, since the Wait instruction (@WAT), when executed, leaves the PC pointing to the next sequential instruction, which in turn may be programmed as a simple jump back to the beginning of the MSC program. In this case, the CPU needs only to issue the "Start MSC" PCO to restart the MSC program.

2.3 BUSY STATE

In the Busy state, the MSC is in the process of executing a program out of main store. A value of 1 in the 0 bit of the IOP Busy/Wait Register indicates this condition.

The Busy state may be entered only from the Wait state, as described in the previous section. When the transaction occurs, the MSC uses whatever value is presently in its Program Counter to fetch the first instruction from memory and start executing it. The MSC continues executing instructions until either a Wait instruction or some kind of invalid instruction is encountered. In either case, the MSC transitions back to the Wait state, again as described in the previous section.

While the MSC is in the Busy state, the CPU may execute any PCI without problem. However, all PCO's that write into MSC Local store should be carefully controlled since the MSC is not aware that this action has occurred, and the resulting MSC program execution may be unpredictable.

2.4 ERROR MODES

During execution of an MSC program, the MSC may encounter one or more errors. These errors may be due to either faulty programming or actual hardware faults. Many can be detected by the IOP microprograms that support the MSC functions, and are reflected by the setting of various status bits in the MSC Status Register and the MSC Program Exception bit. If the error is serious enough, the MSC terminates the instruction and enters the Wait state.

Table 2.2 summarizes all known errors that can affect MSC operation; how they are detected; and the actions taken upon their detection.

TABLE 2.2

IOP ERRORS AFFECTING MSC OPERATION

ERROR	DETECTED BY	ACTION
Illegal MSC Instruction Opcode	Microcode	Set MSC Program Exception Bit, Set Status bits 15, 16, Enter Wait State
32 BIT MSC Instruction Starting on Odd Boundary	Microcode	Set MSC Program Exception Bit, Set Status Bits 14, 16. Enter Wait State
Attempt to Start Busy BCE, or @SIO with ACC[0]=1.	Microcode	Set MSC Program Exception Bit. Set Status Bits 11, 16. Continue Instruction.
Attempt to Load PC of Busy BCE, or @LBP with BCE number=0.	Microcode	Set MSC Program Exception Bit. Set Status Bits 12, 16. Continue Instruction.
Attempt to Load Base of Busy or halted BCE, or @LBB with BCE number=0.	Microcode	Set MSC Program Exception Bit. Set Status Bits 13, 16. Continue Instruction.
DMA Parity Error During Instruction Read	IOP Hardware	Make up all 0's instruction and give to MSC. MSC will terminate with "Illegal Opcode" Interrupt CPU.
DMA Parity Error During Data Read, DMA timeout, Queue Overflow.	IOP Hardware	No data is given to MSC. MSC loops indefinitely. Interrupt CPU.
IOP Faults IOP Microstore Parity Error, Clock Failure, watch-dog timer time out, or fail latch set.	IOP Hardware	MSC Halt bit set to 0, and MSC enters halt state. Interrupt CPU.

Error	Detected by	Action
Self Test detected MSC fault	MSC Self-Test Instruction	Set MSC program exception bit. Set status bits 8,16.
Self-Test generated parity error.	IOP Hardware	Force all processors to the Halt state, set Busy/Wait bits of all processors to Wait.
IOP Data Flow Parity Error	Hardware	All BCE's and MSC are Halted, CPU is interrupted, Transmitter and receiver enables for all BCE's are disabled, and the discrete outputs are reset.

NOTE: The MSC Program Exception bit registers an error when it is set to zero.

3.0 MSC INSTRUCTIONS

The following subsections include descriptions of the instructions presently supported by the MSC. The format for the description of each instruction is as follows:

- The general name of each instruction appears in the upper left of the first page describing that instruction.
- The assembler abbreviation appears in the upper right hand corner.
- The format of the instruction, including binary opcode assignments and field designations.
- A table (where appropriate) relating addressing mode bits to their effect on parameters used in the instruction execution, and how these addressing modes are signalled to the assembler.
- A textual description of the instruction and its uses.

These instructions are grouped into several classes with a short prologue at the beginning of each class. These classes include:

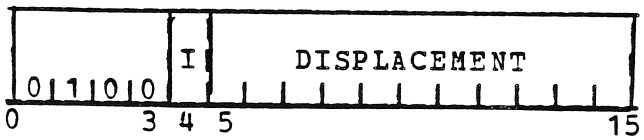
3.1 ACCUMULATOR/MEMORY INSTRUCTIONS

The Accumulator/Memory Instruction set allows the 32-bit MSC Accumulator (ACC) to be modified by arguments found in memory, and for copies of the ACC to be stored in memory. Most of these instructions are short 16-bit formats of the form of Figure 1.2(a), and allow either PC relative or indexed PC relative addressing. Long format instructions of this class are as shown in Figure 1.2(c), and allow absolute addressing of memory.

LOAD ACCUMULATOR

@L

FORMAT:



I	Effective Address ("EA")	INSTR Format
0	PC + Displacement	@L ADDRESS
1	PC + Displacement + X	@L ADDRESS (1)

NOTE:

1) PC refers to the updated program counter value - i.e., the address of the next instruction.

2) Bit 5 is the sign of the two's complement displacement field. The range of the displacement is -1024 to +1023 halfwords.

3) X is the index register.

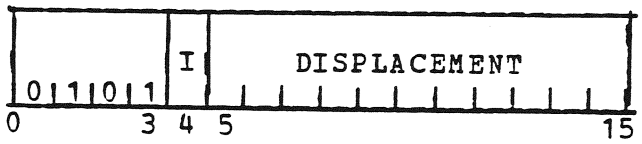
DESCRIPTION

The fullword operand addressed by EA is loaded into the accumulator. The program counter is incremented by one.

ADD TO ACCUMULATOR

@A

FORMAT:



I	Effective Address ("EA")	INSTR Format
0	PC + Displacement	@A ADDRESS
1	PC + Displacement + X	@A ADDRESS(1)

NOTE:

1) PC refers to the updated program counter value - i.e., the address of the next instruction.

2) Bit 5 is the sign of the two's complement displacement field. The range of the displacement is -1024 to +1023 halfwords.

3) X is the index register.

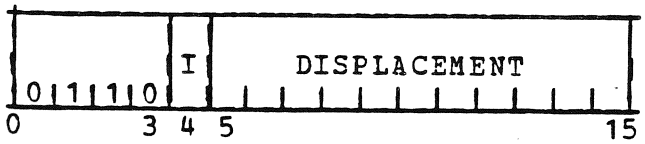
DESCRIPTION

The fullword operand addressed by EA is added to the accumulator. The program counter is incremented by one.

AND TO ACCUMULATOR

@N

FORMAT:



I	Effective Address ("EA")	INSTR Format	
0	PC + Displacement	@N	ADDRESS
1	PC + Displacement + X	@N	ADDRESS(1)

NOTE:

1) PC refers to the updated program counter value - i.e., the address of the next instruction.

2) Bit 5 is the sign of the two's complement displacement field. The range of the displacement is -1024 to +1023 halfwords.

3) X is the index register.

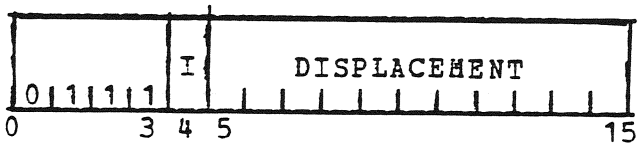
DESCRIPTION

The fullword operand addressed by EA is logically anded to the accumulator. The program counter is incremented by one.

EXCLUSIVE OR

@X

FORMAT:



I	Effective Address ("EA")	INSTR Format	
0	PC + Displacement	@X	ADDRESS
1	PC + Displacement + X	@X	ADDRESS (1)

NOTE:

1) PC refers to the updated program counter value - i.e., the address of the next instruction.

2) Bit 5 is the sign of the two's complement displacement field. The range of the displacement is -1024 to +1023 halfwords.

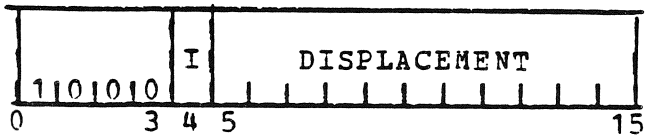
3) X is the index register.

DESCRIPTION

The fullword operand addressed by EA is Exclusive Or'ed with the accumulator. The program counter is incremented by one.

STORE ACCUMULATOR @ST

FORMAT:



I	Effective Address ("EA")	INSTR Format
0	PC + Displacement	@ST ADDRESS
1	PC + Displacement + X	@ST ADDRESS (1)

NOTE:

1) PC refers to the updated program counter value - i.e., the address of the next instruction.

2) Bit 5 is the sign of the two's complement displacement field. The range of the displacement is -1024 to +1023 halfwords.

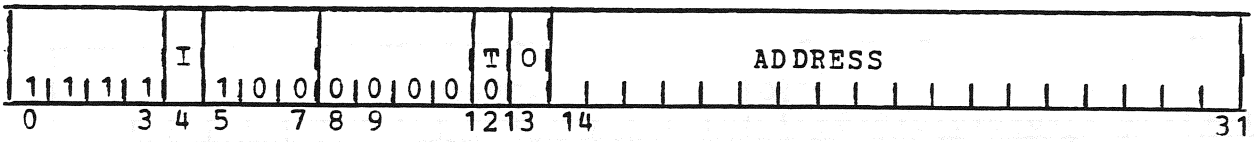
3) X is the index register.

DESCRIPTION

A copy of the contents of the accumulator is stored in the fullword specified by EA. The accumulator is unchanged. The program counter is incremented by 1.

LOAD ACC WITH FULLWORD @LF

FORMAT:



I	EFFECTIVE ADDRESS (EA)	INSTR Format	
0	Address	@LF	ADDRESS
1	Address+X	@LF	ADDRESS(1)

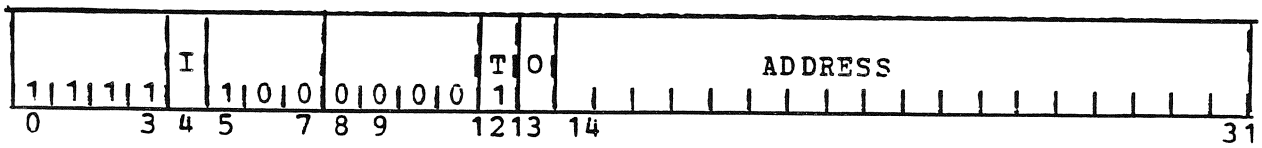
DESCRIPTION

This instruction loads the MSC Accumulator with a fullword from the memory location whose address is defined above. In a fullword load (T=0), the least-significant bit of the effective address is ignored.

LOAD ACC WITH HALFWORD

@LH

FORMAT:



I	EFFECTIVE ADDRESS (EA)	INSTR Format	
0	Address	@LH	ADDRESS
1	Address+X	@LH	ADDRESS(1)

DESCRIPTION

This instruction loads the MSC Accumulator with a halfword from the memory location whose address is defined above. In a halfword load (T=1), the effective address is used as a complete halfword address. The addressed halfword is placed in the lower 16 bits of the MSC Accumulator, with the upper 16 bits sign-extended.

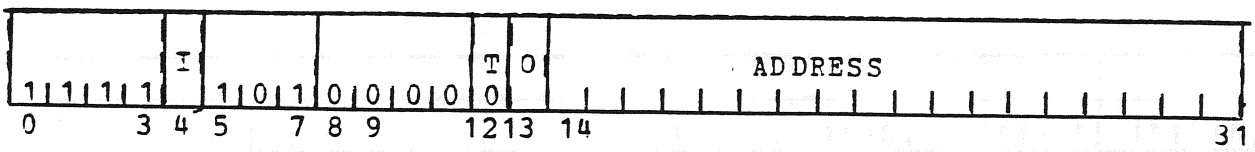
PROGRAMMING NOTE:

In the above instruction format, any non-zero value in bits 8 thru 12 is treated as a valid "LOAD HALFWORD" Opcode.

STORE ACC FULLWORD

@STF

FORMAT:

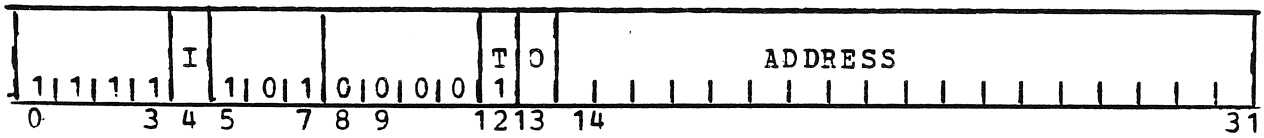


I	EFFECTIVE VALUE E.V.	INSTR Format	
0	Address	@STF	ADDRESS
1	Address+X	@STF	ADDRESS (1)

DESCRIPTION

This instruction stores the MSC Accumulator into the memory location whose address is defined above. In a fullword store (T=0), the least-significant bit of the effective address is ignored and the entire MSC accumulator is stored in the addressed fullword.

FORMAT:



I	EFFECTIVE VALUE E.V.	INSTR Format	
0	Address	@STH	ADDRESS
1	Address+X	@STH	ADDRESS(1)

DESCRIPTION

This instruction stores the MSC accumulator into a memory location whose address is defined above. In a halfword store, the effective address is used as a complete halfword address. The lower 16 bits of the MSC Accumulator are stored in this halfword location.

PROGRAMMING NOTE:

In the above instruction format, any non-zero value in bits 8 thru 12 is treated as a valid "STORE HALFWORD" Opcode.

3.2 BRANCHING INSTRUCTIONS

MSC instructions are normally executed in sequential order. A branch instruction allows a departure from this sequential operation. The branches included in the MSC instruction set provide mechanisms for conditional branches, unconditional branches and subroutine calls, and returns from external calls (See Appendix A).

The conditional branches are short format instructions of the form of Figure 1.2(b). This format provides an 8-bit signed displacement that is added to the PC if the specified condition is true.

The unconditional branches and subroutine calls are 32-bit long format, as shown in Figure 1.2(c). These instructions provide an absolute 18-bit address that is loaded into the MSC PC when the instruction is executed. As options, these addresses may be indexed or used as pointers to fullwords in memory containing the addresses. The latter option allows direct return from subroutines.

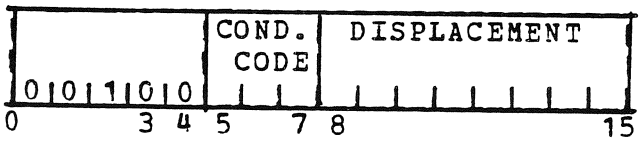
The instruction to return from an external call loads the MSC Accumulator, Index, and Status registers and in addition, causes the MSC to branch to a new location.

The result of executing any branch instruction places a new 18-bit address in the MSC PC. This address points to the next instruction to be executed, and may point to either an even or odd halfword boundary.

BRANCH ON ACCUMULATOR

@BC

FORMAT:



Instruction Format

@BC CONDITION, ADDRESS

DESCRIPTION

This instruction selects the accumulator register to be tested. If any of the three conditions specified by the condition code field are true, the 8-bit two's complemented signed displacement is added to the updated MSC program counter. This displacement has a range from -128 to +127 halfword locations. Branches may be made to even or odd halfword locations.

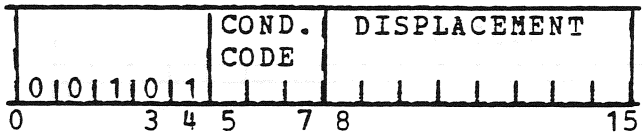
The three bits of the condition code field specify the following tests:

Bit 5 - Register = 0
Bit 6 - Register < 0
Bit 7 - Register > 0

The following table summarizes all 8 combinations and the resulting tests.

COND. CODE	TEST	EXTENDED MNEMONIC
0 - 0 0 0	FALSE - DO NOT BRANCH	----------
1 - 0 0 1	ACC > 0	@BP
2 - 0 1 0	ACC < 0	@BN
3 - 0 1 1	ACC ≠ 0	@BNZ
4 - 1 0 0	ACC = 0	@BZ
5 - 1 0 1	ACC ≥ 0	@BNN
6 - 1 1 0	ACC ≤ 0	@BNP
7 - 1 1 1	TRUE - ALWAYS BRANCH	@B

FORMAT:



Instruction Format

@BXC CONDITION, ADDRESS

DESCRIPTION

This instruction selects the Index Register (sign extended to 32 bits) to be tested. If any of the three conditions specified by the condition code field are true in regard to the index register, the 8-bit two's complement signed displacement is added to the updated MSC program counter. The displacement has a range from -128 to a +127 halfword locations. Branches may be made to even or odd halfword locations.

The three bits of the condition code field specify the following tests:

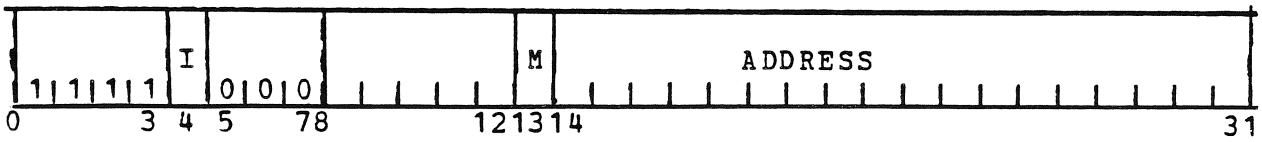
Bit 5 - Register = 0
Bit 6 - Register < 0
Bit 7 - Register > 0

The following table summarizes all 8 combinations and the resulting tests.

COND.CODE	TEST	EXTENDED MNEMONIC
0 - 000	FALSE - DO NOT BRANCH	
1 - 001	X > 0	@BXP
2 - 010	X < 0	@BXN
3 - 011	X ≠ 0	@BXNZ
4 - 100	X = 0	@BXZ
5 - 101	X ≥ 0	@BXNN
6 - 110	X ≤ 0	@BXNP
7 - 111	TRUE - ALWAYS BRANCH	

BRANCH UNCONDITIONAL @BU

FORMAT:



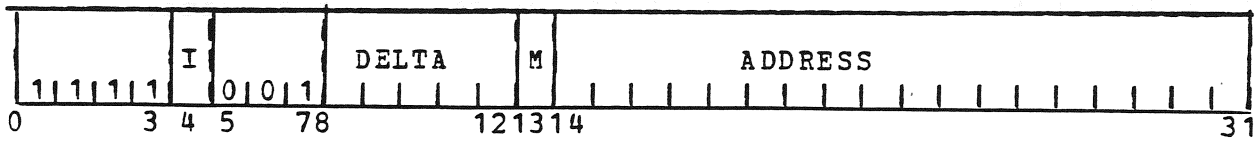
I	M	EFFECTIVE VALUE E.V.	INSTR Format	
0	0	Address	@BU	ADDRESS
0	1	(Address)	@BU@	ADDRESS
1	0	Address + X	@BU	ADDRESS(1)
1	1	(Address + X)	@BU@	ADDRESS(1)

DESCRIPTION

The MSC program counter is loaded with the lower 18 bits of the effective value. A branch to either an even or odd halfword is permitted.

SUBROUTINE CALL

FORMAT:



I	M	EFFECTIVE VALUE E.V.	INSTR Format
0	0	Address	@CALL DELTA,ADDRESS
0	1	(Address)	@CALL@ DELTA,ADDRESS
1	0	Address + X	@CALL DELTA,ADDRESS(1)
1	1	(Address + X)	@CALL@ DELTA,ADDRESS(1)

DESCRIPTION

This instruction implements a subroutine call. The current value of the MSC program counter plus the five bit positive integer delta (bits 8-12) is stored in the fullword specified by the lower 18 bits of E.V. This quantity is zero padded on the left to fill the entire fullword. The MSC program counter is then loaded with the sum of the lower 18 bits of E.V. and two.

PROGRAMMING NOTE

This instruction is typically used to call a subroutine. A typical subroutine sequence is:

```
          .
          .
          .
          @CALL 4,SUB
          +                         ARGUMENT
          .                         FIRST INSTRUCTION AFTER RETURN
          .
          .
SUB       +0                        USED FOR RETURN ADDRESS
          .
          .                         SUBROUTINE BODY
          .
          @BU@  SUB                 SUBROUTINE RETURN
```

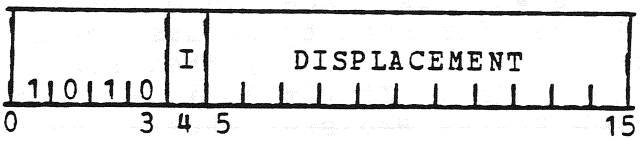
The effective value used in the subroutine call may be either even or odd. With an odd address, the least-significant bit is ignored when the return address is stored, but is considered when the

branch is taken. Thus a @CALL 2,101 will cause the return address to be stored in fullword 100, but a branch to 103 will be taken.

As with any instruction that writes information into memory, the affected memory location must not be storage protected.

RETURN FROM EXTERNAL CALL

FORMAT



I	Effective Address ("EA") 2	INSTR Format
0	PC1 + Displacement	@REC Address
1	PC1 + Displacement + X	@REC Address (1)

NOTE:

1) PC is updated PC, i.e. address of instruction following @REC.
2) EA is treated as a fullword address.

DESCRIPTION

This instruction loads the MSC Status Register, Accumulator, Index Register, and Program Counter from 4 consecutive fullwords starting at the memory location specified by EA. These registers are loaded as follows:

Fullword Location	Register
E.A. bits 14 to 31	MSC Status Register
E.A. + 2, bits 0 to 31	MSC Accumulator
E.A. + 4, bits 14 to 31	MSC Index Register
E.A. + 6, bits 14 to 31	MSC Program Counter

The load of the PC causes the MSC to branch to whatever location was specified by the contents of location E.A. + 6. Additionally the load of the Status Register also causes the MSC Program Exception Bit (bit 0 of STAT 1 Register) to be set to match the conditions indicated by bit 30 of the fullword addressed by E.A. If bit 30 is 1, then the Program Exception Bit is set to 0 (error logged into Status Register); if bit 30 is 0, then the Program Exception bit is set to 1 (no errors).

PROGRAMMING NOTE

This instruction allows an orderly and complete return to a program whose execution was suspended by the execution of a @SEC (Sample for External Call) instruction with a non-zero Local Store Register C6. The @SEC instruction should be seen for more detail on this use.

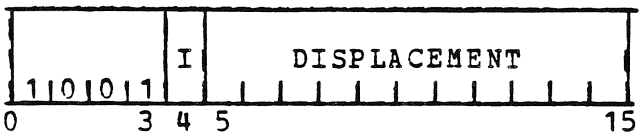
An @REC may also be used to reload the MSC to a completely known state. This allows an MSC program to completely reset all registers including the Status Register, after it has found that, for example, it executed a @SIO with a busy BCE. However, care must be taken in such circumstances to make what is placed into the Status Register meaningful. Bit 31 of fullword addressed by EA must be 1 -- the MSC is still busy. Likewise, if bit 30 is 0, then all error indication in bits 25 to 29 should be 0 also. If bit 30 is a 1, then what is loaded into the rest of the Status Register should indicate the kind of error being flagged.

3.3 CONDITIONAL SKIP INSTRUCTIONS

MSC instructions are normally executed in sequential order. A conditional skip instruction allows this normal sequential operation to be slightly modified. On the basis of some condition specified by the instruction, one or more sequential instructions following the conditional skip may be ignored. Execution is resumed at the instruction immediately following the skipped instructions.

TALLY AND SKIP ZERO @TSZ

FORMAT:



I	Effective Address ("EA")	INSTR Format
0	PC + Displacement	@TSZ ADDRESS
1	PC + Displacement + X	@TSZ ADDRESS(1)

NOTE:

1) PC refers to the updated program counter value - i.e., the address of the next sequential instruction.

2) Bit 5 is the sign of the two's complement displacement field. The range of the displacement is -1024 to +1023 halfwords.

3) X is the index register.

DESCRIPTION

The fullword addressed by EA is incremented by one and rewritten into memory at the same location (EA). If the incremented value is zero, the next instruction is skipped (program counter incremented by two). Otherwise, the program counter is simply incremented by one.

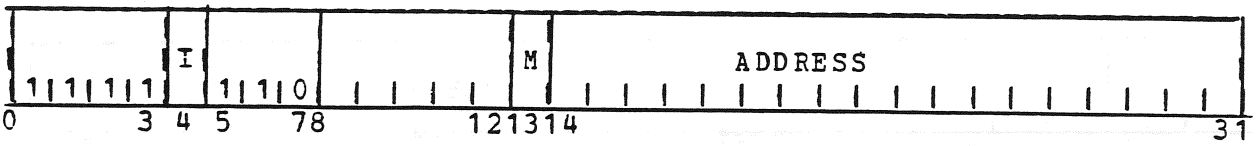
PROGRAMMING NOTE:

As with any instruction that writes information into memory, the affected memory location (EA) must not be storage protected.

COMPARE

@CI,@C

FORMAT:



I	M	EFFECTIVE VALUE E.V.	INSTR Format
0	0	Address*	@CI VALUE
0	1	(Address)	@C ADDRESS
1	0	Address + X*	@CI VALUE(1)
1	1	(Address + X)	@C ADDRESS(1)

* This 18-bit quantity is treated as a two's complement number, and the most-significant bit is sign extended 14 places to the left to fill 32 bits.

DESCRIPTION

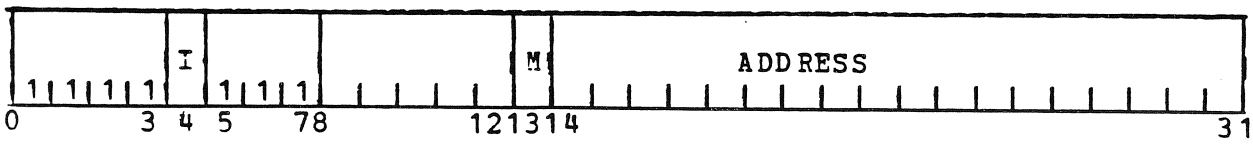
The 32-bit effective value is arithmetically compared to the accumulator. The present program counter is incremented by:

+2 If ACC > E.V.

+3 If ACC < E.V.

+4 If ACC = E.V.

FORMAT:



I	M	EFFECTIVE VALUE E.V.	INSTR Format
0	0	Address*	@TMI VALUE
0	1	(Address)	@TM ADDRESS
1	0	Address + X*	@TMI VALUE(1)
1	1	(Address + X)	@TM ADDRESS(1)

* This 18-bit quantity is treated as a two's complement number, and is right-justified with 14 leading 1's to fill 32 bits.

DESCRIPTION

The effective value is logically anded with the accumulator. If the result is not zero, then the program counter is incremented by 3 (skip next halfword). Otherwise, the program counter is incremented by two (execute next instruction). The original accumulator contents are not changed by this instruction.

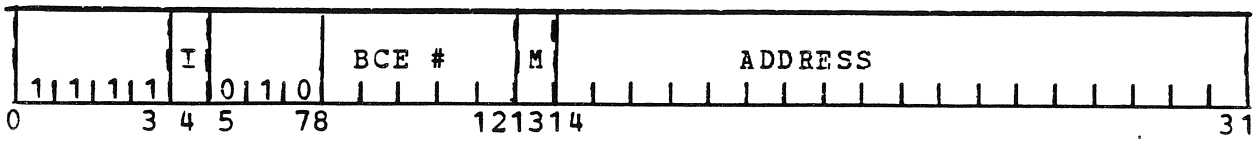
3.4 BCE REGISTER LOAD INSTRUCTIONS

The MSC is responsible for initializing Bus Control Elements (BCE's) in the IOP. These operations include loading certain BCE registers, and starting BCEs. The instruction to start a BCE (@SIO) is listed under Register Operations. Those instructions listed in this section allow the MSC to initialize BCE Base and PC registers.

These instructions are long formats (Figure 1.2(c)), that specify which BCE is to be loaded, and an 18-bit absolute address to be loaded into the specified register. Options allow the number of the BCE to be loaded to come from either the instruction or the MSC Accumulator. Other options allow the 18-bit addresses to come directly from the instruction or from the lower 18-bits of a fullword from memory. Indexing may be specified to facilitate table lookups of these quantities.

LOAD BCE BASE REGISTER @LBB

FORMAT:



I	M	EFFECTIVE VALUE E.V.	INSTR Format
0	0	Address	@LBB BCE,ADDRESS
0	1	(Address)	@LBB@ BCE,ADDRESS
1	0	Address + X	@LBB BCE,ADDRESS(1)
1	1	(Address + X)	@LBB@ BCE,ADDRESS(1)

DESCRIPTION

Field "BCE NUMBER" (Bits 8 through 12) specifies a BCE processor. If the field is non-zero, the field value selects the BCE. If the field is zero, the lower 5 bits of the MSC accumulator, (Bits 27 to 31) specify the desired BCE.

If the specified BCE is in the Wait State, the effective value, computed according to the above table, is stored in the BCE's base register. If the specified BCE is busy or is halted, bit 13 in the MSC status register is set, the MSC Program Exception bit is set to 0 (error), and nothing is done to the specified BCE processor.

In both cases, the MSC program counter is incremented by 2, and the next sequential instruction is executed.

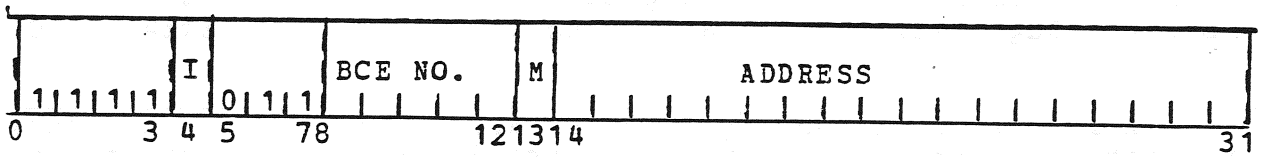
PROGRAMMING NOTE

The IOP interprets a BCE number of 0 as the MSC, and consequently, if an @LBB instruction is executed with a final BCE number of 0 (a possibility only if the ACC = 0), the MSC will test itself and find itself busy. This results in the previously described error action being taken.

LOAD BCE PROGRAM COUNTER

@LBP

FORMAT:



I	M	EFFECTIVE VALUE E.V.	INSTR Format
0	0	Address	@LBP BCE,ADDRESS
0	1	(Address)	@LBP@ BCE,ADDRESS
1	0	Address + X	@LBP BCE,ADDRESS(1)
1	1	(Address + X)	@LBP@ BCE,ADDRESS(1)

DESCRIPTION

Field "BCE NUMBER" (Bits 8 through 12) specify a BCE processor. If the field is non-zero, the field value selects the BCE. If the field is zero, the lower 5 bits of the MSC accumulator (bits 27 to 31) specify the desired BCE.

If the specified BCE is in the Wait State, the effective value, computed according to the above table, is stored in that BCE's program counter. If the specified BCE is busy or halted, bit 12 of the MSC status register is set, the MSC Program Exception bit is set, and nothing is done to the specified BCE processor.

In all cases, the MSC program counter is incremented by 2, and the next sequential instruction is executed.

PROGRAMMING NOTE

The IOP interprets a BCE number of 0 as the MSC, and consequently, if an @LBP instruction is executed with a final BCE number of 0 (a possibility only if the ACC = 0), the MSC will test itself and find itself busy. This results in the previously described error action being taken. An @LBP to a currently executing or halted BCE causes an MSC exception (bits 12, 16) but does not halt the MSC.

3.5 REGISTER OPERATIONS

The MSC instruction set includes a class of instructions that allow the MSC to read or set various IOP registers, not all of which are associated with the MSC. These registers include the Busy/Wait register, the Program Exception Register, the BCE-MSC Indicators, the MSC Status Register, the Fail Discretes, and the External Call register.

All the Register Operations are 16-bit short format instructions of the form shown in Figure 1.2(b) with a common opcode of 1110, and an I-bit of 0. Each Register Operation has a separate OPX field, as shown in Table 3.1.

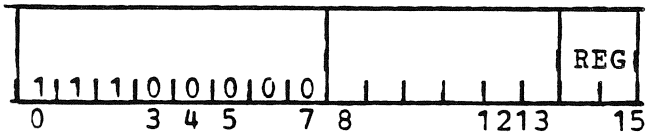
TABLE 3.1

LIST OF REGISTER OPERATIONS

OPX		OPERATION
0	LAR	Load MSC ACC with specified IOP register
1	SFD	Set Fail Discrete Register by ORing it with MSC accumulator bits 0,1,2,3 and 4.
2	RFD	Reset Fail Discrete Register A 1 in ACC Bit i will clear Fail Discrete Register Bit i. A 0 in the ACC causes no change.
3	LMS	Load MSC ACC with MSC Status Word.
4	SIO	Start BCE's by Or'ing accumulator with busy/wait register (STAT4). A one in bit position i of the ACC will place BCEi in busy state.
5	XAX	SWAP ACC AND X. The lower 18 bits of ACC go into X, and X is sign extended to fill ACC.
6	SEC	Sample for External Call. Check MSC Local Store register C6. If non-zero, save MSC register and branch to specified program.
7	RBI	Reset BCE indicator. Reset the specified BCE's indicator bit to 0.

LOAD MSC ACC. WITH AN IOP STATUS REGISTER @LAR

FORMAT:



INSTR Format

@LAR REGISTER

DESCRIPTION

This instruction allows the MSC ACC to be loaded, under program control, with a specified IOP register. The REG field determines which IOP register is involved. The following table summarizes the REG field assignments.

REG	IOP REGISTER
0	Program Exception Indicators - STAT 1
1	BCE-MSC Indicators
2	FAIL DISCRETES
3	BUSY/WAIT INDICATORS - STAT4

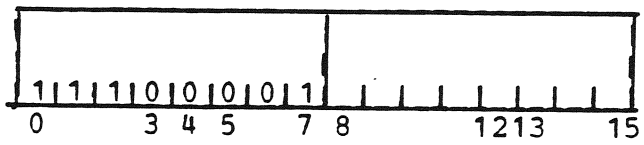
In all cases, bit i of the indicated register goes into bit i of the ACC. ACC bits with no corresponding IOP register bits are zero filled.

Fail Discrete register bits are assigned as follows:

Bit 0 - MSC inhibit Fail Discrete outputs
Bit 1-4 - Input Fail Discretes 1-4
Bit 5-8 - Output Fail Discretes 1-4
Bits 9-31 - Zero

SET FAIL DISCRETES @SFD

FORMAT:



INSTR Format

@SFD

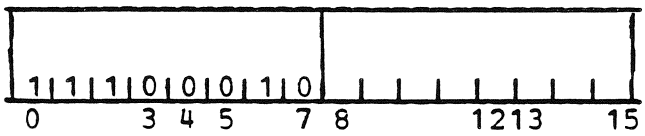
DESCRIPTION

This instruction allows the MSC ACC bits 0,1,2,3, and 4 to be logically OR'ed with the Fail Discrete register. Thus, a 1 in bit i of the ACC (i=1,2,3,4) will set the equivalent Fail Discrete bit. A 0 will cause no change.

Setting bit 0 will inhibit the output of the four fail discretes from bits 1-4 of the four Fail Discrete registers, but will not affect the way these bits may be changed by @SFD or @RFD.

RESET FAIL DISCRETES @RFD

FORMAT:



INSTR Format

@RFD

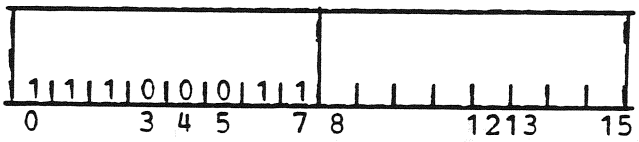
DESCRIPTION

This instruction allows the Fail Discrete Register to be selectively reset. A 1 in MSC Accumulator bit i (i=0,1,2,3,4) will cause bit i in the Fail Discrete Register to be reset. A 0 in the MSC Accumulator causes no changed.

Resetting bit 0 of the Fail Discrete Register allows activation of the four fail discretes from the Fail Discrete Register bits 1-4.

LOAD ACC WITH MSC STATUS @LMS

FORMAT:



INSTR Format

@LMS

DESCRIPTION

This instruction allows the MSC ACC to be loaded under program control, with the MSC status word. The format of this status word as it is loaded into the ACC is as follows:

Bit 31 STAT4 bit 0 - The Busy/Wait bit for the MSC. It will always be 1 when this instruction is executed.

Bit 30 STAT1 bit 0 - The Program Exception bit for the MSC. This bit will be set to 1 only if the execution of some previous instructions resulted in an error of some kind. Bits 29 through 26 catalog the exact error. Note that a 1 in this bit corresponds to a 0 in bit 0 of the Program Exception register. The converse is also true.

Bit 29 Illegal opcode. If set, this bit indicates that an illegal opcode was encountered at some point in the past. Note that the only way this bit could be set when an @LMS is executed is if an illegal opcode was detected in the past, and the CPU restarted the MSC without clearing the status word first.

Bit 28 Boundary alignment error. If set, this bit indicates that a long format instruction was encountered at an odd halfword boundary.

Bit 27 LBB error. The BCE that was specified by a previous @LBB instruction was busy at the time the instruction was executed, and consequently its Base Register was not loaded.

Bit 26 LBP error. The BCE that was specified by a previous @LBP was busy at the time the instruction was executed, and consequently its Program Counter was not loaded.

Bit 25 SIO error. A previous execution of a @SIO instruction tried to set busy at least one BCE that was already busy.

Bit 24,23 Reserved

Bit 22 Self-Test Fault. The execution of a previous MSC self test instruction detected a fault in the MSC.

Bits 21-14 Reserved.

Bits 13-0 Set to zero.

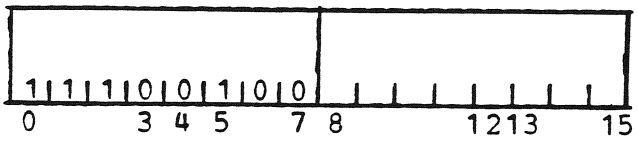
PROGRAMMING NOTE

The MSC Status Register may be set to any desired value as part of the execution of a Return From External Call (@REC) instruction.

START I/O

@SIO

FORMAT:



INSTR Format

@SIO

DESCRIPTION

This instruction starts execution of BCE's by ORing the Accumulator with the Busy/Wait register (STAT4). A 1 in bit position i of the ACC will place BCE(i) in a busy state.

If any of the BCE's designated by the ACC are already busy (their bit in the Busy/Wait Register is already set) bit 11 of the MSC Status Register is set and the MSC Program Exception bit is set.

In any case, the ACC is still OR'ed into STAT4 and the next sequential instruction is executed.

PROGRAMMING NOTE

Bit 0 of STAT4 is the MSC Busy/Wait bit, and is set to 1 whenever the MSC is executing instructions. Consequently, execution of this instruction with a 1 in bit 0 of the ACC will cause the error status bits to be set (Program Exception bit and Status Register bit 11). As before, execution continues with the next instruction.

Although an @SIO instruction will set a BCE's Busy/Wait bit to busy within the time frame of the instructions' execution in the MSC, the BCE will not necessarily begin immediate execution of the first instruction.

Possible execution delays include:

1) The time required for the BCE to recognize that its Busy/Wait bit is set (up to 16.5 usec).

2) The time required to bring out the PC and request the instruction word from memory (up to 33 usec.)

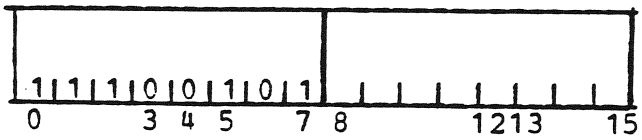
3) Any time required for the memory read request to reach the DMA channel, be read from memory and be returned to the BCE.

The last case delays the BCE setup only when it requires over 16 usecs to be handled. This may occur whenever several BCE/MSC processors have requested memory reads at the same time.

EXCHANGE ACC AND X

@XAX

FORMAT:



INSTR Format

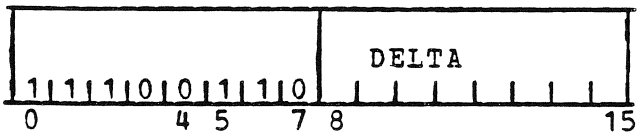
@XAX

DESCRIPTION

This instruction allows a swap between ACC and X. The lower 18-bits of the ACC go into X, and X is sign extended to fill the ACC.

SAMPLE FOR EXTERNAL CALL

FORMAT:



INSTR Format

@SEC ADDRESS

NOTE:

Address = PC + DELTA, where PC is updated Program Counter and DELTA is an integer between -128 and 127.

DESCRIPTION:

In conjunction with the MSC instruction, Return from External Call (@REC), this instruction allows the CPU to request that the MSC suspend execution of its present program and initiate a new one. Further, the suspension of the present program is done in a fashion that allows an orderly return to it once the CPU-designated program is complete. The program specified by the CPU is termed an "external call".

The point of communication between the MSC and the CPU is MSC Local Store Register C6. To request a program to be executed, the CPU loads C6 with the (fullword) address of the program's start in main memory. Whenever the MSC executes an @SEC it samples C6. If C6 is zero, the MSC continues with the next instruction. The @SEC is a no-operation in this case.

If, however, the MSC finds a non-zero C6 when it executes an @SEC, it takes the contents as a main memory address, and stores in the 4 fullwords addressed by C6 the following quantities:

*Fullword C6 -- MSC Status Register right justified and padded on left with 0's.

Fullword C6 + 2 -- MSC Accumulator

Fullword C6 + 4 -- MSC Index Register right justified and padded on left with 0's.

*If C6 is odd, registers are stored on EVEN word boundaries (Low Order Address bit is ignored).

Fullword C6 + 6 -- MSC Program Counter plus DELTA right justified and padded on left with 0's. (PC is updated PC, i.e. points to next instruction after @SEC.)

These quantities are used by @REC at the end of the new program to restore the MSC to its original condition.

After this, PC is loaded with C6 + 8, C6 is zeroed, the MSC Status Register is set to 1 (busy but no errors) and the MSC Program Exception bit (bit 0 of STAT 1) is set to 1 (no errors). The MSC thus branches to the program beginning at C6 + 8 with a clean status. Any errors made by the MSC while executing this new program will be reflected in the Status Register, but with no confusion with the status of the original program. Note that if C6+8 points to an odd halfword, the MSC will branch to that halfword.

The fullword at C6 + 6 contains the return address to the program containing the @SEC. Adding the DELTA field to PC when the store is made allows the original program the option of being informed when an external call is actually taken. A 0 DELTA will always return the MSC to the instruction immediately following the @SEC with the result that the MSC program would never know when a call was taken. A non-zero DELTA will return the MSC to a programmer-specified location after the CPU specified program is complete. This would allow, for example, the original MSC program to update a timer or clock to take into account the time spent in the CPU specified program.

Once C6 has been zeroed, the CPU is free to reload it again with another program address, which in turn will be accepted by the MSC the next time it executes an @SEC.

PROGRAMMING NOTE

Figure 3.1 lists a possible MSC program segment using @SEC's. The @SEC at location 100 allows an external call from the CPU to be accepted, but with a 0 DELTA, control will always return to 101.

Following 100 is a section of code that monitors the STAT 4 (Busy/ Wait) Register for a maximum of about 10 msec. This could be done by a single @RAW with a time out count of about 300. However, CPU issued external calls would be ignored for the entire 10 msec. period. The segment shown here waits at most 1 msec. between @SEC's. It executes a @RAW for about 1 msec. and then branches to 300 where an @SEC permits the acceptance of a CPU-specified external call. If no external call is present, the @TSZ at 301 increments a counter

(initialized to -10) and returns to repeat the @RAW if less than 10 msec. have passed. If an external call is taken at 300, the non-zero DELTA will cause the return from the call to be to location 310 where the counter is decremented an extra time to account for time spent in the external call. This can obviously be elaborated if more accurate timing is required.

Also on Figure 3.1 is an outline of a MSC program "PGM" that may be started by an external call. The first four fullwords are allocated to holding the MSC registers at the time of the @SEC. The program to execute the desired function starts at PGM + 8 and ends with the @REC PGM, which returns the MSC to the original program.

Section 4.0, External Calls, contains further information on the use of the @SEC instruction.

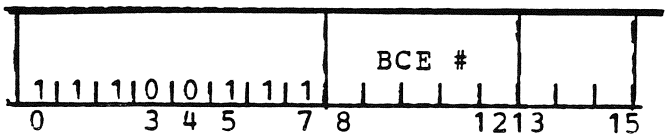
As with any instruction that writes information into memory, the affected memory locations (C6,C6+2,C6+4,C6+6) must not be storage protected.

```
LOCATION                INSTRUCTION                     COMMENT

Main Program:

100                     @SEC    0

                        @LI     -10                     Initialize REPEAT
                        @S      COUNT                   Count.

200 LOOP:               @RAW    30                      Watch for some condition
201                     @BC     0, TIMECHECK            -time out
202

                              Condition fulfilled


300 TIMECHECK:          @SEC    UPDATETIME              Allow an interrupt
301                     @TSZ    COUNT                   decrement timer
302                     @BC     0, LOOP                 timer not zero yet
303 TIMFOUT:
                              Start of timeout          COUNT reached 0
                              routine
310 UPDATETIME:         @TSZ    COUNT                   An interrupt was taken,
311                     @BC     0, TIMECHECK + 1        decrement clock one
312                     @BC     0, TIMEOUT              extra time.

Program specified by External Call

400 PGM:                DS      4F                      Allocate 4 fullwords of
401                                                     storage for MSC Registers

                              Execute desired function

                        @REC    PGM                     Return to original
                                                        MSC Program
```

Figure 3.1. Sample Code Using @SEC

RESET BCE INDICATOR

@RBI



INSTR Format

@RBI BCE#

DESCRIPTION

This instruction resets the specified BCE's BCE/MSC indicator bit to zero. For BCE i, this is bit i of the BCE/MSC indicator register. If the BCE specified in the instruction word is zero, the lower 5 bits of the MSC Accumulator (bits 27-31) indicate the desired BCE.

After resetting the specified indicator bit, the MSC increments its Program Counter by 1 and continues with the next sequential instruction.

PROGRAMMING NOTE

No check is made as to whether or not the specified BCE number is valid (i.e., between 1 and 24). If the specified BCE does not exist, this instruction is effectively a "no-operation".

3.6 REGISTER IMMEDIATE INSTRUCTIONS

Many common programming sequences involve the use of small constants in operations involving the MSC ACC and X registers. To facilitate such operations without the penalty of constant references to memory, the MSC instruction set includes a class of 16-bit short format instructions that has, as part of the instruction, an 8-bit signed integer. All such instructions have the format shown in Figure 1.2(b), with an opcode of 1110 and an I-bit of 1. The OPX field determines what is to be done with the constant found in bits 8 through 15 of the instruction. Table 3.2 lists the OPX fields and their interpretation.

TABLE 3.2

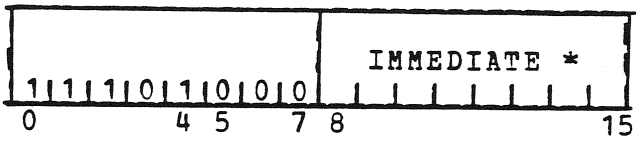
REGISTER IMMEDIATE INSTRUCTION

OPX		OPERATION
0	@NIX	Normalize and Increment X. The ACC is shifted left until it becomes ≤ 0. Each time the ACC is shifted X is incremented by the Immediate constant.
1	@TAX	Tally Accumulator to X. Store the sum of the Immediate and ACC into X.
2	@TXI	Tally X. Add Immediate to X. If result is ≤ 0, skip next instruction.
3	@LXI	Load X Immediate. Load the X register with the Immediate constant.
4	@TXA	Tally X to ACC. The sum of the X register and the Immediate constant is loaded into the Accumulator.
5	@TI	Tally Accumulator Immediate. The Immediate constant is added to the ACC.
6	@SAI	Subtract ACC from Immediate. The Immediate constant minus the Accumulator is loaded into the Accumulator.
7	@LI	Load Immediate. The Immediate constant is loaded into the Accumulator.

NORMALIZE AND INCREMENT X

@NIX

FORMAT:



* Two's complement number between -128 and +127. Forms a 32-bit constant by sign extending bit 8.

INSTR Format

@NIX IMM

DESCRIPTION

The NIX instruction shifts the ACC left until either the ACC sign bit (bit 0) is one, or the ACC. equals zero. Each time the ACC is shifted, the signed immediate value is added to the index register X. The shift and increment operation is done before either test. If the ACC. reaches zero, the index register is cleared. The following figure flowcharts the operation of this instruction.

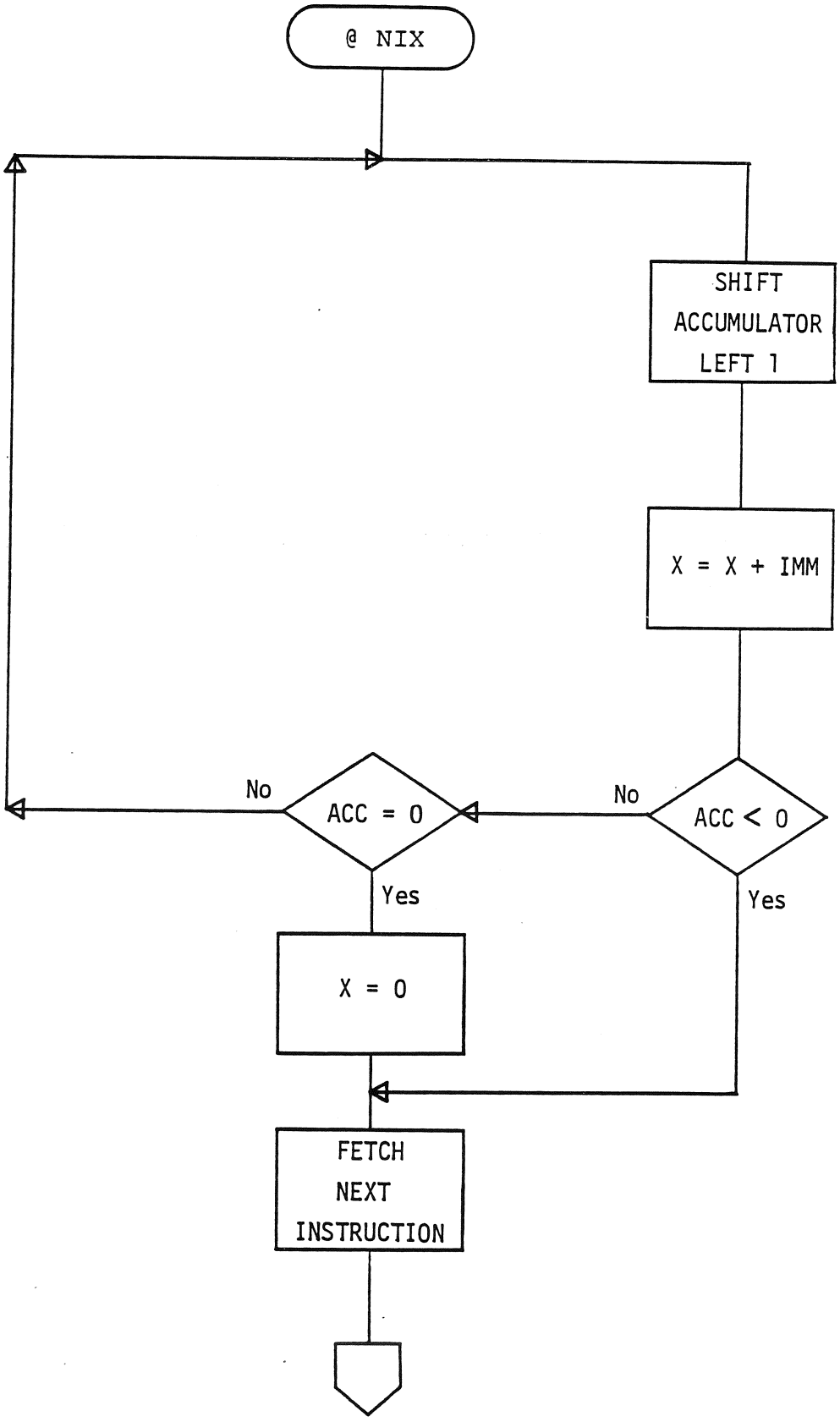


Figure 3.2 @ NIX Instruction Execution

PROGRAMMING NOTE

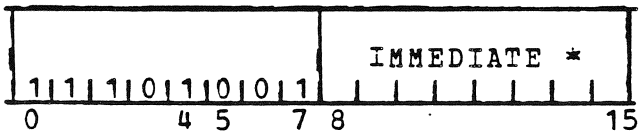
A typical use of the NIX instruction would be to determine which BCE's have set their NOGO bit. The following sequence performs this process:

```
          .
          .
          .
          @LXI      0          INITIALIZE X TO ZERO
          @LAR      0          LOAD ACC WITH GO/NOGO BITS
          @SAI      -1         INVERT SO THAT:  1=ERROR, 0=OK
LOOP:     @NIX      1          FIND NEXT BAD BCE
          @BC       4,DONE     EXIT WHEN NO MORE
               .
               .
               .
          PROCESS BAD BCE.X REG. CONTAINS BCE NUMBER.  NOTICE THAT
          FIRST EXECUTION OF NIX BYPASSES MSC POSITION WITH INITIAL
          SHIFT, AND ALL FUTURE NIX's ELIMINATE BCE PROCESSED ON
          PREVIOUS ITERATIONS.
               .
               .
               .
          @BU       LOOP
```

TALLY ACC TO X

@TAX

FORMAT:



* Two's complement number. Between -128 and +127. Forms a 32-bit constant by sign extending bit 8.

INSTR Format

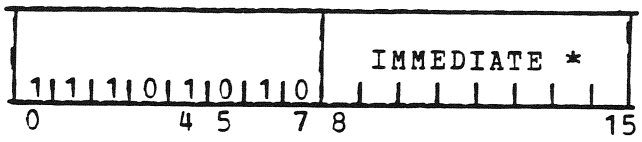
@TAX IMM

DESCRIPTION

The Lower 18 bits of the sum of the ACC and the IMM field is stored in the Index Register (X). An IMM value of 0 permits X to be loaded directly from the ACC. The ACC is unchanged.

TALLY X

FORMAT:



* Two's complement number. Between -128 and +127. Forms a 32-bit constant by sign extending bit 8.

INSTR Format

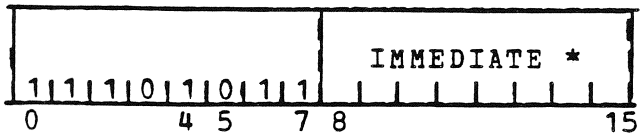
@TXI IMM

DESCRIPTION

The TXI instruction allows the Index Register plus IMM to be loaded back into X. If the new X is ≤ 0 the next halfword instruction is skipped. (program counter incremented by 2). Otherwise, the next halfword instruction is executed (program counter incremented by 1).

LOAD X IMMEDIATE @LXI

FORMAT:



* Two's complement number. Between -128 and +127. Forms a 32-bit constant by sign extending bit 8.

INSTR Format

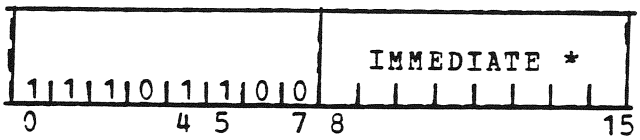
@LXI IMM

DESCRIPTION

The LXI instruction allows the data in IMM (sign extended to 18 bits) to be loaded into the Index Register (X). The MSC program counter is incremented by 1.

TALLY X TO ACC

FORMAT:



* Two's complement number. Between -128 and +127. Forms a 32-bit constant by sign extending bit 8.

INSTR Format

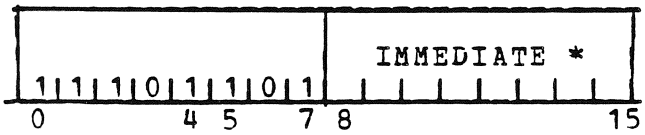
@TXA IMM

DESCRIPTION

The TXA instruction allows the X plus IMM to be stored in ACC (sign extended to 32 bits) IMM equal to zero allows sign extended X to be stored in ACC. X is treated as an 18-bit two's complement number with the sign in its most-significant bit. The MSC program counter is incremented by 1. The contents of X are not changed.

TALLY ACC IMMEDIATE @TI

FORMAT:



* Two's complement number. Between -128 and +127. Forms a 32-bit constant by sign extending bit 8.

INSTR Format

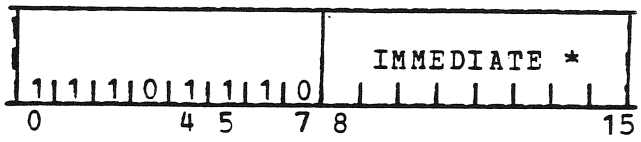
@TI IMM

DESCRIPTION

The TI instruction allows ACC plus IMM (sign extended to 32 bits) to be loaded in ACC. The MSC program counter is incremented by 1.

SUBTRACT ACC FROM IMM @SAI

FORMAT:



* Two's complement number. Between -128 and +127. Forms a 32-bit constant by sign extending bit 8.

INSTR Format

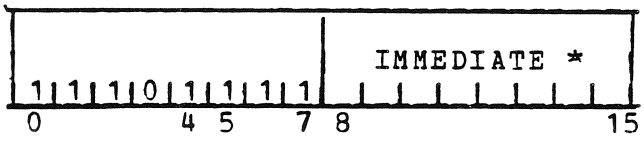
@SAI IMM

DESCRIPTION

The SAI instruction allows the contents of the ACC to be subtracted from IMM (sign extended to 32 bits) and loaded in ACC. If IMM is equal to zero, a two's complement of ACC is performed. If IMM is equal to minus one, a one's complement of ACC is performed (each bit is inverted). The MSC program counter is incremented by 1.

LOAD ACCUMULATOR IMMEDIATE @LI

FORMAT:



* Two's complement number. Between -128 and +127. Forms a 32-bit constant by sign extending bit 8.

INSTR Format

@LI IMM

DESCRIPTION

The LI instruction allows the contents of IMM (signed extended to 32 bits) to be loaded into ACC. The MSC program counter is incremented by 1.

3.7 REPEAT INSTRUCTIONS

One of the major tasks of the MSC is to monitor the BCEs. A typical MSC program involves initiating and starting some BCEs and then waiting for the BCEs to complete their assigned tasks. The Repeat Instructions perform this task directly, namely they wait until a set of BCEs specified by the ACC reach some condition. In addition, they include a programmable time-out check to prevent them from waiting indefinitely if some BCE never reaches the desired state.

All Repeat instructions use the same 16-bit short formats (Figure 1.2(b)) with an Opcode of 1101. The lower 8-bits, bits 8 through 15, and the I-bit are used to compute a count of the maximum time the instruction will spend waiting. If the time is exceeded, the next sequential instruction is executed. If the desired condition is reached before the time out, the next halfword instruction is skipped. Figure 3.3 diagrams the operation of any Repeat.

A time out value of zero will cause the Repeat instruction to test once, and only once, for the desired condition. If the condition is true, the next instruction is skipped. If the condition is false, the next instruction is executed.

A non-zero time out value will cause the specified test to be tried three times every 33 usec, or roughly, once every eight microseconds with 8 usec used to check for a time out. Every 33 usec the time out value is decremented, and the test is repeated until either the time out value reaches zero or the desired condition is reached.

A 33 usec period corresponds roughly to the rate at which a BCE can transmit or receive a word of data. This allows the programmer to estimate the time a Repeat should loop by analyzing how many data words are transferred through the MIA's by the relevant BCE's before the desired condition is reached. This time out value must also take into account the time required by BCE's to fetch and execute any non-transmit/receive instructions that might be present in the BCE program.

The OPX field determines exactly which condition will be tested. These conditions are listed in Table 3.3.

PROGRAMMING NOTES:

An accumulator value of 0 will cause both the @RAI and @RAW to reach their specified conditions the first time that the test is made. Consequently, the PC will be incremented by 2 and the next halfword skipped. A zero accumulator will never allow an @RNI or @RNW to reach their repeat conditions. Consequently, each repeat will loop

until the maximum repeat count is reached, at which point the next sequential instruction is begun.

Since processor 0 corresponds to the MSC, (which is busy during instruction execution), the MSC execution of an @RAW instruction will correspond to a loop until the maximum repeat count is reached.

TABLE 3.3

REPEAT INSTRUCTIONS

OPX		CONDITION
000	@RAW	Repeat until all BCEs for which there is a 1 in the ACC reset their Busy/Wait bits to Wait.
001	@RNW	Repeat until any BCEs for which there is a 1 in the ACC reset their Busy/Wait bits to Wait.
100	@RAI	Repeat until all BCEs for which there is a 1 in the ACC set their BCE-MSC Indicators to 1.
101	@RNI	Repeat until any of the BCEs for which there is a 1 in the ACC set their BCE-MSC Indicators to 1.

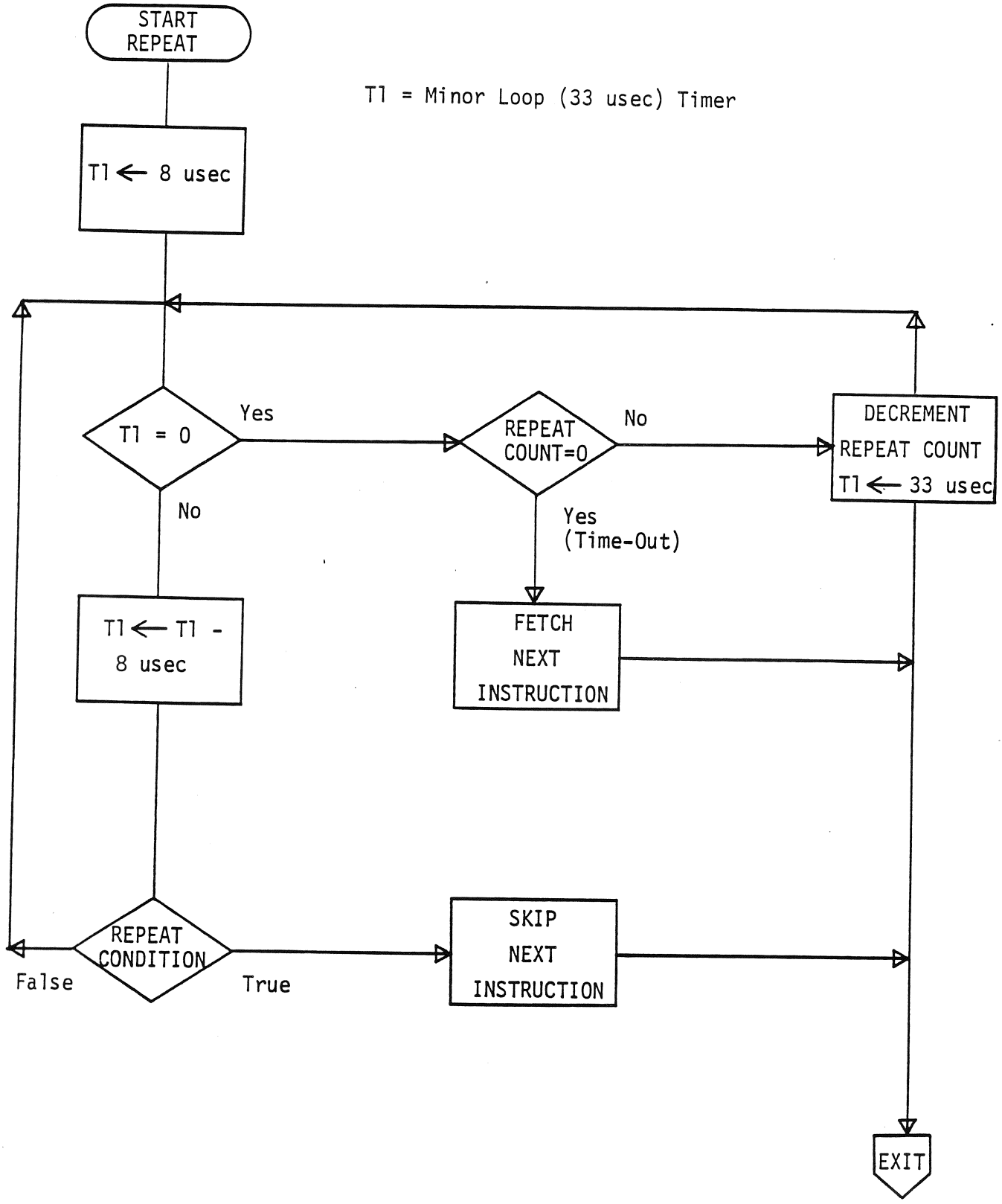
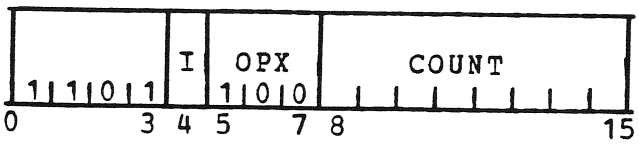


Figure 3.3 General Repeat Instruction Execution

REPEAT UNTIL ALL INDICATORS

@RAI

FORMAT:



I	MAXIMUM REPEAT COUNT	INSTR Format
0	Count	@RAI COUNT
1	X + Count	@RAI COUNT(1)

DESCRIPTION

This instruction causes a delay in MSC program execution until either:

1) All of a set of specified BCE's have set their BCE - MSC indicator bits to 1.

2) A maximum wait time has been reached.

If condition 1) is reached before the time out, the MSC skips the next halfword instruction and resumes program execution. If the maximum wait time, as specified by the maximum repeat count, is reached first, the MSC begins execution of the next halfword instruction.

The condition "are all specified BCE indicator bits set" is tested by logically ANDING the MSC accumulator with the complement of the BCE-MSC indicator register. If the result is all zeros, the desired condition has been reached.

The maximum wait time has a basic resolution of 33 microseconds and is derived from the maximum repeat count. In non-index mode, the maximum repeat count of 255 corresponds to a maximum wait time of 8.4 milliseconds. In index mode, the maximum possible count of 262143 corresponds to 8.65 seconds.

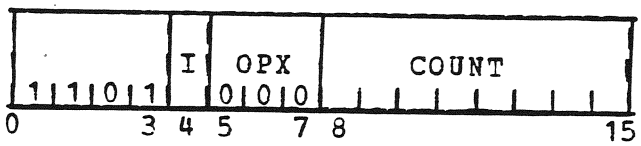
At the time this instruction is executed, the MSC accumulator is presumed to contain a bit mask specifying which BCE's are to be involved in the test for the desired repeat condition. A 1 in bits i of the accumulator specifies that the status of BCE(i) is to be included in the test; a 0 indicator that BCE(i) can be ignored.

Figure 3.3 diagrams the exact operation of this and the other Repeat instructions.

REPEAT UNTIL ALL WAITING

@RAW

FORMAT:



I	MAXIMUM REPEAT COUNT	INSTR Format
0	Count	@RAW COUNT
1	X + Count	@RAW COUNT(1)

DESCRIPTION

This instruction causes a delay in MSC program execution until either:

1) All of a set of specified BCE's have set their busy/wait status bits to "wait"

2) A maximum wait time has been reached.

If condition 1) is reached before the time out, the MSC skips the next halfword instruction and resumes program execution. If the maximum wait time, as specified by the maximum repeat count, is reached first, the MSC begins execution of the next halfword instruction.

At the time this instruction is executed, the MSC accumulator is presumed to contain a bit mask specifying which BCE's are to be involved in the test for the desired repeat condition. A 1 in bit i of the accumulator specifies that the status of BCE(i) is to be included in the test; a 0 indicates that BCE(i) can be ignored.

The maximum wait time has a basic resolution of 33 microseconds and is derived from the maximum Repeat count. In non-index mode, the maximum Repeat count of 255 corresponds to a maximum wait time of 8.4 milliseconds. In index mode, the maximum possible count of 262143 corresponds to 8.65 seconds.

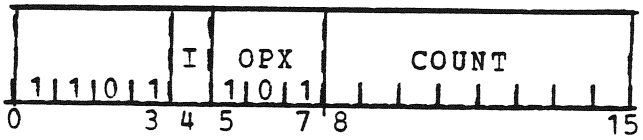
The condition "are all specified BCE's waiting" is tested by logically ANDING the MSC accumulator with STAT4 (the Busy/Wait register), and testing for all zeros. A 1 in bit i of STAT4 indicates that BCE(i) is busy; a 0 indicates that it is not. Consequently, if

the result of this "and" operation is all zeros, then all the specified BCE's will be in the desired state.

Figure 3.3 diagrams the exact operation of this and the other Repeat instruction.

REPEAT UNTIL ANY INDICATORS

FORMAT:



I	MAXIMUM REPEAT COUNT	INSTR Format
0	Count	@RNI COUNT
1	X + Count	@RNI COUNT(1)

DESCRIPTION

This instruction causes a delay in MSC program execution until either:

1) Any of a set of specified BCE's have set their BCE-MSC indicator bits to 1 or,

2) A maximum wait time has been reached.

If condition 1) is reached before the time out, MSC skips the next halfword instruction and resumtes program execution. If the maximum wait time; as specified by the maximum repeat count, is reached first, the MSC begins execution of the next halfword instruction.

The condition "are any of the specified BCE indicators set" is tested by logically ANDING the MSC accumulator with the BCE-MSC indicator register. If the result is non-zero, the desired condition has been reached.

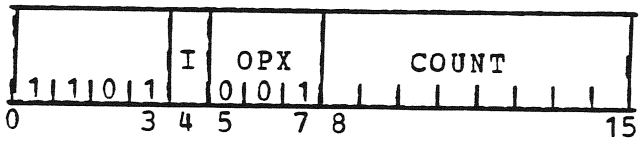
The maximum wait time has a basic resolution of 33 microseconds and is derived from the maximum Repeat count. In non-index mode, the maximum Repeat count of 255 corresponds to a maximum wait time of 8.4 milliseconds. In index mode, the maximum possible count of 262143 corresponds to 8.65 seconds.

At the time this instruction is executed, the MSC accumulator is presumed to contain a bit mask specifying which BCE's are to be involved in the test for the desired repeat condition. A 1 in bit i of the accumulator specifies that the status of BCE(i) is to be included in the test; a 0 indicates that BCE(i) can be ignored.

Figure 3.3 diagrams the exact operation of this and the other Repeat instructions.

REPEAT UNTIL ANY WAITING @RNW

FORMAT:



I	MAXIMUM REPEAT COUNT	INSTR Format
0	Count	@RNW COUNT
1	X + Count	@RNW COUNT(1)

DESCRIPTION

This instruction causes a delay in MSC program execution until:

1) Either any of a set of specified BCE's have set their Busy/Wait status bits to "Wait" or,

2) A maximum wait time has been reached.

If condition 1) is reached before the time out, the MSC skips the next halfword instruction and resumes program execution. If the maximum wait time, as specified by the maximum repeat count, is reached first, the MSC begins execution of the next halfword instruction.

The condition "are any specified BCE's waiting" is tested by logically ANDing the MSC accumulator with the complement of STAT4 (the Busy/Wait register) and testing for non-zeros. A 1 in bit i of STAT4 indicates that BCE(i) is busy; a 0 indicates that it is not busy. Consequently, if the result of the above operation is a non-zero value, then at least one BCE from the set is not busy, and the condition is fulfilled.

The maximum wait time has a basic resolution of 33 microseconds and is derived from the maximum Repeat count. In non-index mode, the maximum Repeat count of 255 corresponds to a maximum wait time of 8.4 milliseconds. In index mode, the maximum possible count of 262143 corresponds to 8.65 seconds.

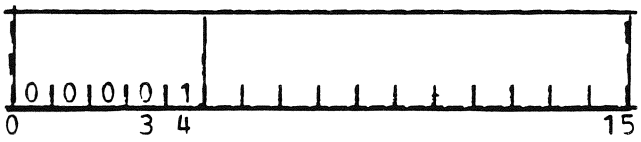
At the time this instruction is executed, the MSC accumulator is presumed to contain a bit mask specifying which BCE's are to be involved in the test for the desired repeat condition. A 1 in bit i

of the accumulator specifies that the status of BCE(i) is to be included in the test; a 0 indicates that BCE(i) can be ignored.

Figure 3.3 diagrams the exact operation of this and the other Repeat instructions.

3.8 SPECIAL INSTRUCTIONS

In addition to the instructions described earlier, the MSC instruction set includes a variety of instructions that place the MSC in the Wait state, delay MSC execution for program-settable periods of time, and cause an interrupt in the CPU. This section describes these instructions.



DESCRIPTION

This instruction causes the MSC to go into the WAIT state (bit 0 of STAT4 and bit 17 of the MSC Status Register set to zero). No more instructions are executed. The MSC program counter is updated to point to the next instruction. No other MSC register is changed.

PROGRAMMING NOTE

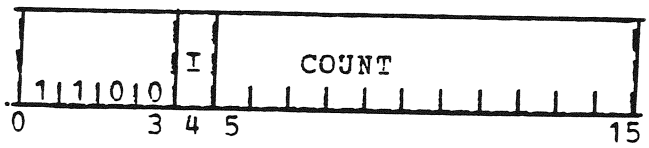
The MSC does not reference any of its Local Store registers once it is in the Wait State. During this time, the CPU is free to PCO or PCI any MSC register. However, once the CPU performs a PCO (to set the MSC busy), the MSC will detect the setting of its busy bit within 2 microseconds, and will use whatever is in its PC as the address of the first instruction. Therefore, once the MSC has been set to busy, CPU PCOs to the MSC Local Store should be avoided.

Note that if there is no PCO to the MSC's PC (MSC Local Store register A2), then the MSC will start with the first instruction following the last #WAT instruction.

DELAY

@DLY

FORMAT:



I	Effective Count ("EC")	INSTR Format
0	Displacement	@DLY COUNT
1	X+Displacement	@DLY COUNT(1)

NOTE:

1) Displacement is a positive number in the range 0 to 2047.

2) X refers to the index register.

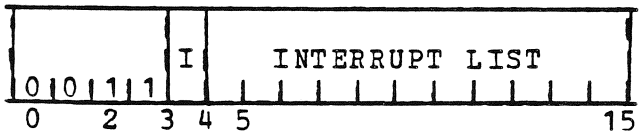
DESCRIPTION

This instruction performs no operation other than delaying the execution of the next instruction. The time period delayed is indicated by the 18-bit EC, with a resolution of 2 microseconds. The maximum delay is roughly .54 sec. At the end of this delay, the program counter is incremented by one, and the next instruction is executed.

Note that after every eighth MSC microcycle the time between MSC microcycles is 2.5, not 2, microseconds.

INTERRUPT CPU @INT

FORMAT:



Effective Interrupt List (EIL)

I

0	Interrupt List *	@INT IL
1	X "OR" Interrupt List*	@INT IL(1)

* Right justified and left padded by 7 zeroes.

INSTR Format

@INT INTERRUPT TEST

DESCRIPTION

This instruction loads the 12 bit IOP Interrupt Register C and causes an interrupt to the GPC to be set if at least one of the bits is a 1. Each bit in the least significant 12 bits of the effective interrupt list corresponds to one of the 12 "IOP Program Interrupts". In the table shown below, bit 0 is defined as the most significant bit of EIL, and bit 17 as the least significant bit. The table lists the correspondence between EIL and the IOP program interrupts. Note that any combination between none and 12 of these interrupts may be set simultaneously.

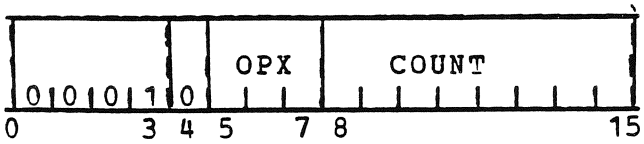
EIL BIT	IOP PROGRAM INTERRUPT NO.	INTERRUPT LEVEL
17	12	16
16	11	16
15	10	16
14	9	16
13	8	16
12	7	16
11	6	16
10	5	16

09	4	16
08	3	16
07	2	16
06	1	16
05-00	UNASSIGNED	

SELF TEST - MSC*

@STP

FORMAT:



INSTR Format

@STP

NOTE:

1) The COUNT field is at a don't care state unless the OPX field equals 001.

2) Use of the OPX field selects one of three tests as follows:

   a) OPX = 000 - Normal STP sequence

   b) OPX = 001 - Queue-overflow test

   c) OPX = 010 - ROS parity error test

   d) OPX = 011 - Diagnostic Data Flow error test

DESCRIPTION

If OPX equals 000, this instruction initiates execution of a special micro program to perform self tests on the hardware supporting the MSC. These tests include checks of the:

1) MSC Local Store Registers

2) MSC Data Flow operations

3) Ability of MSC to read and write from memory

4) Redundancy Management Fail discretes

5) IOP ROS parity circuit.

If any faults are detected, the MSC Program Exception bit is set to 0 (fault detected) and the MSC status registers bit 8 is set to 1 (Self Test failure).

In any case, after completion of these tests, the MSC updates its Program Counter by 1 and continues with the next instruction.

If OPX equals 001, the instruction performs a queue-overflow test by placing the number of memory requests specified by the COUNT field in the queue. If the queue overflows, the channel interrupts the CPU and the request is lost. The program loops on this instruction.

If OPX equals 010, the instruction attempts to execute a bad parity microword. This sets the HALT bit of all processors and sets BUSY/WAIT to WAIT.

If OPX equals 011, the Diganostic 25 Error Latch is set, causing an External 1 Interrupt to the CPU, with the appropriate interrupt code in the Group 2 Interrupt Register.

NOTE: Diagnostic Processor 25 should not be enabled while self-test is running. MSC self-test modifies Proc 25's locations in local store which results in IOP diagnostic errors.

4.0 EXTERNAL CALLS

The MSC allows the CPU to request the MSC to suspend execution of its current program and to start another program. When this action is undertaken, the MSC saves enough information to allow a full restart of the displaced program at the point of suspension. The stored information is similar to that stored by the CPU when it is interrupted. However, unlike an interrupt, the CPU specified program will only be recognized by the MSC at the discrete instances when the MSC is executing a Sample For External Call (@SEC) instruction. The CPU requested superseding program is termed an 'external call'.

The contact point between the MSC and the CPU is the MSC Local Store Location C6 - The External Call Register (ECR). When the CPU wishes the MSC to perform an 'external call', it should perform a PCO to write the starting address of the requested program into the MSC. If the MSC has been programmed to accept external calls, it will periodically execute an @SEC that will sample the ECR. If it finds a 0 in the ECR, it will continue with the next instruction. In this case, the @SEC is equivalent to a "no operation".

If the MSC finds a non-zero ECR, it uses the value in the ECR as a memory address and stores its Accumulator, Index, Program Counter, and Status Registers in the four words following this memory address. It will also clear the Status Registers and Program Exception bit of any error flags that they might contain. Program execution begins at the fifth fullword after the ECR (ECR+8). The MSC will also clear the ECR to zero. Register storage is defined in this document's description of the @SEC instruction.

The clearing of the ECR by the MSC may be used by the CPU as an indication that the MSC has accepted an external call, and that the CPU is now free to PCO out a new external call address, if it so desires. Any attempt by the CPU to PCO out a new external call before the MSC resets the ECR to zero will result in at least one of the external calls being lost. Therefore, the CPU should check that the ECR is zero before PCO'ing a new external call address.

An externally called program may return to the original program at any time by executing a Return from External Call (@REC) instruction. This instruction reloads the MSC Accumulator, the Program Counter, the Index Register, the Status Register, and the Program Exception bit to the exact condition that they were in at the time the original @SEC instruction accepted the External Call. This allows the original MSC program to continue as if no external call had been accepted.

The description of the @REC instruction indicates which memory words are loaded into what MSC registers.

The reason for clearing the MSC Status Register and Program Exception bit to an error free indication, after an @SEC has found a non-zero ECR and before entering the requested program, is to avoid confusing an original program error with any error the external call program may make. If the externally called program performs any erroneous operations (such as attempting to start a busy BCE) the error will be flagged in the Status Register and may be detected and diagnosed in the usual fashion. Yet, when the MSC returns to the original program, via an @REC, the Status Register will contain only the original program error indicators and none from the externally called program. The original program can proceed with no concern for sorting out what, if any, errors it made and what, if any, errors any externally called program may have made.

As mentioned before, the MSC will accept external calls only when it executes @SECs. Consequently, if external calls are expected, the operational MSC programs should include periodic @SEC's in places where they will give timely response to an external call and yet not unduly delay the execution of the main function. Since an @SEC normally acts as a no operation (or a @DLY 0), a convenient location for an @SEC would be just before a long format instruction in a situation where an extra short instruction is needed to align the long format instruction on a fullword boundary.

The only place where an @SEC should be avoided is within the body of an externally called program. In this situation, the CPU may request a second execution of that same program before the first execution is complete. Since storage is available for only one set of return information, acceptance of the second external call would destroy the return information from the first call.

An MSC program designed for external calls may also be called by another MSC program in a fashion similar to that used for subroutines. The calling program would simulate an @SEC by initializing PGM, PGM + 2, and PGM + 4 to whatever values it desires upon return, and then do a @CALL PGM + 6. The @CALL will store the return address at PGM + 6 and start execution at PGM + 8, just as @SEC does. Note, however, that upon return from the @REC the MSC Status Register and Program Exception bit will be set according to what is in fullword PGM, which if not initialized correctly may not match the original status at the time of the call. Figure 4.1 diagrams a segment of MSC code to do this set-up properly. Note that three of these instructions can be eliminated if the contents of ACC and X are of no concern after return from PGM.

```
@ST         PGM + 2        Save the Accumulator
@TXA        Zero
@ST         PGM + 4        Save the Index Register
@LMS
@ST         PGM            Save the Status Register
@CALL       2, PGM + 6     Start the routine
```

Figure 4.1. MSC Code to Call an "External Call" Routine

APPENDIX A

IOP MSC INSTRUCTION REPERTOIRE

IOP MSC Instruction Repertoire (Cont.)

Appendix III

Input/Output Processor (IOP) — Principles of Operation for Bus Control Element

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 BUS CONTROL ELEMENT

The Bus Control Element (BCE) is a microprogrammed controller specifically tailored for management of I/O traffic on one of the Space Shuttle system busses. Within each IOP there is one BCE for each system bus, for a total of 24 BCE's. Each of these BCE's is capable of independent program execution, data buffering to and from memory, and communication with the MSC. Further, each BCE is connected to its own bus via its own Multiplexer Interface Adapter (MIA), which performs all parallel to serial and serial to parallel conversions. Table 1.1 summarizes the basic characteristics of a BCE.

The major purpose of a BCE is threefold:

(1) Initiate transmission of commands to subsystems on the bus.

(2) Handle data coming back from a commanded subsystem.

(3) Fetch data to be sent to a commanded subsystem.

To handle these tasks there are two classes of instructions (transmit and receive), and two special operating modes (Command, and Listen) that are unique to the BCE.

The transmit instructions allow transmission of both commands and data to a subsystem. When transmitting data a BCE/MIA pair performs:

(1) Update of main memory buffer addresses.

(2) Conversion between 32 bit main memory data format and 25 + Sync bit bus data format.

(3) Check on number of words to be transferred.

The receive commands allow a BCE to accept a stream of input data from a subsystem through its MIA. When receiving data a BCE performs:

(1) Time outs on data arrival.

(2) Error checks on incoming data.

(3) Assembly into main memory 32 bit data format.

(4) Maintenance of main memory buffer addresses.

(5) Transferral of data to main memory.

(6) Check on number of words to be received.

The two operating modes that a BCE may be in influence the way the BCE uses its bus. In Command mode, a BCE is master of its bus, and is free to transmit both commands and data. This allows a BCE to command a subsystem, receive data from it, or transmit data to it. In Listen mode a BCE monitors its bus for directions on how to handle any data that might appear on the bus. In this mode a BCE may only receive data, and may not transmit either commands or data. This handles the common situation in the Space Shuttle where several IOP's and thus several BCE's may be connected to one bus. In such a situation only one BCE is allowed to issue subsystem commands, but all BCE's on that bus wish to receive copies of the resulting data. The listening mode allows the command BCE to tell the others what data to expect, and when to expect it.

TABLE 1.1

BCE CHARACTERISTICS

Type -- Programmable I/O traffic controller
Number -- One per bus, 24 BCE's per IOP
Control Structure -- Microprogrammed

Programmable Registers (per BCE)

18 Bit Base Register (BASE)
18 Bit Program Counter (PC)
18 Bit Maximum Time Out Register (MTO)
5 Bit Interface Unit Address Register (IUAR)
1 Bit BCE/MSC Indicator Bit

Other BCE Registers (per BCE)

32 Bit Status Register
1 Bit Program Exception Register (part of STAT 1)
1 Bit Busy/Wait Bit (part of STAT 4)
1 Bit MIA Transmitter Enable
1 Bit MIA Receiver Enable
6 Bit Identify Register

Instruction Formats : 16 Bit Short/32 Bit Long/ 64 Bit Extended

Instruction Repertoire: 10 Short/5 Long/ 2 Extended

Addressing Space: 131,072 32 Bit Fullwords/262,144 16 Bit Halfwords

Addressing Modes: Immediate, PC relative, Base relative, Absolute

Special Operating Modes: Command, Listen

Bus Data Format: 25 + Sync Bit serial

1.1 FORMATS

1.1.1 Bus Formats

There are four basic formats for data or commands carried over a system bus in serial form. These are pictured in Figure 1.1. They are all 28 bits long, with 3 bit times for sync, 24 bits of information, and 1 bit for word parity. These 28 bits are transmitted at a serial rate of 1 bit per microsecond.

On transmission from an IOP to a subsystem, a BCE provides the middle 24 bits of information and an indication of the type of sync to use -- either command or data sync. Command sync is used when the 24 bits of information are to be treated as a command to be obeyed by some subsystem. Data sync is used when a BCE has previously conditioned a device, via a command, to accept a stream of data, and the word being put on the bus is a member of this stream.

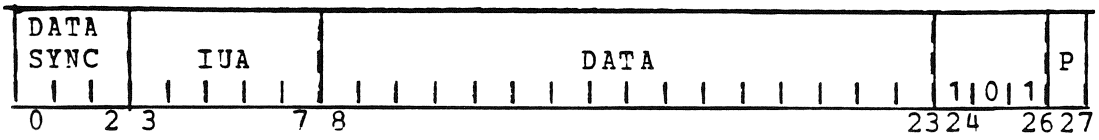
Conversely, there are situations where a BCE may accept, through its MIA, words with either command or data sync. In such cases the MIA simply provides the type of sync that a word had, the 24 bits of information present in the word, and appropriate error signals. Command sync is recognized when a BCE is in the "Listening Mode" and is expecting a command from another BCE connected to the same bus. Data sync is accepted when a BCE is expecting to receive a stream of data from a subsystem on the bus. This subsystem was previously commanded to send this data by a message with command sync from some BCE on the bus.

The contents of the 24 bits of information in a bus word depends both on the sync type and the direction of transfer -- from a BCE or to a BCE. In all cases the upper 5 bits contain an interface unit address (IUA). When used in data words or commands to subsystems, these 5 bits identify the subsystem on the bus who either originated the data or is to accept the command or data. In Listen Mode commands, the BCE in one IOP sends to all other BCE's in the other IOP's connected to this bus a command that has a "Common IOP address". This special pattern is not used by any subsystem, and allows a listening BCE to distinguish between listen commands and commands to subsystems.

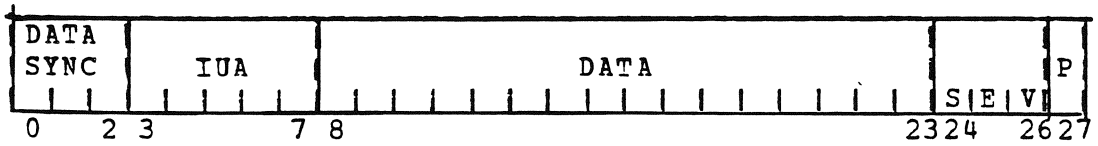
In data transfers the remaining 19 bits consist of 16 bits of data (one half of a standard 32 bit main memory fullword) and 3 bits nominally containing the pattern 101. In BCE to subsystem data transfers, this pattern never changes. However, the subsystem to BCE data transfers, any variation in the 101 pattern indicates to the BCE that the sending subsystem has encountered a problem, such as power, invalid commands, etc.

In commands sent by a BCE to a subsystem, the format of the lower 19 bits of information is subsystem dependent, and not discussed further here.

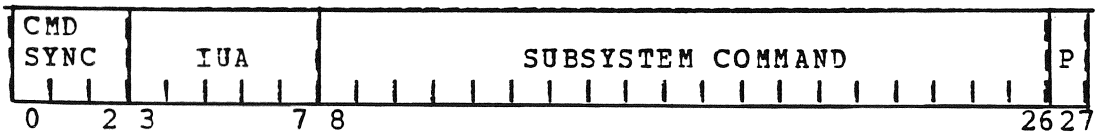
In Listen commands sent from one IOP to another, the 18 bits of the information contain a 5 bit number representing a terminal or subsystem on the BCE's bus and an 8 bit index into a table of BCE branch addresses.



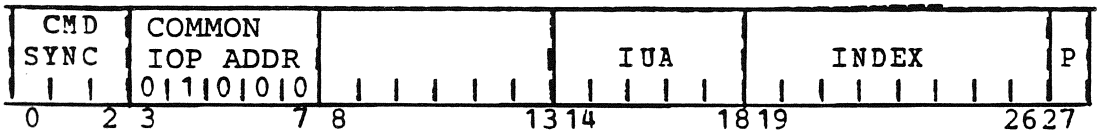
(a) DATA FORMAT -- FROM BCE TO SUBSYSTEM



(b) DATA FORMAT -- FROM SUBSYSTEM TO BCE



(c) COMMAND FORMAT -- BCE TO SUBSYSTEM



(d) COMMAND FORMAT -- BCE TO BCE (LISTEN MODE)

IUA = INTERFACE UNIT ADDRESS

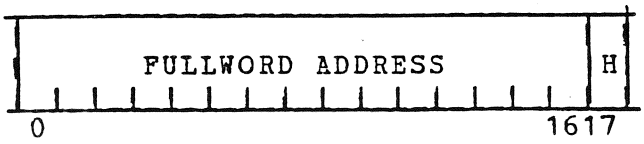
Figure 1.1. Bus Message Formats

1.1.2 Addressing and Instruction Formats

A BCE may directly address up to 262,144 16-bit halfwords or 131,072 32-bit full words.* To achieve this, all main memory addresses computed by the BCE are represented as 18-bit absolute numbers, as pictured in Figure 1.2. The upper 17-bits (bits 0 through 16) represent the fullword location, and the lowest bit (bit 17) the halfword portion of the addressed fullword. A 0 in this lower bit refers to bits 0 - 15 of the 32-bit fullword; a 1 refers to bits 16 - 31. When used as a fullword address, bit 17 is ignored. Thus, H'276' and H'277' refer to the same fullword.

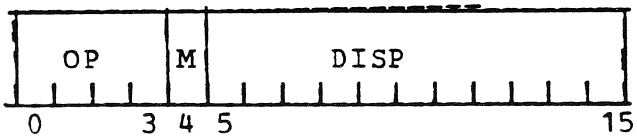
There are four basic instruction formats used by the BCE and they are depicted in Figures 1.3 and 1.4. Both data fullwords and fullword instructions must reside at a fullword address.

*Note that the AP101S performs 19 bit addressing, and can address 524,288 halfwords. Only the first 256KHW of these, those where the most significant CPU addressing bit = Ø, are addressable by the IOP.



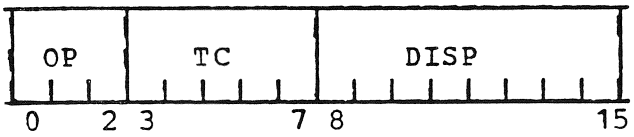
H = HALFWORD

Figure 1.2. BCE-Computed Main Memory Addresses



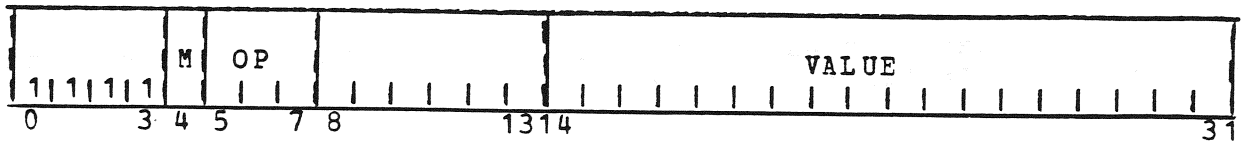
M = INDEX SPECIFICATION OR OPX

(a) SHORT FORMAT 1

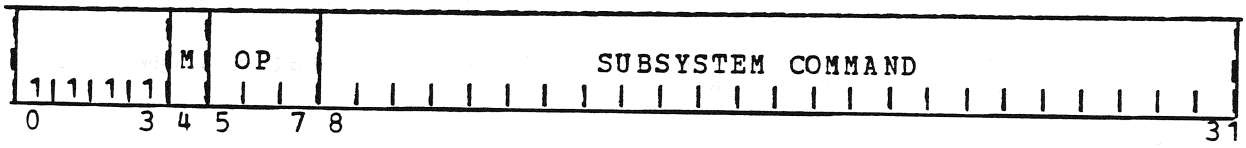


(b) SHORT TRANSMIT/RECEIVE DATA FORMAT

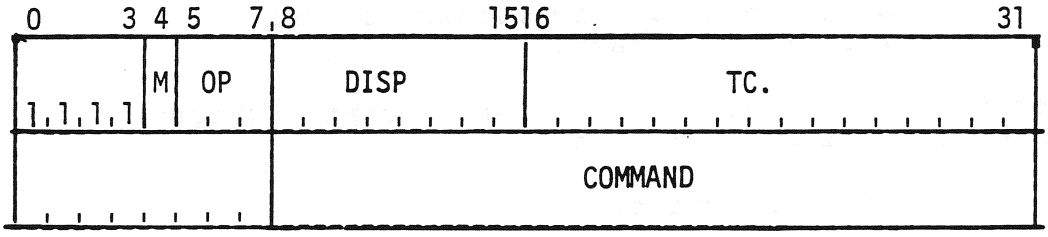
Figure 1.3. Basic BCE Short Instruction Formats



(a) STANDARD FORMAT



(b) COMMAND INSTRUCTION FORMAT



(c) EXTENDED FORMAT

Figure 1.4. BCE Instructions - Long Formats

Short format 1 is used primarily by instructions dealing with the BCE register. It has the following fields:

Field	Field Description
OP	This 4-bit field defines the basic operation to be performed by the BCE.
M	This bit serves either as an opcode extension or as an index mode specification in address generation.
DISP	This 11-bit field serves either as immediate data or as a PC relative address displacement. The PC used is the updated BCE Program Counter.

Short format 2 is used by instructions that transmit and receive data. It has the following fields:

Field	Field Description
OP	This 3-bit field defines the basic operation to be performed (data read or write).
TC	Transfer Count. This field defines the number of inputs or outputs to be handled.
DISP	This 8-bit field serves as a displacement off of the BCE's base register in the computation of the main memory buffer addresses associated with the I/O data.

Most long format instructions use long format 1 (Figure 1.4). Long format provides the following fields:

Field	Field Description
OP	This 3-bit field defines the basic operation to be performed by the BCE.
VALUE	This 18-bit address is used either for immediate data or as an address.
M	This bit influences how the Value field is used.

Long format 2 is used by the Transmit Command instruction to provide 24 bits to be given to the MIA for command transmission.

Extended format 3 is used by the Message In/Out instructions to specify Displacement, Transfer Counts and Commands.

In many BCE instructions the direct addressing mode includes automatic indexing by twice the BCE's number. This allows BCE programs to be written in a table driven fashion, where the same BCE program can be used by many different BCE's, and yet each BCE will use a separate set of parameters in the programs execution. The assembler recognizes "(1)" following a BCE operand as specifying the BCE number indexing.

1.2 BCE REGISTERS

1.2.1 BCE Programmable Register

Each BCE contains several registers under direct program control. They are:

BASE --	An 18-bit Base Register
PC --	An 18-bit Program Counter
MTO --	An 18-bit Maximum Time Out Register
IUAR --	A 5-bit Interface Unit Address Register
Indicator --	A 1-bit indicator bit in the BCE/MSC Indicator Register.

The BASE is used in locating I/O buffers in main memory. All such buffers are base-relative, allowing the same BCE program to be used with different buffers by simply changing the Base before entering the common program segment.

The Program Counter is an 18-bit register indicating the main memory halfword or fullword location of the BCE instruction being executed by this BCE. It should be emphasized that there is a separate PC for each BCE, and that the contents of one PC need not have any relation to the contents of a different PC in another BCE.

The Maximum Time Out Register is used primarily during the reception of data to indicate the maximum time a BCE should wait for a subsystem to respond to a command with the beginning of a stream of input data. This time is defined as the "latency" of the subsystem. Failure of a subsystem to respond within this period of time results in a BCE declared error condition.

The Interface Unit Address Register contains the 5-bit subsystem address of the subsystem presently in communication with the BCE. This address is derived when a command is issued by a BCE (Figure 1.1(c)), and is used in the construction of data words to be sent to a subsystem (Figure 1.1(a)), and in checking for proper subsystem response when data words from a subsystem are being received (Figure 1.1(b)).

The BCE/MSC Indicator Register has 1 bit associated with each BCE. A BCE is free to set or clear this bit under program control. The MSC can read all such bits and monitor when various BCE's have set or cleared their bits. The MSC may also reset a BCE's Indicator bit via an @RBI instruction. This provides a means for BCE's to signal to the MSC the occurrence of various events such as execution of listening mode programs.

The detection of various errors will also set a BCE's Indicator bit to 1 (See Paragraph 2.2).

1.2.2 BCE Status Registers*

The modes and status of each BCE is recorded in the following registers:

(1) BCE Busy/Wait Bit -- For BCE i this is bit i of the Busy/ Wait Register (STAT 4). A 1 in this bit indicates that the BCE is busy executing a program located in main memory. A 0 indicates it is not busy.

(2) BCE Program Exception Bit -- For BCE i this is bit i of the Program Exception Register (STAT 1). A 0 indicates that the BCE has encountered some problem in the execution of a BCE program. A 1 indicates that no problem was encountered. The Status Register associated with that BCE contains a description of the cause of the problem.

(3) Transmitter Enable Bit -- For BCE/MIA i this is bit i of the MIA Transmitter Enable Register. A 0 in this bit prevents the BCE from issuing a command or transmitting data over the bus to which its MIA is connected. This bit may be changed only by a PCO from the CPU.

(4) Receiver Enable Bit -- For BCE/MIA i this is bit i of the MIA Receiver Enable Register. A 0 in this bit prevents the MIA from transferring any data or commands it receives on the bus to the BCE. This bit may be changed only by a PCO from the CPU.

(5) BCE Status Register. Each BCE maintains a unique 32-bit status register describing what, if any, errors the BCE has encountered during the execution of a program. Table 1.2 describes the format used in these registers.

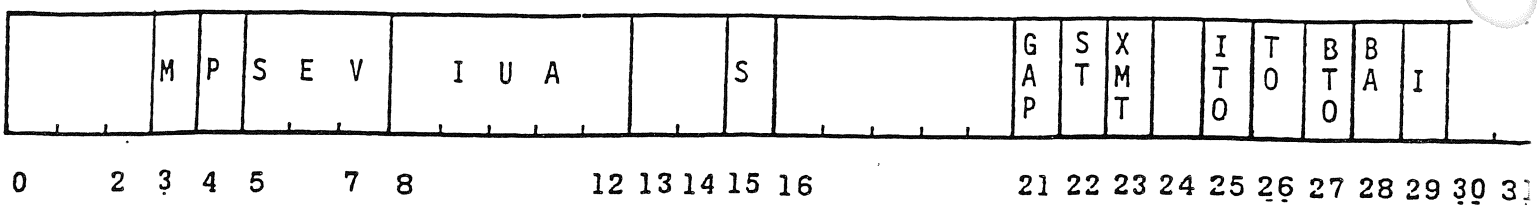
(6) BCE Identity Register. Each BCE maintains a register that contains twice its own BCE number. This register is used in computing addresses during direct mode BCE instruction. The register is set while the BCE is leaving the Halt state and entering the Wait state, and is not altered by the BCE at any other time.

*PROGRAMMING NOTE

The BCE Status Registers should not be changed via PCO commands while the BCE is busy.

TABLE 1.2

BCE STATUS REGISTER



BIT		
31 - 30	Reserved	
29 - I	Illegal Opcode	-- This BCE encountered an illegal instruction in the execution of a program.
28 - BA	Boundary Alignment Error	-- This BCE encountered a long format instruction on an odd halfword boundary.
27 - BTO	Block Time Out	-- A Receive Data Instruction timed out while waiting for an interblock gap to end. This is a Modulo 512 word timeout for mass memory operations.
26 - TO	Time Out	-- A Receive Data Instruction timed out while waiting for a data word to arrive. This time out occurred on data inputs other than the first. See Sections 3.4.1 through 3.4.8 for receiver error details.
25 - ITO	Initial Time Out	-- A Receive Data Instruction timed out while waiting for the first input word to arrive. See Sections 3.4.1 through 3.4.8 for receiver error details.
24	Reserved	
23 - XMT	Transmitter Disabled	-- At some point in the execution of a Transmit Data. Message Out or Message In instructions the MIA associated with this BCE had its transmitter disabled. This bit also is set if the MIA was found busy when it was time to initiate transmission of command word or new data word.

22 - ST	Self Test Error--A BCE Self Test Instruction has detected a fault in the BCE.
21 - GAP	Gap of greater than 21.5 usec. occurred during execution of a transmit data instruction, or 5 usec during a MOUT.
20 - 16	Reserved.
15 - S	Sync Error -- While executing a Receive Data Instruction, an input word with command sync was received. (See #RDS Instruction)
14 - 13	Reserved
12 - 8	Subsystem Address -- This field is the logical "OR" of the received subsystem addresses of all input words that were detected to have errors during execution of previous Receive Data Instrs.
7 - 5	SEV -- This field is the logical "OR" of the SEV bits of all input words that were detected to have errors during execution of previous Receive Data Instruction. The S and V bits were inverted before the "OR". Thus, any pattern other than 101 will be recorded in these bits.
4 - P	Parity -- While executing a Receive Data Instruction, an input word with bad parity was detected.
3 - M	Signature Mismatch -- While executing a Receive Data Instruction, a mismatch between the

input's IUA and the BCE's IUAR. The input IUA was simultaneously 'OR'ed into bits 8 - 12 of the status word.

2 - 0	Reserved.

1.3 BCE IMPLEMENTATION

Any BCE implemented within an IOP except BCE 25 (self-test) consists of:

(1) A segment of Local Store. This consists of 4 words from each of Banks A and B, and 8 words from Bank C.

(2) A bit in the IOP Busy/Wait Register.

(3) A bit in the IOP Program Exception Register.

(4) A bit in the IOP Halt Register.

(5) A bit in the BCE-MSC Indicator Register.

(6) The associated MIA.

(7) A buffer location in the MIA Buffer.

(8) A bit in the MIA Transmitter Enable Register.

(9) A bit in the MIA Receiver Enable Register.

(10) One of the Microprogram Counters. This contains the ROM address of the next micro instruction to be executed for this BCE.

As described in the IOP Functional Description (IBM No. 74-A31-016;), the operation of all BCE's and the MSC is time-shared. Each BCE executes its next microinstruction from ROS at intervals of one every 16.5 microseconds. During other periods of time the only BCE-related operations still carried on are previously requested memory operations, MIA reception/transmission of data, and CPU directed PCI/O.

The general makeup of a BCE's Local Store segment is shown in Figure 1.5. Several of these locations are BCE programmable registers, and several are accessible by the MSC. All local store locations are available to the CPU via PCI and PCO. The MSC cannot load BCE local store unless the BCE is in the wait state. In general, the CPU should not alter local store while the BCE is busy.

Self-test processor BCE 25 consists of:

(1) A segment of Local Store. This consists of 4 words from each of banks A and B and 8 words from bank C.

(2) A bit in the IOP halt register.

(3) One of the microprogram counters. This contains the ROM address of the next micro instruction to be executed for this BCE.

BANK A	BANK B	BANK C	WORD
AND	WR	WR	0
RWC	WR	WR	1
PC	IH	IL	2
ID	MTO	BASE	3
		WR	4
		IUAR	5
		STATUS HIGH	6
		STATUS LOW	7

PC = Program Counter
IH = High half of Instruction Register
ID = BCE Identify Register
IL = Low Half of Instruction Register
MTO = Max. Time Out Register
BASE = Base Register
IUAR = Interface Unit Address Register
STATUS HIGH = High Half of Status Register
STATUS LOW = Low Half of Status Register

AND = Address of Next Data
RWC = Residual Word Count
WR = Working Register

Figure 1.5. BCE Local Store Usage

2.0 GENERAL BCE OPERATION

During normal operation a BCE can be in one of three states: Halt, Wait and Busy. In the Halt state the BCE is physically restrained from performing any operations; in the Wait state the BCE is awaiting a command to execute a program; and in the Busy state the BCE is executing BCE programs from main store. Figure 2.1 summarizes these states and the transitions between them.

Typical state transitions are as follows:

(1) During any system or CPU-directed BCE reset, the BCE is in the Halt state.

(2) Upon release from the Halt state, the BCE enters the Wait state.

(3) A signal from the MSC initializes a BCE, and places it in the Busy state.

(4) In the Busy state, the BCE is executing a program located in main memory. It exits the Busy state only upon execution of a Wait instruction, detection of an invalid instruction or operating error, or some reset signal. In all but the latter case, the BCE re-enters the Wait state; in the latter case it is forced to the Halt state.

The two bits that indicate the current state of BCE i are its Halt bit (Bit i of the IOP Halt Register) and its Busy/Wait bit (Bit i of the IOP Busy/Wait Register). In addition, the BCE Program Exception bit indicates if the BCE found an error while in the Busy state. If it has found an error, the BCE Status Register contains a record of the exact error.

The following sections describe each state in detail.

2.1 HALT STATE

The primary purpose of the Halt state is two-fold:

(1) Allow the external world to reset BCE operation to a known condition.

(2) Upon the detection of very serious IOP faults (such as failure in the microstore) to isolate the BCE and prevent it from performing potentially erroneous operations.

Entry and exit from the Halt state are controlled by the value of a single "BCE Halt" status bit which is part of the IOP Halt Register. There is one such bit for each BCE. As long as this bit is zero, the BCE microprogram counter is forced to point to a micro-

instruction that performs no operation other than clear the BCE Busy/Wait bit. The Halt bit may be set to 0 at any time, and effectively terminates anything that the BCE is doing.

The signals that set a BCE Halt bit to 0 (Halt) include:

(1) BCE/MSC disable discrete (CPU internal DO 1).

(2) Power-Up/Down Signal, Discrete Input 0 (HALT/System Reset)

(3) IOP detected serious errors such as ROS parity error.

(4) A Master Reset PCO from the CPU.

(5) A "Halt Processor" PCO from the CPU with a 1 in position i (when BCE i is to be halted).

The first four signal classes halt all BCE's.

Exit from this state to the Wait state occurs only when the following occurs:

(1) BCE halt is reset.

(2) Power-Up sequence is complete

and a CPU "Enable Processors" PCO with bit i (for BCE i) set to 1 is present. The PC I/O manual (Appendix I) describes the PCO that set/reset the Halt register.

Upon any exit from the Halt state, the BCE Program Exception bit is set to one (no errors). The BCE Status Register is cleared to all zeroes, and the BCE Identity Register is set to twice the BCE's number.

Although a BCE's Halt bit may be changed at any time, it does not affect the BCE's operation until the next time that the BCE executes a microinstruction. Consequently, to guarantee that a BCE has been halted, the BCE's Halt bit should be at 0 (halted) for a minimum of 16.5 usec. before being reset to 1 (enabled). This will guarantee that the BCE has truly been forced to the halt state for at least one BCE microcycle.

For similar reasons, a BCE should not be considered released from the Halt state and in the Wait state the instant its Halt bit is set to 1 (enabled). Release from the Halt state does not begin until the first BCE microcycle after the Halt bit has been reset, and continues for several BCE microcycles (about 100 usec.) as the BCE resets its internal registers and prepares to enter the Wait state. After this transition, the BCE will be in the Wait state, and will perform as described in the next section.

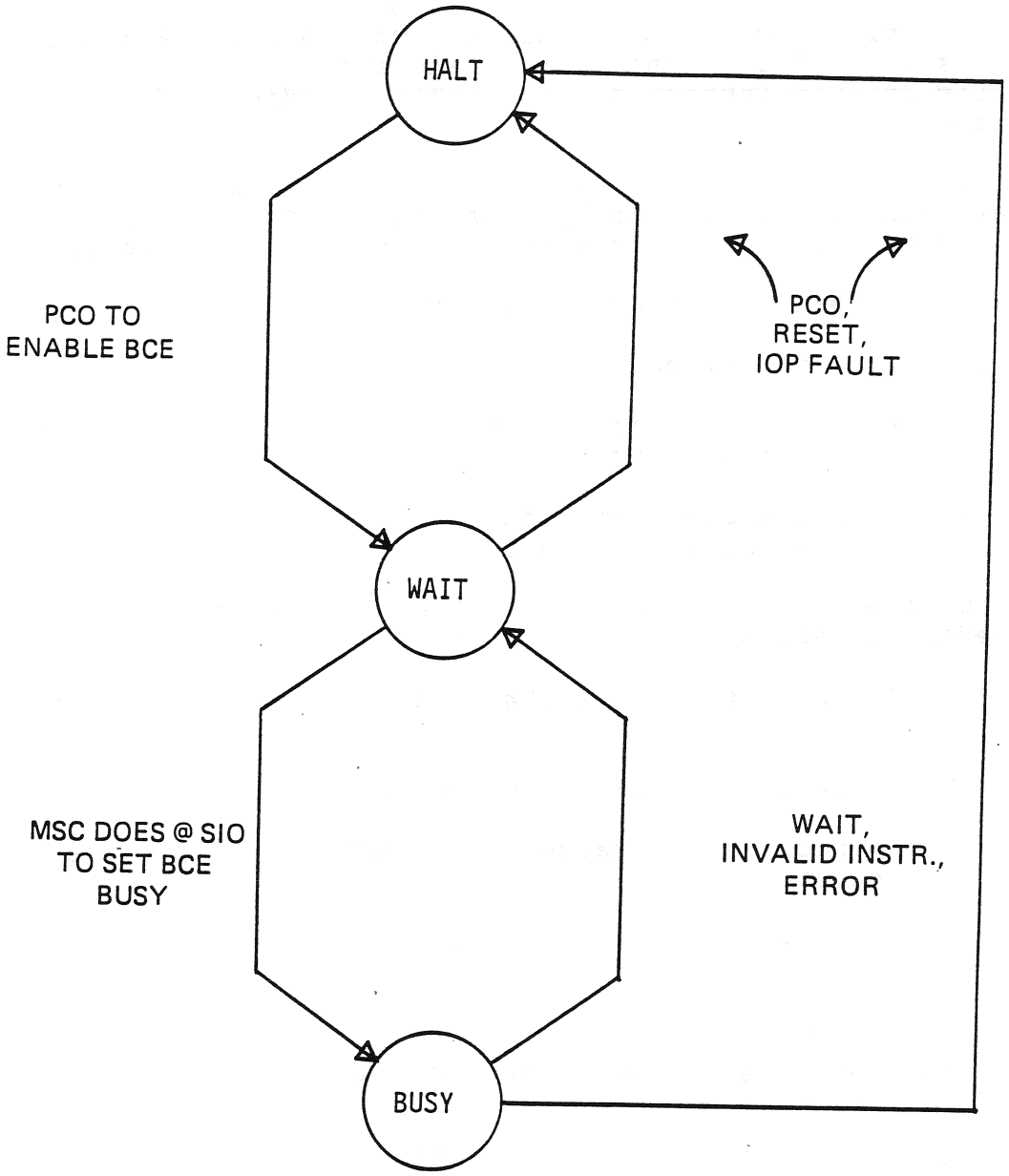


Figure 2.1. BCE States

2.2 WAIT STATE

In the Wait state, a BCE is prepared to be told to perform a BCE program. This wait loop is implemented by a micro-routine that monitors the status of the BCE's Busy/Wait bit. The BCE remains in the Wait state as long as its bit is reset to the "Wait" value (0). When the MSC (via an @SIO) sets this bit to busy, the BCE transits to the Busy state. This transition consists of using the present value of the BCE's Program Counter as the starting address of a BCE program in main store.

Entry to the Wait state occurs either upon exit from the Halt state, as described above, or by a transition from the Busy state. A BCE performs this latter transition when any of the following occurs:

(1) A "Wait" instruction is executed.

(2) An instruction with an illegal opcode is encountered.

(3) A valid long format instruction is found starting on an odd halfword boundary.

(4) A significant error occurs. (such as in transmit or receive data instruction).

In the final three cases, a BCE performs the following actions before entering the Wait state:

(1) It sets its bit in the Program Exception Register (STAT 1) to 0.

(2) It sets its BCE-MSC Indicator bit to 1.

(3) It records the cause of the error in its own BCE status register.

(4) It leaves the Program Counter pointing to the offending instruction.

The BCE's Indicator bit is set, upon detection of an error, to aid in handling programs where this bit is used to signal completion of some operation to the MSC (via a Repeat on Indicator instruction).

Exit from the Wait state is normally to the Busy state. This is done, by the MSC, via a sequence of MSC instructions that can include:

1) A "Load BCE Base" instruction to set the BCE's Base register.

2) A "Load BCE Program Counter" instruction to set the BCE's Program Counter.

3) A "Start I/O" instruction to set the BCE's Busy Wait bit.

Note that the CPU can perform the equivalent of the first two functions via PCO's to the BCE's Local Store. Also, via PCO's, the CPU can reset the BCE's Status Register and Program Exception bit. The CPU cannot, however, directly set a BCE's Busy/Wait bit.

Note also that a BCE's Program Counter need not always be set before the MSC sets the BCE to busy, since the BCE Wait instruction (#WAT) when executed, leaves the PC pointing to the next sequential instruction. This next instruction may be programmed as a simple branch to the beginning of the next BCE program segment. In this case, the MSC need only execute an SIO instruction to restart the BCE at the next segment.

While a BCE is in the Wait state, the CPU may perform PCI/O activity without disturbing the BCE.

2.3 BUSY STATE

In the Busy state, BCE i is in the process of executing a program out of main store. A value of 1 in the ith bit of the IOP Busy/ Wait Register indicates this condition.

The Busy state may be entered only from the Wait state, as described in the previous section. When the transition occurs, the BCE uses the value in its Program Counter to fetch the first instruction from memory and start executing it. The BCE continues executing instructions until either a Wait instruction or some kind of invalid instruction is encountered. In either case, the BCE transitions back to the Wait state, again as described in the previous section.

Although an MSC @SIO instruction will set a BCE's Busy/Wait bit to busy within the time frame of the instruction's execution on the MSC, the BCE will not necessarily begin immediate execution of the first instruction. The possible delays include:

- The time required for the BCE to recognize that its Busy/Wait bit is set (up to 16.5 usec.)
- The time required to bring out the PC and request the instruction word from memory (up to 33 usec) and,
- Any time required for the memory read request to reach the DMA channels, be read from memory, and be returned to the BCE.

The memory read request time delays the BCE setup only when handling time exceeds 16 usec. This situation can occur when several BCE/MSC processors have requested memory simultaneously.

While the BCE is in the Busy state, the CPU may execute any PCI without problem. However, all PCO's that write into BCE Local Store or Status Registers should be carefully controlled since the processors are not aware that this action has occurred, and the resulting BCE program execution may be unpredictable.

2.3.1 Busy State Operating Modes

Once in the Busy state a BCE may operate in one of two modes; Command and Listen. In the Listen Mode the BCE's MIA receiver is enabled, but the BCE's MIA transmitter is disabled. In Command mode both transmitter and receiver are enabled.

The BCE Mode affects the way certain BCE instructions are executed. In the Command Mode, a BCE is in command of its bus and is free to transmit commands and data over the bus. A BCE in the Listen Mode relies on a BCE in another IOP to issue the appropriate commands to subsystems on the bus to perform the operations. Consequently, a BCE in Listen mode will handle instructions that transmit commands in a different manner than a BCE executing the same program in the command mode. Additionally, a BCE in the Listen mode can use a Wait for Index instruction as an instruction to wait for some other BCE in another IOP to provide a signal as to what it should do. This signal is used by the Listening BCE in a table-look up process to determine an appropriate BCE program for execution.

Section 4 describes the differences between Command Mode and Receive Mode in greater detail.

2.3.2 Summary of Error Modes

Table 2.1 summarizes all the errors detectable during a BCE program and the resulting actions. Separate columns indicate which, if any status bits are set, and whether or not the BCE enters the wait state. If any status bits are set, the BCE Program Exception bit is also assumed set to 0, and the BCE/MSC Indicator bit is set to 1.

TABLE 2.1

SUMMARY OF BCE-RELATED ERRORS

Error	Detected By	Action: Status Bit Set?	Action: Wait State Entered?	Other Action
Illegal BCE Instruction	Microcode	Bit 29	Yes	
Long Format BCE Instruction on Odd Boundary	Microcode	Bit 28	Yes	
Bad Command Received in #WIX	Microcode/ MIA	None	No	Command ignored. Wait continued.
DMA Parity Error on Instruction Read	Channel Logic	(see Illegal Instruction)		Write all 0's into Instruction register (illegal opcode) CPU interrupt.
DMA Parity Error, DMA Timeout, or Queue Overflow on Data read for Transmit Data Instruction	IOP Hardware/ Microcode	21	Yes	CPU Interrupt
DMA Parity Error, DMA Timeout, or Queue Overflow on other data reads	IOP Hardware	None	No	CPU Interrupt. BCE does not receive data.
ROS Parity Error, Clock Failure,	IOP Hardware	No	No	BCE Reset to halt. CPU Interrupt

Error	Detected By	Status Bit Set?	Action: Wait State Entered?	Action: Other Action
Self Test Detected BCE Fault	Bce Self Test Instruction	Bit 22	No	BCE Loops
Excessive Concurrency resulting in > 20 usec gap in transmit data, or 5 usec in MOUT	Microcode Time out	21	Yes	
Transmitter Disabled during transmit data	Microcode Test	23	Yes	
Transmitter Busy too long	Microcode Test	23	Yes	
Parity Error on input data, invalid Manchester, or bit count error	MIA	4	Yes	Save IUA from data. Data not stored in MIA Buffer.
Wrong SEV bits	Microcode/ Hardware	5,6,7	Yes	" "
Command Sync in Receive Data	Microcode	15	Yes	" " (See #RDS)
Wrong IUA in incoming data	Microcode/ Hardware	3	Yes	" "
Failure of Subsystem to respond with data (1st word)	Microcode time-out	25	Yes	

Error	Detected By	Status Bit Set?	Action: Wait State Entered?	Action: Other Action
Excessive interword gap in incoming data	Microcode time out	26,27	Yes	
IOP Data Flow Parity Error	Hardware	None	No	All BCE's and the MSC are halted, CPU is interrupted and interrupt register shows data flow error transmitter and receiver enables for all BCE's are disabled, and the discrete outputs are reset
MIA busy when asked to transmit.	Microcode	Instruction Specific	Instruction Specific	Note: See transmit instructions description

3.0 BCE INSTRUCTIONS

The following pages include descriptions of the instructions presently supported by each BCE. The format for the description of each instruction is as follows:

- The general name of each instruction appears in the upper left of the first page describing that instruction.
- The assembler abbreviation appears in the upper right hand corner.
- The format of the instruction, including binary opcode assignments and field designations.
- A table (where appropriate) relating addressing mode bits to their effect on parameters used in the instruction execution, and how these addressing modes are signalled to the assembler.
- A textual description of the instruction and its uses.
- The minimum instruction execution time will not be presented here as it is supplied in Document No. 74-A31-00009A.

Since the IOP is a multiprocessor with 25 BCEs and the MSC requesting memory access independently of each other, it is possible that a memory request from a BCE can be delayed due to the servicing of earlier requests from other IOP processors. If these delays become significant (become greater than about 16.5 usec) then the BCE will be forced to wait for the memory operation to complete in increments of 16.5 usec. This will of course increase the instruction times.

These descriptions of instructions are grouped into several separate classes with a short prologue at the beginning of each class. These classes include:

3.1 BCE REGISTER INSTRUCTIONS

The BCE instruction set has several instructions that allow it to modify or store many of its registers. These registers include the Maximum Time Out register, the BCE-MSC Indicator bit, the Base register, and the Status register. In each case, the register referred to by the instruction is the one owned by the BCE executing that instruction.

The formats for these instructions are primarily 16 bits of the form of Figure 1.3, with the exeception of the Load Base, which is a long format instruction (Figure 1.4). The Load Time Out and Load Base instructions use instruction bit 4 to indicate whether the addressing of the data to be loaded into the appropriate register is immediate or direct. For immediate, the data to be loaded is present in the instruction itself; for direct the instruction determines the address of the word containing data.

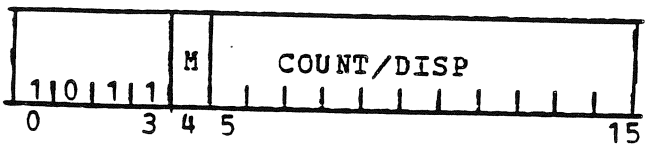
All instructions that have direct mode allow the specification of indexing by the current BCE number in the computation of the memory address to contain the status words. This permits the same instruction to be used by many BCE's since the index by processor number will construct a table of status words.

This group includes instructions to set or reset the BCE-MSC indicator bits. These bits have no meaning to the hardware, but can be read by MSC instructions, allowing them to be used as general-purpose flags by the BCE and MSC programs.

LOAD TIME OUT REGISTER

#LTOI, #LTO

FORMAT



M	Effective Count	INSTR. Format	
0	TIMEOUT[1]	#LTOI	COUNT
1	(PC+DISP+2 x BCE#)[2]	#LTO	ADDRESS

NOTES:

1. Any value between 0 and 2047. This corresponds to time outs from 0 to 33.78 millisec.

2. The lower 18-bits of the fullword addressed by PC(updated)+ displacement + 2 x BCE#. This allows any count between 0 and 262143, or 0 to 4.325 sec.

DESCRIPTION

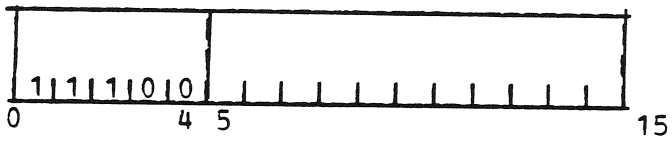
This instruction loads the Maximum Time Out Register (MTO) with the Effective Count. This register is used by the Receive Data instructions to determine how long a BCE will wait for the first input word to arrive from a previously commanded subsystem. The resolution of this timeout count is 16.5 microseconds.

After loading the Maximum Time Out register, the BCE increments its program counter by 1 and begins execution of the next sequential instruction.

PROGRAMMING NOTE

In direct mode, the computation of the effective address includes the number of the BCE executing the instruction. This allows many BCE's to execute the same BCE program, but, each one uses a time-out parameter best suited for the subsystem with which they are communicating. Note that two times the BCE number gives a fullword index.

Listening BCEs may decrement the ITO value to zero and declare an error before command BCEs, if the ITO value is borderline. A listening BCE executing a #RDLI instruction will begin decrementing its ITO value one 16.5 μs cycle after receiving the command issued by the commanding BCE. A commanding BCE executing a #MIN instruction takes three 16.5 μs cycles to begin decrementing its ITO value.



INSTRUCTION FORMAT

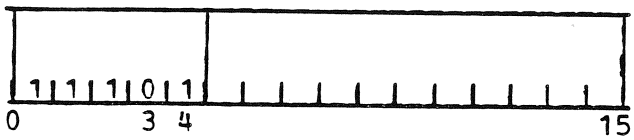
#RIB

DESCRIPTION

The BCE to MSC indicator bit associated with the BCE that is executing this instruction is reset to 0. This bit will not change to 1 until either a Set Indicator Bit (SIB) instruction is executed or, the BCE error terminates some instruction at a later time. After this bit is reset, the BCE's program counter is incremented by 1, and BCE program execution continues.

SET INDICATOR BIT

#SIB



INSTRUCTION FORMAT

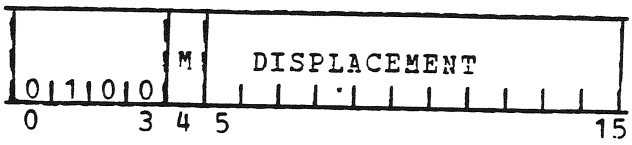
#SIB

DESCRIPTION

The BCE to MSC indicator bit associated with the BCE that is executing this instruction is set to 1. This bit will not change to 0 until either the BCE executes a Reset Indicator (#RIB) instruction or the MSC executes a Reset BCE Indicator (@RBI) instruction with this BCE's number. After this bit is set, the BCE's program counter is incremented by 1, and BCE program execution continues.

STORE STATUS AND CLEAR

#SSC



M	Effective Address (EA)	INSTR.Format	
0	PC + DISP.	#SSC	ADDRESS
1	PC + DISP + 2XBCENO	#SSC	ADDRESS (1)

NOTES: 1. PC is the updated Bus Control Element (BCE) Program Counter, i.e. the address of the next sequential instruction.

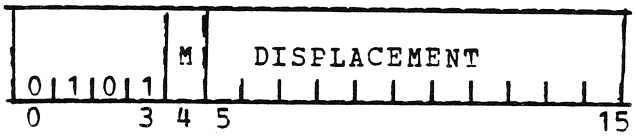
DESCRIPTION

This instruction stores the BCE's status register in the fullword location addressed by the Effective Address. The least significant bit of EA (the halfword address) is ignored. After initiating the store operation the BCE will then clear its status register and set its program execution bit (from STAT1) to 1 (GO). It will then increment its program counter by 1 and continue with execution of the next sequential instruction.

PROGRAMMING NOTE

In indexing mode the computation of the effective address includes the number of the BCE that is executing this instruction. This allows many BCE's to use the same BCE program, and yet store the status from each BCE in a different location. Note that the BCE number is multiplied by 2 to give a fullword index.

STORE STATUS #SST



M	Effective Address (EA)	INSTR.Format	
0	PC + DISP.	#SST	ADDRESS
1	PC + DISP + 2XBCENO	#SST	ADDRESS (1)

NOTES: 1. PC is the updated Bus Control Element (BCE) program counter, i.e., the address of the next sequential instruction.

DESCRIPTION

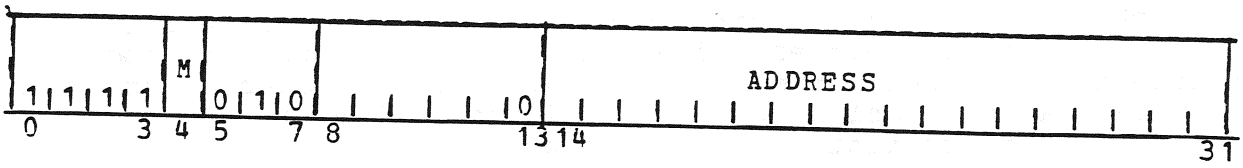
This instruction stores the BCE's status register in the fullword location addressed by the Effective Address. The least significant bit of EA (the halfword address) is ignored. After initiating the store operation, the program counter is incremented by 1 and the next sequential instruction is begun.

PROGRAMMING NOTE

In indexing mode the computation of the effective address includes the number of the BCE that is executing this instruction. This allows many BCE's to use the same BCE program, and yet store the status from each BCE in a different location. Note that the BCE number is multiplied by 2 to give a fullword index.

LOAD BASE REGISTER

#LBR



M	EFFECTIVE ADDRESS	INSTRUCTION FORMAT	
0	Address	#LBR	Address
1	(Address + 2 x BCE#)	#LBR@	Address

DESCRIPTION

The 18 bit effective address is loaded into the current Bus Control Elements (BCE) Base Register. The associated program counter is incremented by two.

PROGRAMMING NOTE

In direct mode the computation of the effective address includes the number of the BCE executing the instruction. This allows many BCE's to use the same program but, each BCE will get a different Base register. Note that twice the BCE number gives a fullword index.

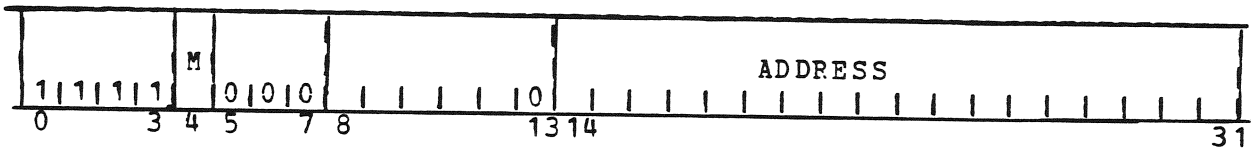
3.2 BCE BRANCHING

The BCE instruction set includes instructions directing it to reset its own Program Counter and execute instructions at locations other than the next sequential one. As with the register class, these instructions affect only the PC register belonging to the BCE executing this instruction, and affects no other BCE.

This class of instruction includes an unconditional branch instruction and an instruction that allows a Listen Mode BCE to translate a Listen Command into a new Program Counter setting via a table lookup.

BRANCH UNCONDITIONAL

#BU



M	Effective Address (EA)	INSTR. Format	
0	ADDRESS	#BU	ADDRESS
1	(ADDRESS + 2 x BCE#)	#BU@	ADDRESS

DESCRIPTION

The 18 bit effective address is stored into the current Bus Control Elements (BCE) Program Counter. The next instruction is found at this designated location, which may be on either a full or halfword boundary.

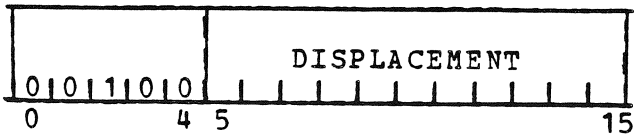
PROGRAMMING NOTE

In direct mode, the computation of the effective address includes the number of BCE executing the instruction. This allows many BCE's to use the same program and still retain the capability to branch to different segments as required for each BCE's operation.

Note that twice the BCE number gives a fullword index.

WAIT FOR INDEX
#WIX

FORMAT



INSTRUCTION FORMAT

#WIX Table

NOTE: Table = PC + Displacement, where PC is updated Program Counter, and Displacement is integer between -1024 and +1023.

DESCRIPTION

This instruction places the BCE in a state where it will monitor, through its MIA, the system bus to which it is attached for commands from other IOPs. When it receives such a command, it uses part of the command as an index into a table of branch addresses, and branches to the indicated location. This procedure allows one BCE in one IOP to signal to another BCE in a different IOP that it is time to perform some BCE program.

Figure 3.2 diagrams operation of this instruction. The starting address of the table of branch addresses (one address/fullword) is the sum of the updated PC (1 + address of present #WIX) and the 11 bit Displacement field. This address is rounded up by 1 if necessary to make it a fullword address (least significant bit = 0). After computing this address the BCE sets itself up to accept from its MIA a bus word termed a "Listen Command" that has command sync and an Interface Unit Address of 0 1 0 0 0 (in binary).

The BCE then goes into a tight loop of monitoring the MIA Buffer for a valid Listen Command. If at the entry to this loop, or at any time during this loop, the BCE finds that it is not in Listen Mode (i.e. its transmitter is enabled -- see Paragraph 4.1) then it will exit the loop and enter the Wait State. If it stays in Listen Mode, and finds a valid Listen Command, it exits the loop. The BCE then places bits 14 to 18 of the command in its Interface Unit Address Register (IUAR). It also adds the Index bits 19 to 26 to the Table address computed earlier. This 8 bit Index is right justified and padded on the left by ten zero's when it is added to the 18 bit Table address. Figure 3.1 diagrams the makeup of a Listen command.

This computed address is used to reference a fullword in memory which contains in bits 14 through 31 a branch address. These 18 bits are

then loaded into the BCE's PC, and execution of the indicated instruction begun.

PROGRAMMING NOTES

If a #WIX is executed with the MIA's transmitter enabled, execution of the #WIX is equivalent to that for a #WAT, i.e., it resets its Busy/Wait bit, updates its PC by 1, and goes into a loop until the MSC sets the bit back to 1. If the #WIX is executed with the MIA's transmitter disabled, the PC is not updated and remains at the #WIX instruction.

If the BCE is in Listen Mode, then during the entire time that the BCE is waiting for a Listen Command the BCE's Busy/Wait bit stays set to Busy. Thus any attempt by the MSC to execute a @LBB, @LBP, or @SIO involving this BCE will not go through, and will result in an MSC error and the setting of appropriate bits in the MSC Status Register.

After execution of a #WIX, the BCE's IUAR has been set to bits 14 to 18 of the received Listen Command. This permits the BCE/IOP that placed the Listen Command on the bus to condition the listening BCE(s) to accept data only from a certain subsystem. Typically the commanding BCE sends this subsystem a command to return a stream of data, which will then be picked up properly by not only the commanding BCE but also those on the same bus that were "listening" to the command BCE. Paragraph 4.1 should be referenced for more complete detail.

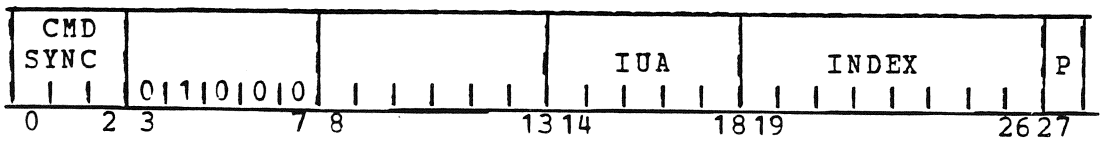


Figure 3.1. Bit Times of Listen Command

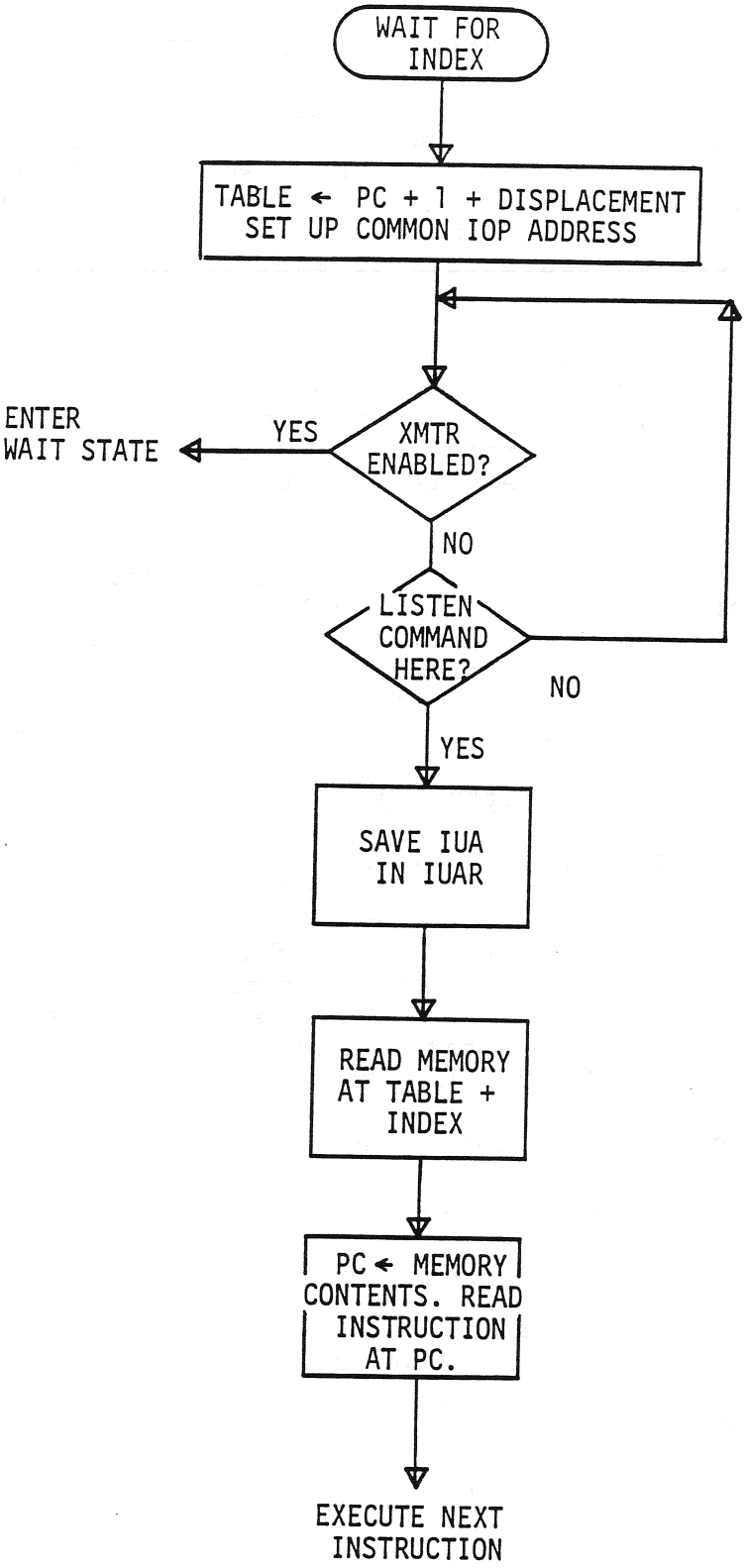


Figure 3.2. Wait for index

3.3 BCE TRANSMISSION INSTRUCTIONS

Each BCE in an IOP has the capability of directing its MIA to initiate the transmission of a word over the associated bus. These words may be either command or data words and when they are transmitted over the bus they have the formats shown in Figure 1.1(a) or (c). Out of these bits the BCE provides only the 24 information bits 3 through 26 and an indication of whether the word is a command or data word.

Command words are used to tell a subsystem to perform some action such as, get setup to accept N words of data that will be transmitted later. Since there can be many subsystems on a bus, each command contains a 5-bit Interface Unit Address (IUA), bits 3-7 of a bus command word, that specifies which subsystem should obey the command.

Data words are typically sent after a command, and contain the actual information that the BCE wants transferred to the subsystem. For each bus word this information consists of a 16-bit halfword from GPC memory. Surrounding this word are sync bits, a 5-bit IUA, the pattern 101, and parity. The BCE provides the MIA with just the IUA, the 16-bit data, and 101. The MIA adds the rest.

3.3.1 Transmit Command Instructions

The BCE instruction set contains instructions that direct the transmission of both commands and data. The Transmit Command instructions (#CMDI and #CMD) provides all 24 bits of command information needed for the transmission of a single command word. The Message Out Instruction provides both a command and the location of a stream of data to be transmitted. The execution of such instructions also sets the BCE's Interface Unit Address Register (IUAR) to whatever IUA is specified in the command. The Message In (#MIN) instruction combines both the transmission of a command and the reception of a stream of returning data. These instructions behave differently in Listen mode than in Command mode. The differences are described in the instruction description and in Section 4.3.

3.3.2 Transmit Data Instruction

The Transmit Data instructions (#TDS, #TDLI, #TDL, and #MOUT) specify the starting address of a buffer in memory and the number of halfwords that are to be taken sequentially from this buffer and given to the MIA for transmission as data. The subsystem to which this data is directed is specified in the IUAR, which typically was set during the transmission of the last command.

All buffer addresses for data transmission are Base register relative. The instructions contain only a displacement (sometimes assumed to be zero) that is added to the Base to compute the buffer starting address. This implementation is particularly useful when I/O

buffer areas are formatted in exactly the same way (for example for identical buffers containing torquing data for a set of identical gyros). The same BCE program can be used to output the data from any of these buffers by appropriate initialization of the BCE's Base register before the BCE is started at that program.

Base relative addressing also allows simultaneous use of the same BCE progam by several BCE's. Initializing each BCE's Base register differently allows each BCE to transmit different data, while using the same BCE program.

Each Transmit Data instruction specifies the number of memory halfwords to be transferred. In all cases, the actual binary field used to specify this transfer count is treated as a positive integer that is numerically 1 less than the number of halfwords that will be transferred. Thus, a count of 0 corresponds to a transfer of 1 halfwords; a count of 262,143 corresponds to a 262,144 transfer.

3.3.3 Typical Bus Timing - Gaps Between Outputs

A typical sequence of BCE instructions that transmit a set of data to a subsystem consists of either a Transmit Command instruction followed by a Transmit Data instruction or a single Message Out instruction. The resultant sequence of activity as seen on the bus is pictured in Figure 3.3. Each bus word requires 28 microseconds for transmission - during which time the MIA is considered busy - and is separated from the next bus word by a short period of time during which the bus is inactive. These periods of time are called "inter-word gaps". On Figure 3.3, the gap between the ith and i+1st data words is denoted as g(i), where the gap between the command and first data word is g(0).

Under normal operating conditions the gaps between data words ($g(i)$, $i \geq 1$) transmitted by a BCE are 5 microseconds. The length of gap g(0) depends on the instruction sequence used to transmit the command. For a Message Out instruction this gap is fixed at 5 microseconds. For a Transmit Command instruction followed by a simple transmit data instruction (non #MOUT), the gap g(0) depends on the exact type of Transmit Command and the Transmit Data, whether the first data word comes from an even or odd halfword, and how long it takes to make all memory references between execution of the Transmit Command and the Transmit Data. Assuming no delay due to memory contention (See Error Modes - Excessive Concurrency), the gap g(0) for such cases can be estimated from:

Time to execute all instructions between the Transmit Command instruction and the Transmit Data instruction

+

Time to set up the Transmit Data Instruction

- 12 usec

The time to set up any Transmit Data instruction is the time required to compute the address of the first word of data, to save the transfer count and to request and receive this first word. Table 3.1 summarizes some typical initial gaps and assumes no intervening instructions and no delays due to memory contention.

Ideally, the gaps between data words (g(i), i≥1) transmitted by a BCE is 5 usec. However, due to memory contention, this gap can be either 5 or 21.5 usec with the transmit instructions #TDS, #TDL, and #TDLI. With #MOUT, this gap is fixed at 5 usec. If the BCE cannot meet these constraints, the transmit instruction will fail, the BCE will go to the WAIT State, set its "NOGO" bit in the program exception register (STAT1), and set bit 21 in its status register.

3.3.4 Echo Back

Whenever a MIA transmits something it echos back into the MIA Buffer a copy of what is sent out. This copy stays in the MIA buffer until either it is overwritten by something else received or transmitted by the MIA or it is removed by the BCE during a Receive Data Instruction.

Note that if one or more data words are transmitted, the last data word remains in the MIA buffer. A subsequent read data instruction would mistake this word from the first data word and probably time out waiting for what it thought was the second word. At least the data would be skewed one word to the right and the last word would be lost.

This problem can be avoided by entering the halt state between the transmit and the receive instructions or by issuing a read of one word and discarding the data and then issuing the desired read command.

The above situation does not create a programming problem in most BCE operations, because in the usual case the word in the buffer when the data read is executed will be a command word, which the read microprogram will inspect and discard. One case in which the programmer must take into account the problem is that of mass memory write operations, where the IOP transmits 512 data words and then receives a Search Complete Word (SCW) which was automatically transmitted by the mass memory without a command word intervening. In this case the read of one word must be made to clear the buffer, then a read with delay to await the SCW. The problem has also been reported to occur in DEU dump responses. The same avoidance technique is effective here.

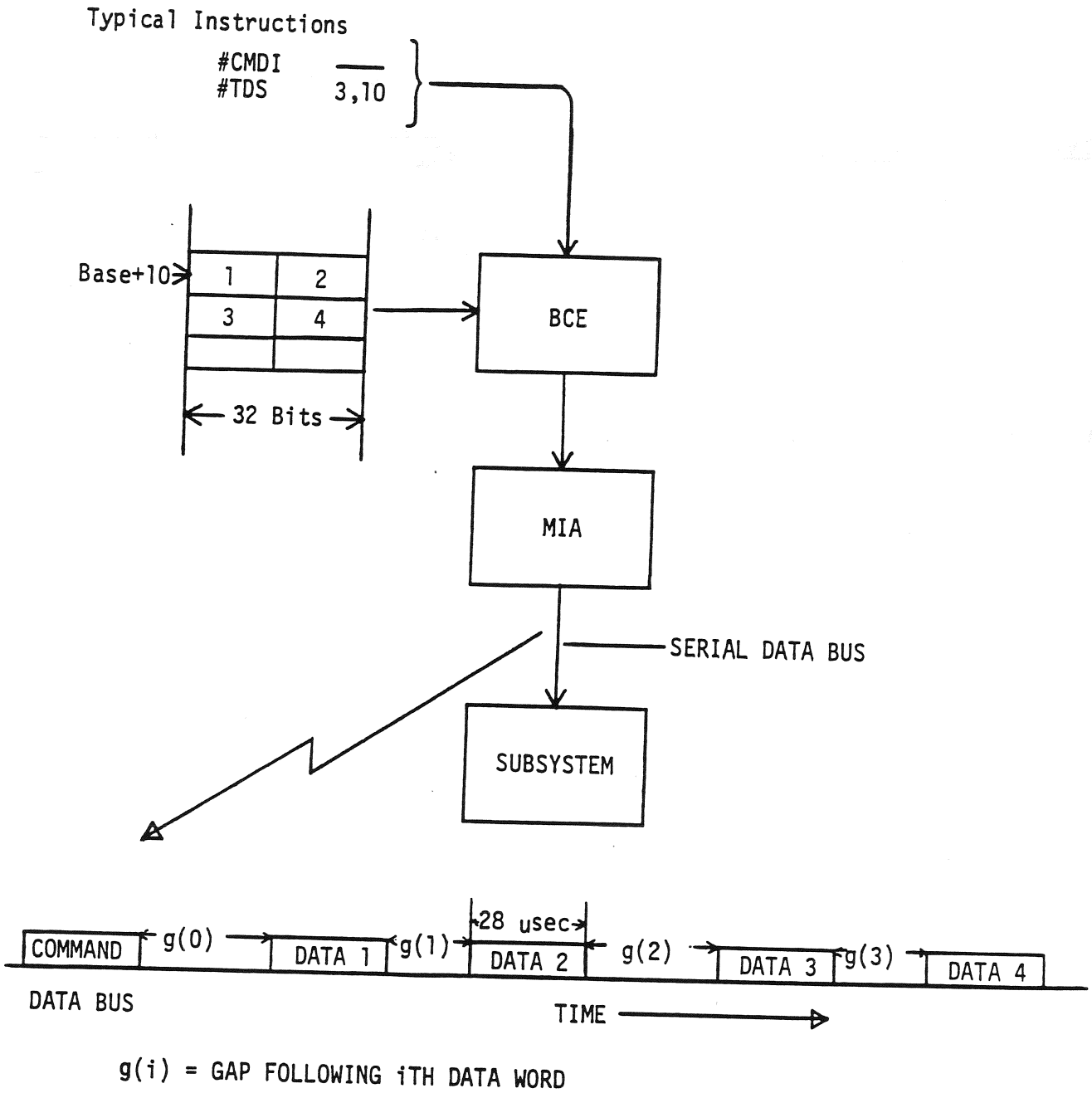


Figure 3.3. Typical Transmit Sequence

TABLE 3.1

TYPICAL GAPS BETWEEN COMMAND AND DATA WORD

Type of Transmit Data	Assuming 1st Data from Even Addressed Halfword	Assuming 1st Data from Odd Addressed Halfword
#TDS	21.5	38
#TDLI	21.5	38
#TDL	38	54.5
#MOUT	5	N.A.

All Times in Microseconds

3.3.5 Error Modes - Disabled MIA

There are two aspects of general IOP operation that are outside the control of an individual BCE but that affects the execution of a Transmit instruction. These are the condition of the MIA's transmitter and receiver (enabled or disabled), and the amount of traffic through the IOP DMA channel due to PC I/O and DMA requests from the MSC and other BCE's. The status of the MIA is controlled by the Transmitter Enable and Receiver Enable bits in the Control Monitor. These bits are set/reset strictly by the CPU via a PCO. The transmitter must be enabled before a BCE will attempt to transmit either commands or data. If the transmitter is disabled at any time during the execution of a Transmit Data instruction (an error mode), the BCE executing that instruction will terminate the transmission of the rest of the data stream, set its Program Exception bit to 0 and bit 23 of its Status register and its BCE-MSC Indicator bit to 1, and enter the Wait State. It is up to the MSC or CPU, by analysis of the Program Exception register and Status Registers, to detect this error and perform appropriate recovery.

A related error mode occurs if the MIA is still busy when the BCE has decided it is time to transmit another word of data. The normal timing of a BCE makes this impossible unless either the MIA or BCE has failed. Therefore, the termination is the same as for a disabled transmitter.

Execution of a Transmit Command instruction, with the transmitter disabled, will not cause an error condition. This allows the Listen Mode BCE's to execute the same programs segments that the BCE/IOP in command executes. For example, a BCE must tell a subsystem to return some data but a listen BCE whose transmitters are disabled should ignore such commands and simply continue to the following receive instructions.

Note that this is particularly true of the Message In instruction, where a Command mode BCE will transmit the command and a Listen Mode BCE will go directly to the receive loop.

3.3.6 Error Modes - Excessive Concurrency

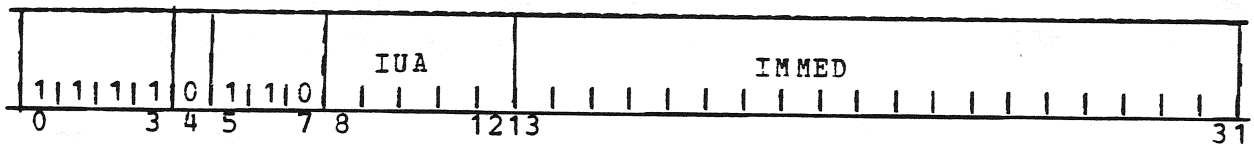
The IOP is functionally a multiprocessor with at least 25 separate processors (24 BCE's, 1 MSC), all of which can be generating memory requests independently. Since there is only one DMA channel and it can handle only one request at a time, these requests may stack up, delaying the honoring of the later requests for significant periods of time. If these requests were for data for a Transmit Data instruction (#TDS, #TDLI, or TDL), and they were delayed beyond the time at which the BCE wished to transmit them, then there could be significant periods of dead time on the bus between separate transmission of bus data words. These dead times are called "inter-word gaps", and if they become excessive they will cause the subsystem at the receiving end to time out and declare the BCE to be at fault.

Consequently, a BCE in the middle of a Transmit Data loop will allow a gap of at most 21.5 microseconds to occur between data words. If a data request is not honored by the time a gap of 21.5 microseconds has occurred since completion of the last data word transmission, then the BCE executing the instruction terminates the transmission, sets its Program Exception bit, its BCE-MSC Indicator, and bit 21 of its Status register, and enters the Wait State. An MSC or CPU routine must then handle recovery or retry as deemed appropriate.

A Message Out instruction provides more time for data read requests to be handled and thus is less susceptible to excessive concurrency than any of the simple Transmit Data instructions. Consequently, a #MOUT instruction will error terminate in the above fashion if it cannot maintain a 5 usec gap between all outputs in the message including those between the command word and the first data word.

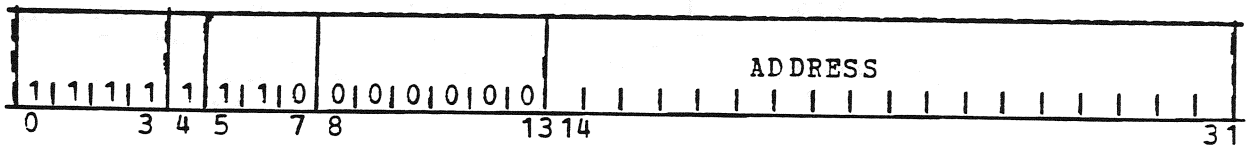
If the DMA Channel discovers a parity error when it is honoring a memory request for a Transmit Data instruction, it interrupts the CPU and terminates the request. This is reflected in the BCE by the failure of the data to arrive on time, and consequently by eventual execution of the time-out sequence described above.

Note that there is no time out check on the first data request for #TDS, #TDLI, or #TDL, or command fetch on #MOUT, #MOUT@, #MIN, or #MIN@. This can result in the instruction cycling indefinitely.



INSTRUCTION FORMAT (BIT 4 = 0)[1,2]

#CMDI IUA, IMMED



INSTRUCTION FORMAT (BIT 4 = 1)

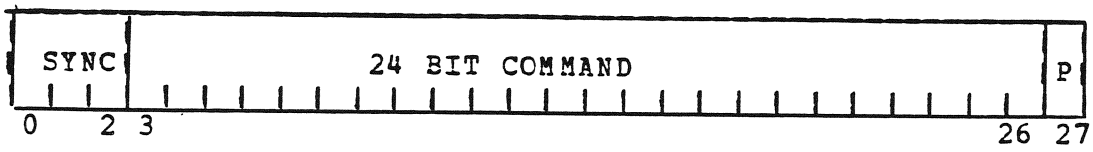
#CMD ADDRESS[3]

NOTES:

1. IUA is the Interface Unit Address, a 5 bit quantity in the range of 0 to 31.
2. Immed is a 19 bit constant.
3. Address + 2 x BCE# is the main storage location of a fullword containing a 24 bit command, right justified.

DESCRIPTION

This instruction is used to send 24 bit commands to a subsystem on the serial bus connected to the current BCE's MIA. In immediate mode (#CMDI) the command is found immediately in bits 8 thru 31 of the instruction. In direct mode (#CMD) the command is the lower 24 bits of the main store fullword computed from the address field (Bits 14 thru 31) of the instruction. (The halfword addressing bit, bit 31, is ignored). The format of the commands sent out by the MIA is:



SYNC = COMMAND SYNC

P = PARITY

The actual transmission of the command depends on the BCE's associated MIA's Transmitter being enabled and the MIA being not busy. When the command is transmitted, the BCE IUAR is loaded with the contents of the command's IUA field (command bits 3-7). There is a slight difference in the way the two OP codes handle the case where either one of the required conditions are not met:

#CMD : if MIA busy is true or transmitter enabled is false, no action is taken, no error condition is set.

#CMDI: if MIA busy is true or transmitter enabled is false, this OP code cycles and repeats the check one time. If either condition exists at that time, the command is also not transmitted. On this OP code, the IUA register is loaded with the command IUA. No error condition is set.

After execution of this instruction the BCE's program counter is incremented by 2, and the next sequential instruction is executed.

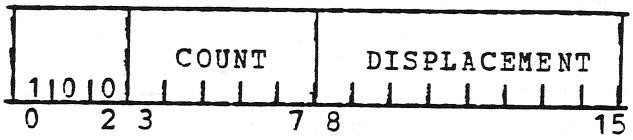
PROGRAMMING NOTES:

The start of the actual transmission of the command by the MIA begins about 16 usec before the end of the instruction. Thus execution of the next instruction following a #CMDI or #CMD begins before the MIA has completed transmission of the command. If this next instruction is a #CMDI, it will fail the MIA busy check and try again, as described. This condition does not affect the 2nd check for #CMDI or the check on #CMD.

For the #CMD instruction, the address of the command word includes the number of the BCE executing the instruction. This allows many BCE's to execute the same BCE program, but at the same time it allows each BCE to transmit a different command. Note that twice the BCE number gives a fullword index.

TRANSMIT DATA SHORT

#TDS



INSTRUCTION FORMAT

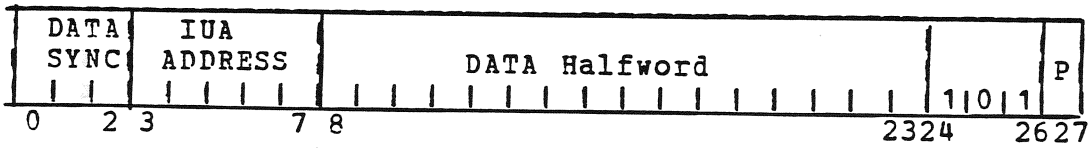
#TDS COUNT, DISPLACEMENT

NOTES: Count has range 0 to 31. (1 less than the number of transfers)

Disp. has range of 0 to 255.

DESCRIPTION

This instruction directs the Bus Control Element (BCE) to transmit a number of 16 bit memory halfwords (determined by the COUNT) through the BCE's MIA to a subsystem attached to the MIA's serial bus. The location of the first halfword is the sum of the BCE Base Register and the Displacement field. This may be any halfword location. Succeeding output quantities come from succeeding halfwords in memory. Each halfword is assembled by the BCE into the following format and given to the MIA for transmission. (The MIA adds the sync and parity).



P = PARITY

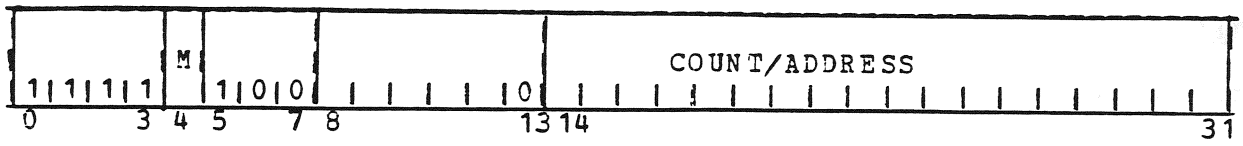
The Interface Unit address (IUA) is a copy of the current contents of the Interface Unit Address Register (IUAR) which in turn was loaded during the execution of the last #CMD instruction performed by the BCE executing this #TDS. The value in the IUAR is thus the last subsystem to which this BCE has sent or attempted to send a command.

The number of bus words actually sent is 1 more than the number in the Count Field. Thus a Count of 0 causes one memory halfword to be transmitted; a count of 31 corresponds to 32 half words.

Completion of data transmission is followed by incrementation of the BCE's program counter by one, and execution of the following instruction.

If at any time during execution of this instruction, the BCE is not able to keep the time gap between intermediate data words on the bus to less than 21.5 microseconds, then bit 21 of the status register is set, the BCE's Program Exception bit (STAT1) is set to 0, the BCE-MSC Indicator is set to 1, the present instruction terminated, and the Wait State entered. The cause of such gaps may be due to memory contention that prevented a data read from being completed in time, or a parity error when the DMA channel attempted to read the data from memory for the BCE.

The instruction will also be error terminated (with bit 23 in the Status Register set to 1) if the MIA transmitter is either disabled or found busy when it was time to transmit a data word. Either situation represents an error condition.



M	EFFECTIVE TRANSFER COUNT	INSTRUCTION FORMAT	
0	Bits 14 thru 31 of instruction (Count Field)	#TDLI	Count
1	Bits 14 thru 31 of fullword addressed by address field plus 2 x BCE#.	#TDL	Address

DESCRIPTION

Execution of this instruction is the same as that for Transmit Data Short, with the following exceptions:

1. The displacement from the base is zero. The base must then point to the beginning of the buffer.

2. The count of input words to be transmitted can range from 0 to 262143, and may be specified by either bits 14 thru 31 of the instruction (#TDLI) or by bits 14 thru 31 of the main storage fullword addressed by bits 14 thru 31 of the instruction (#TDL). In the second case the least significant bit of the address (the halfword selection) is ignored. In either case the number of memory halfwords actually transmitted is 1 more than the Effective Transfer Count.

3. After completion of the instruction, the BCE's program counter is incremented by 2.

PROGRAMMING NOTE

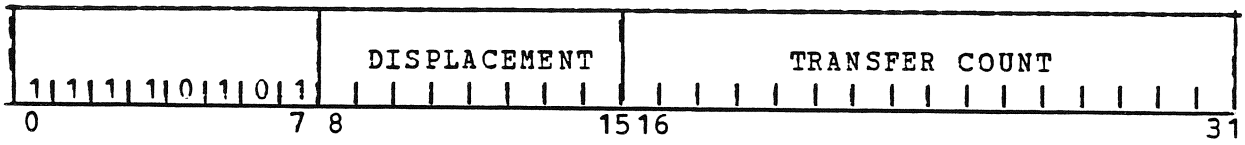
For the #TDL instruction the address of the transfer count includes the number of the BCE executing this instruction. This allows many BCE's to execute the same program while at the same time allowing each BCE to transmit a different number of outputs.

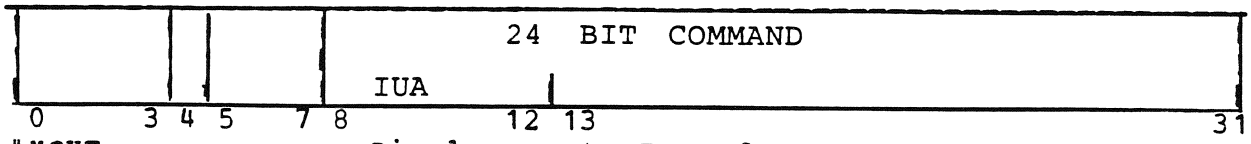
When a BCE is transmitting to BCEs in one or more IOPs, the number of words transmitted may be limited by slight variations between GPC oscillator frequencies. In order to preclude the possibility of data loss for this reason and yet to satisfy system requirements, an upper limit of 2048 words is recommended for GPC to GPC transfers.

MESSAGE OUT #MOUT

INSTRUCTION FORMAT (Bit 4 = 0)

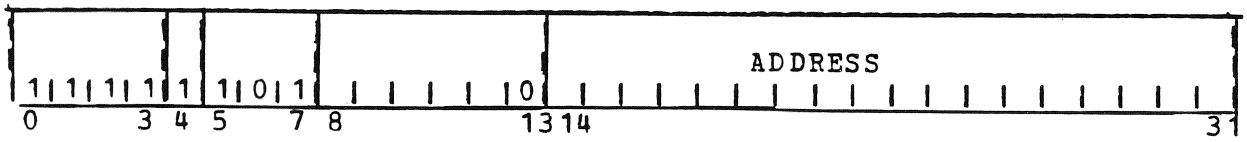
FORMAT (2 Words)





#MOUT Displacement, Transfer Count
#MOUTC IAU, Command

FORMAT (1 Word)



INSTRUCTION FORMAT (Bit 4 = 1)
#MOUT@ Address

NOTE: Displacement, Transfer Count found at Address +2 x BCE #; Command found at Address + 48 + 2 x BCE #

DESCRIPTION

This instruction initiates the transmission of a command followed immediately by a stream of data. As such it performs in one instruction approximately the same functions as a Transmit Command (#CMDI or #CMD) followed by a Transmit Data (#TDS, #TDLI or #TDL). The command followed by the data is termed a "message".

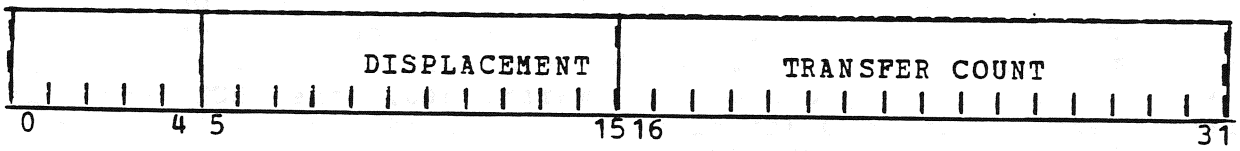
Message Out comes in two formats, either of which allows independent specification of the command to be sent, the number of data words to be sent, and the location of the data buffer in main memory. The first format consists of a 2 word instruction. The second word contains all 24 bits of the command to be sent. The first word contains an 8 bit Base relative displacement and a 16 bit Transfer Count. The Displacement may be any positive integer between 0 and 255, and is added to the present contents of the Base register to form the address of the data buffer. Unlike #TDS, #TDLI, and #TDL

this address is assumed a fullword address -- the least significant bit of the address is ignored. Consequently, the first data word to be sent out will come from the upper half (bits 0 to 15) of the selected fullword.

In this format the Transfer Count may be any positive integer between 0 and 65535, and represents one less than the actual number of data halfwords to be taken from memory and transmitted as data words on the BCE's bus.

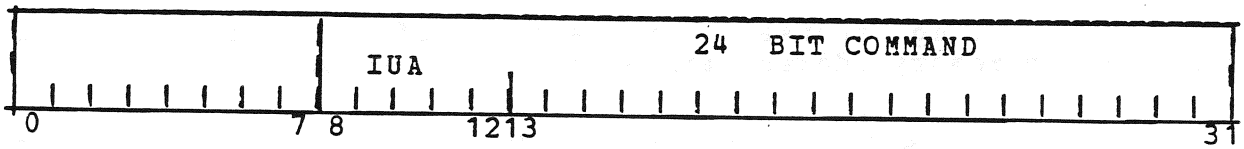
The second format allows automatic indexing by BCE number into two tables providing basically the same information found in the first format. Each table has 24 entries, one per BCE. Twice the number of the BCE executing this instruction is used as an index so that the table entries are each fullwords.

The first table starts at ADDRESS +2, and contains the Displacement and Transfer Count. For BCE i the table entry at ADDRESS + 2 x i is formatted as:



This allows a range of displacement between 0 and 2047 and a transfer count between 0 and 65535. As before, however, the sum of the Base and Displacement is treated as a fullword address.

The second table starts at ADDRESS + 50, and contains the 24 bit command. For BCE i the table entry at ADDRESS + 48 + 2 x i is formatted as:



Operation of #MOUT is diagrammed in Figure 3.3. The Transfer Count and Displacement are derived from the instruction, and are followed by requests to read the memory words containing the commands and the first fullword of data. After the command word has been read from memory, the 5 bit IUA in the command (bits 8 to 12 of the memory word containing the command) is saved in the BCE's IUAR, and the command given to the MIA for transmission.

After command transmission, the BCE goes into a loop that transfers data words to the MIA for transmission at a rate of 1 per 33 usec. These data words consist of the 5 bit IUA found in the IUAR and

a half-word of data from memory assembled as in Figure 1.1(a). When appropriate, the BCE also requests that new fullwords be read out of memory into the BCE. These words contain data that is to be transmitted later in the sequence. When the specified number of data words has been given to the MIA, the BCE exits this loop, updates the PC, and begins the next instruction.

As with any of the Transmit Data instructions, if the BCE ever attempts to transmit a word but finds either its transmitter disabled or its MIA busy at a time when it should be idle, the BCE will terminate the #MOUT, set its Program Exception Bit to 0 (an error has occurred), set its Status Register Bit 23 to 1, set its BCE-MSC indicator, and enter the Wait State.

Also as in a Transmit Data instruction, a BCE will terminate a #MOUT if it cannot maintain a controlled interword gap between output words. However, the nature of #MOUT allows memory requests, and data requests in particular, to be made much earlier than they could in a Transmit Command/Transmit Data instruction sequence. This reduces the effects of memory contention from other BCEs to the point where only very exceptional conditions will interfere with a #MOUT transmission. Consequently, a BCE will terminate a MOUT only when it cannot maintain an even 5 usec. gap between output words. This includes maintaining a 5 usec. gap between the command word and first data word. When this gap cannot be maintained, the BCE sets its Program Exception Bit to 0 (an error has occurred), sets bit 21 of the Status Register sets its BCE-MSC indicator, and enters the Wait State.

As with any of the Transmit Data instructions, the BCE will terminate in the same fashion if there is a problem in the DMA interface when one of its data read requests is being handled, such as a parity error or DMA hangup. The data from memory will never reach the BCE, and when the BCE reaches the point where that data is to be transmitted, it will detect the absence of the data and terminate.

PROGRAMMING NOTE

The starting address for the data buffer can be no greater than the maximum memory address minus three full words.

When a BCE is transmitting to BCEs in one or more IOPs, the number of words transmitted may be limited by slight variations between GPC oscillator frequencies. In order to preclude the possibility of data loss for this reason and yet satisfy system requirements, an upper limit of 2048 words is recommended for GPC to GPC transfers.

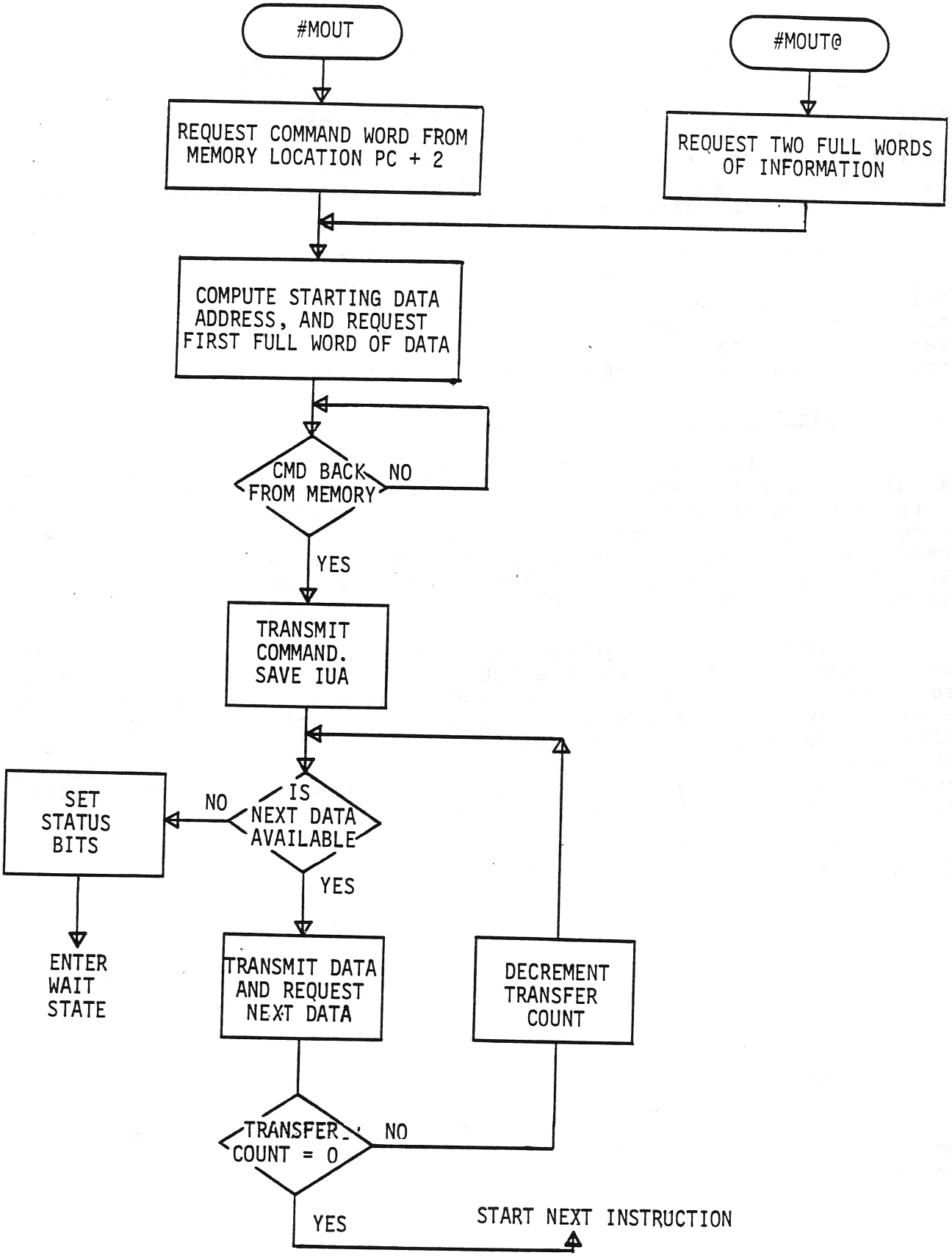


Figure 3.4. MOUT Operation

3.4 BCE RECEIVE DATA INSTRUCTIONS

Each BCE in an IOP has the capability of accepting, from its MIA, data that has been placed on the bus by either a subsystem or another GPC. When the data is on the bus, the data words are in the format shown in Figure 1.1(b). After BCE processing, only the 16 bits of data are saved and placed in memory. The BCE Receive Data Instructions (#RDS, #RDLI, #RDL, and the Message In instruction #MIN) tell the BCE how many such words to receive and the location in memory where the resulting data should go.

The following paragraphs describe each phase of the execution of a Receive Data instruction. This includes operation of the interface between the BCE and its serial bus, setup of a BCE Receive Data instruction, reception of the first word of a data stream, reception of intermediate words, and error handling. Figures 3.5 through 3.8 diagram the general outline of these procedures.

3.4.1 MIA-MIA Buffer-BCE Operation

A BCE obtains data from its system bus through its MIA and MIA Buffer. The MIA (Multiplex Interface Adapter) performs the serial to parallel conversions needed for the conversions between the serial bus format and the internal IOP parallel format. The MIA Buffer serves as an intermediate storage register between the MIA and the BCE. It is loaded by the MIA after the MIA has processed an input and unloaded by the BCE when the BCE is ready to accept data.

Figure 3.5 diagrams the time relationship between the bus/MIA, the MIA Buffer, and the BCE. Data words on the bus occupy 28 usec. During this time, the MIA (independent of the BCE) recognizes the sync pattern, accepts the bits one at a time, and accumulates parity on the incoming word. The 28th bit of the input is compared with this accumulated parity for fault detection. The MIA is considered busy in the interval from the detection of the sync pattern until the time for this 28th bit has elapsed.

Upon reception of an entire word, the MIA transfers the following data to its entry in the MIA Buffer:

1. The 24 bits of information from the word,
2. An indication of the kind of sync the word had (command or data) and,
3. Whether or not parity matched.

This is the last contact the MIA has with the data.

It should be noted that the MIA requires from between 1 to 5 usec to transfer a word to the MIA buffer after the receipt of the last bit.

The BCE operates asynchronously to the MIA, and samples the MIA Buffer to ascertain when a new data input has arrived. It samples the MIA Buffer only during either execution of a Receive Data or Wait for Index instruction. This latter operation is described in Section 4.

In either case the sampling process occurs at most once every 16.5 usec. When an entry is found in the MIA buffer, the BCE removes it and performs whatever series of error checks are deemed necessary.

Note that once an entry is placed in the MIA buffer it stays there until either the BCE removes it or the MIA overwrites it with a new value.

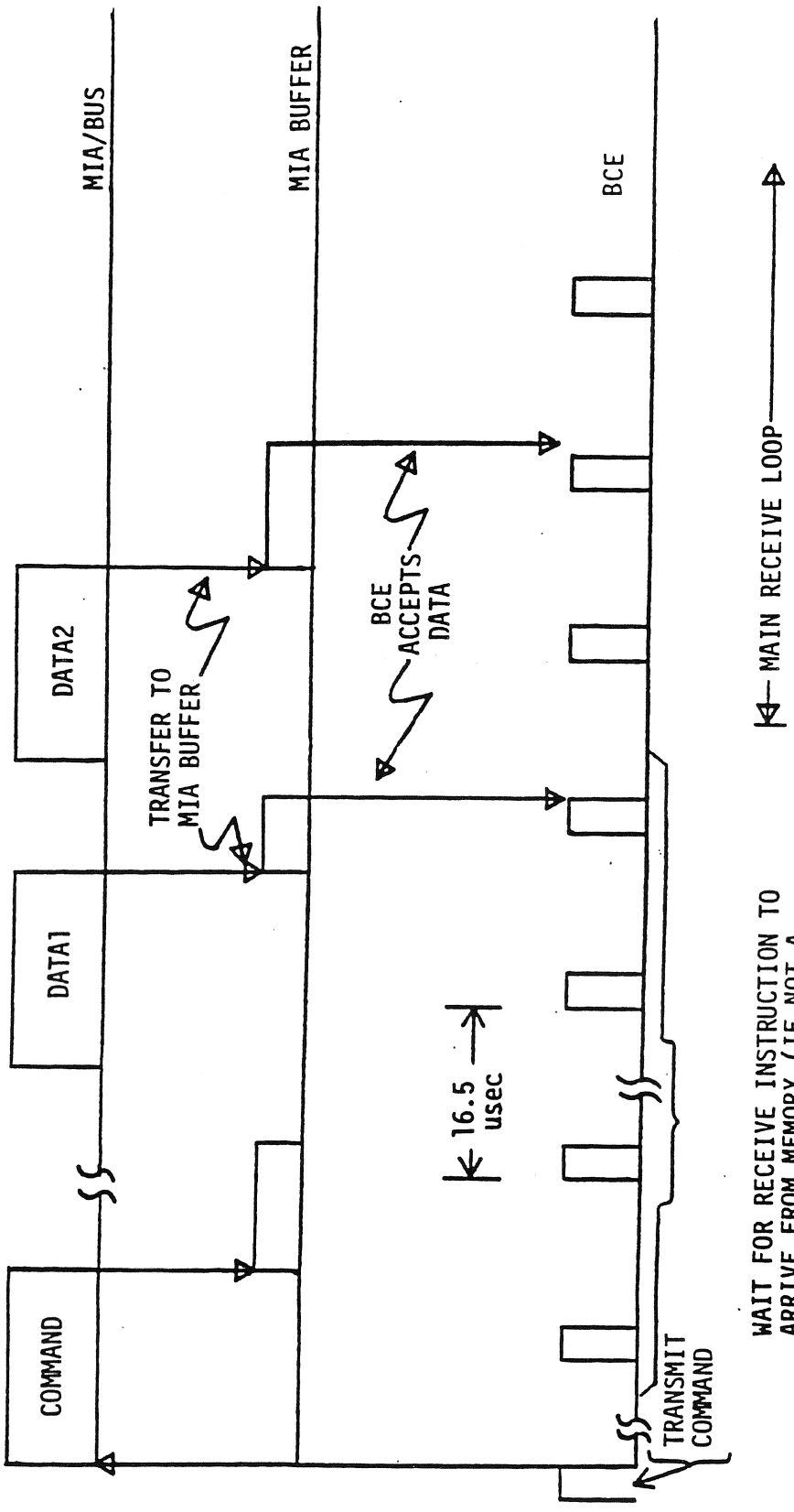


Figure 3.5. MIA-MIA-Buffer-BCE Operation

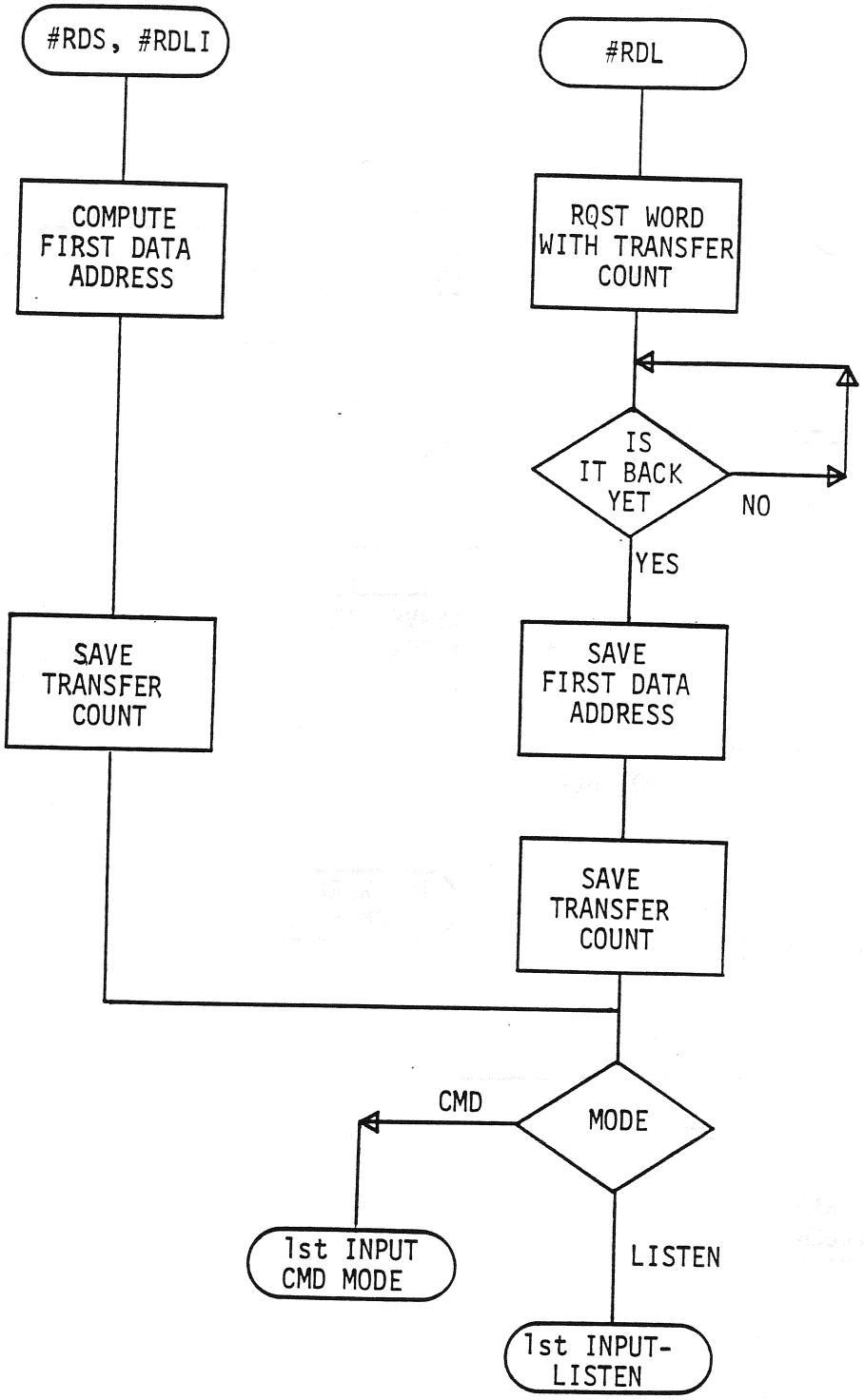


Figure 3.6. #RDS, #RDLI, #RDL Setup

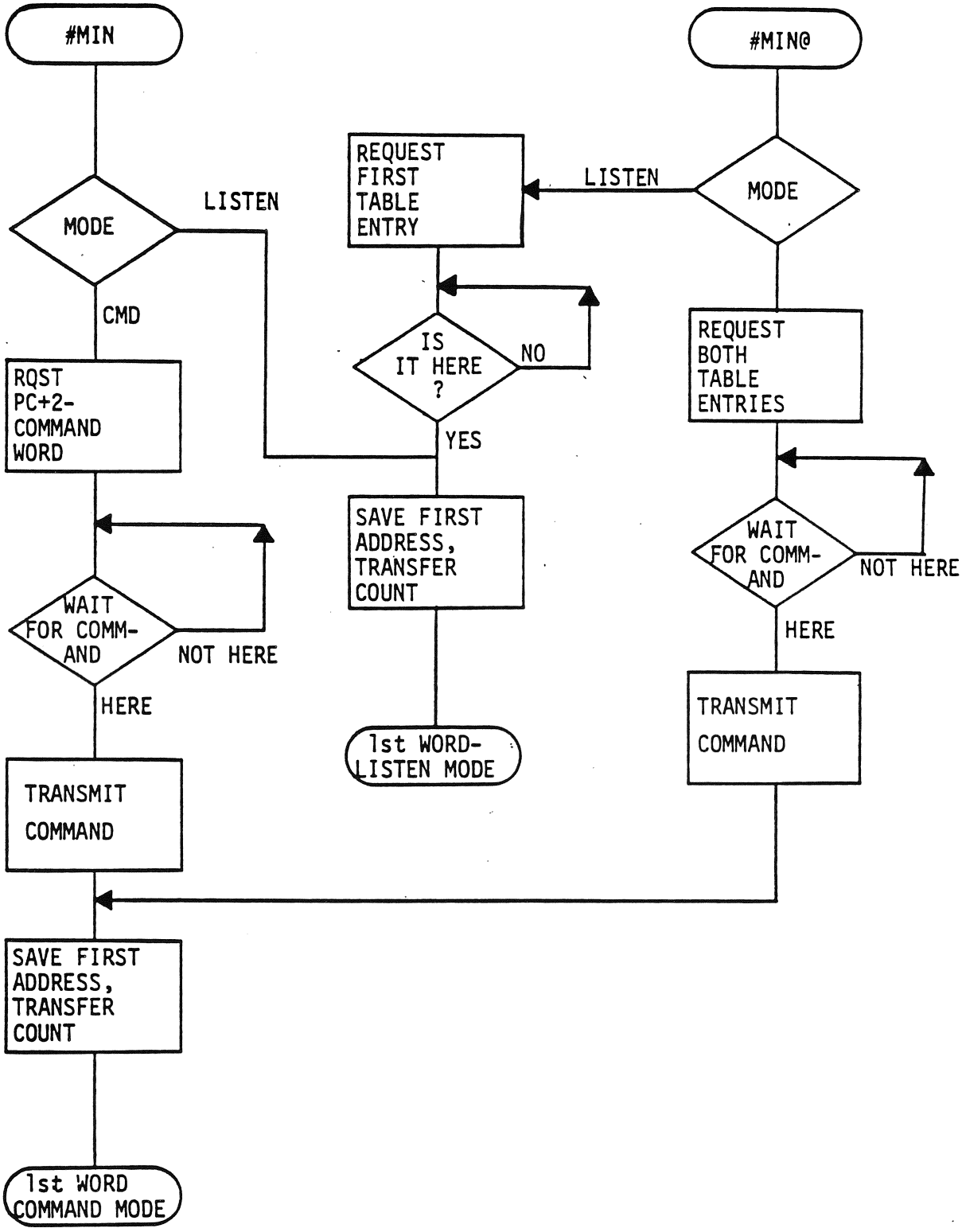


Figure 3.7. #MIN Setup

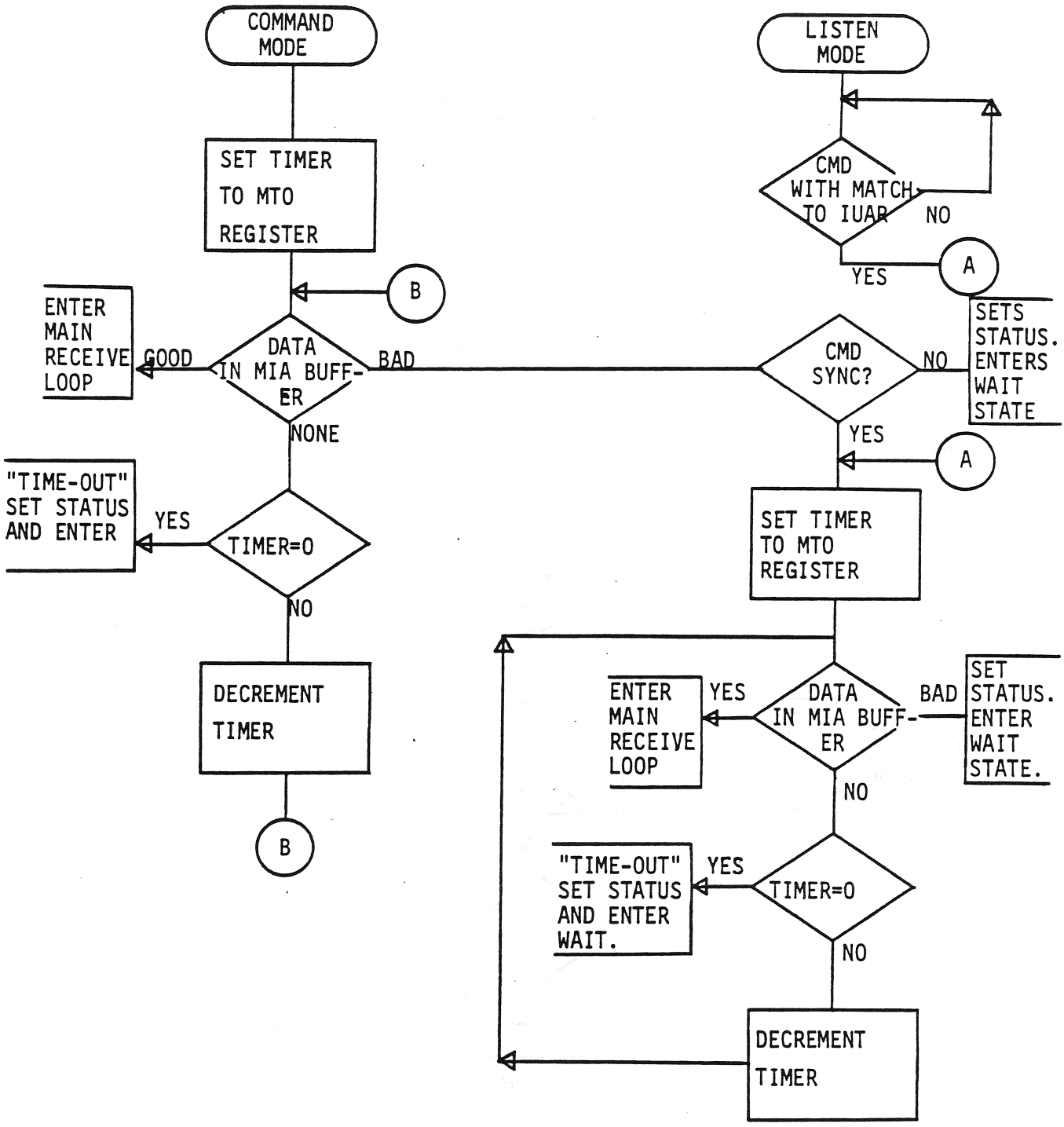


Figure 3.8. Receive Data Algorithm-First Input

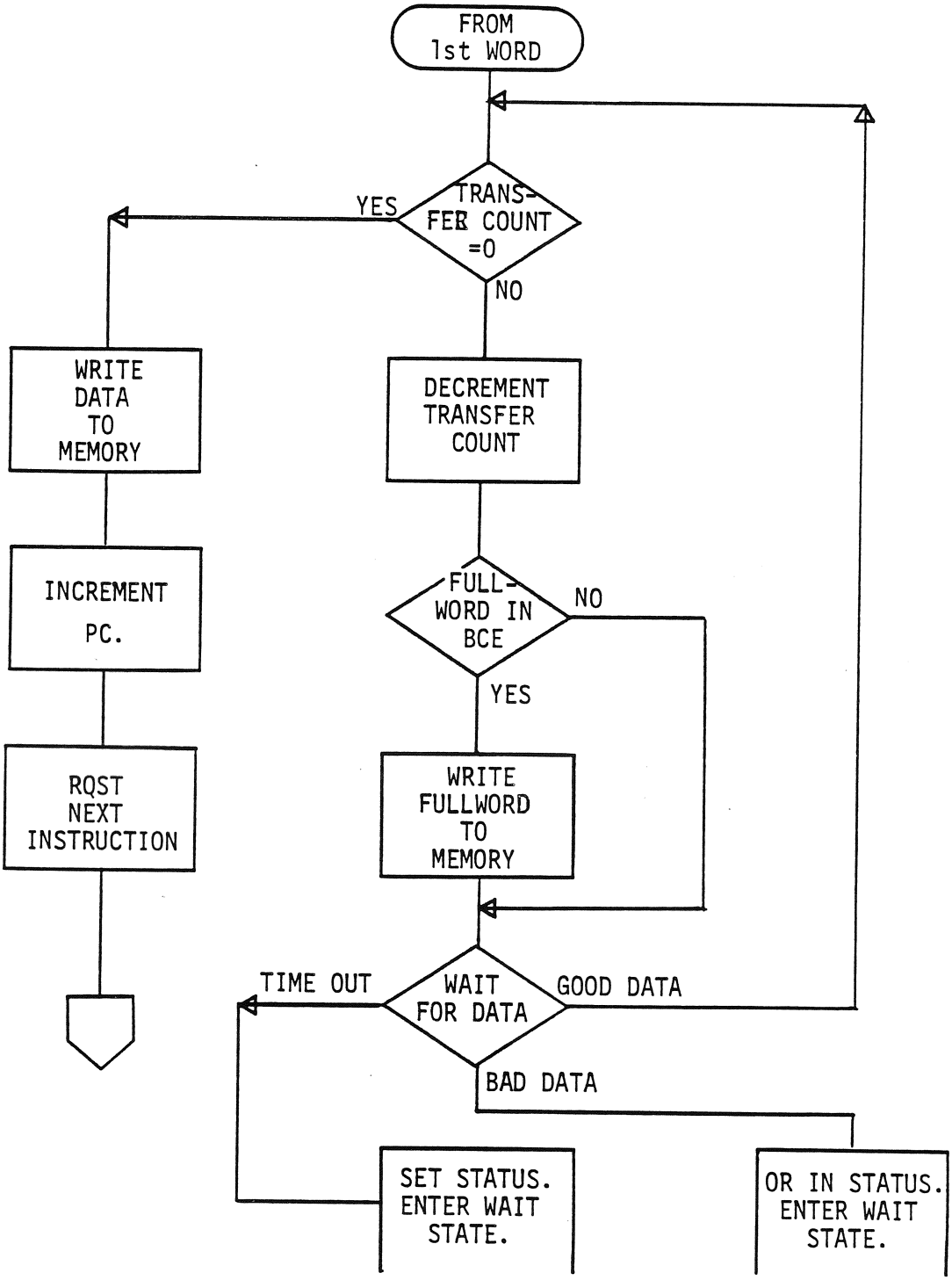


Figure 3.9. Main Receive Loop

3.4.2 Receive Data Setup

All Receive Data instructions specify:

1. The starting address of a buffer in memory in which the incoming data should be placed

2. The number of input words to be received.

Additionally, the Message In instruction specifies the command that should be sent to the subsystem before data should be expected in return.

The subsystem from which this BCE expects to receive data is specified in the IUAR.

This register could have been set in a variety of ways; including:

1) A CPU PCO while the BCE was in Wait State

2) For a command mode BCE, the execution of an instruction that included transmission of a command word (#CMDI, #CMD, #MIN or #MOUT).

3) For a Listen Mode BCE, reception of a Listen command while the BCE was executing a Wait for Index instruction or the execution of a #CMDI instruction.

All buffer addresses for data reception are Base register relative. The instructions contain only a displacement (sometimes assumed to be zero) that is added to the Base to compute the buffer starting address. This implementation is particularly useful when I/O buffer areas are formatted in exactly the same way (for example for identical buffers to contain data from a set of identical gyros). The same BCE program can be used to input the data to any one of these buffers by appropriate initialization of the BCE's Base register before the BCE is started at that program. Both the BCE itself (via an #LBR), the MSC (via an @LBB) and the CPU (via an PCO) can set any BCE's Base.

Base relative addressing also allows simultaneous use of the same BCE program by several BCE's. Initializing each BCE's Base register differently allows each BCE to receive different data into different buffers while using the same BCE program.

3.4.3 Acceptance of First Input

Once past setup, a BCE executing a Receive Data instruction will begin watching its MIA Buffer for bus inputs from the MIA. A BCE in Command mode will immediately start monitoring the MIA Buffer for inputs with data sync. A BCE in Listen Mode, however, will first await an input with command sync and an IUA that matches the BCE's IUAR. As demonstrated in Section 4, this procedure synchronizes the BCE/IOP that is commanding a subsystem to return data with a Listen Mode BCE (in another IOP) that is waiting for the data to return. Once a Listen BCE receives such a command it will begin waiting for the first data word.

A BCE in either mode will not wait indefinitely for the first input with data sync to arrive. Instead, once it starts waiting for a data word, it initializes a timer to the value contained in the BCE's Maximum Time Out (MTO) register. This timer is then decremented every 16.5 usec until either the timer reaches 0, or the BCE finds something in the MIA buffer. If the timer reaches zero first, the BCE terminates the reception and enters the Wait state as described in paragraph 3.4.7 (with bit 25 in the Status Register set to 1).

Thus, if the MTO register has a value of N in it, the BCE will look at most N + 1 times into the MIA Buffer before declaring a timeout.

Table 3.2 lists each Receive Data instruction and the time from the start of the instruction until the point where the BCE first looks at the MIA Buffer and initializes the timer to MTO.

TABLE 3.2

TIME TO FIRST LOOK AT DATA

Instruction	Mode	Minimum Time from Start of Instruction to First Look at MIA Buffer	
#RDS, #RDLI	Command	33 usec	
#RDL	Command	49.5 usec	3
#MIN	Command	82.5 usec	1,3
#MIN @	Command	99 usec	1,3
#RDS, #RDLI	Listen	49.5 usec	2
#RDL	Listen	66 usec	2,3
#MIN	Listen	66 usec	2
#MIN @	Listen	66 usec	2,3

1. This is 65.5 usec after the BCE has handed the command to the MIA for transmission.

2. This is 16.5 usec after the BCE has found a command in the MIA Buffer.

3. These times may be increased if necessary read requests are not handled within about 16 usec after they are made.

3.4.4 Error Checks

Upon receipt of an input from the MIA Buffer, the BCE performs a series of error checks, including:

1. Does the input word's IUA (bits 3-7 of Figure 1.1) match that contained in the BCE's IUAR (bits 13-17 of the BCE's Local Store cell C5.

2. Did the MIA detect a parity error.

3. Are any of the input word's SEV bits other than 101.

4. Was the sync data and not command?

If all these error checks are satisfied, then the data is accepted. If any one condition is not satisfied, the input is not accepted as a valid data word.

If the input is not the first input, the BCE error terminates. If it is the first data input, the resultant processing is a function of the BCE's mode. A BCE in Listen Mode will treat any discrepancy as an error and the BCE will terminate in the manner described in Paragraph 3.4.6. On the other hand, a BCE in command mode will check to see if the first input had command sync, and if so, it will ignore the input, reset the timer to MTO, and await the first input. This procedure allows a BCE that has just transmitted a command to skip over the copy of that command that has been echoed back by the MIA into the MIA Buffer. Note, however, that a second input with command sync will be treated as an error.

If the problem with the input is other than command sync, the BCE will error terminate regardless of its mode.

3.4.5 Handling of Good Inputs

When a BCE has found an input in the MIA buffer that satifies all of the above conditions, it extracts the 16 bits of data for storage in memory. If the data is destined for an even halfword location, and if further data is expected, this halfword is saved until the next input arrives. When it does, the two halfwords are combined and written as a fullword to memory.

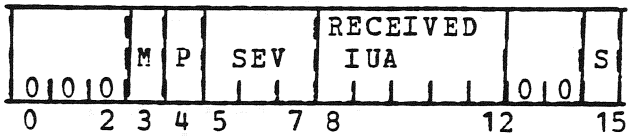
If the data is the first to be received, and it is destined for an odd halfword location, or it is the last input to be received, and is destined for an even halfword, the BCE will do a halfword write to store away the single 16 bits of data.

A BCE is ready to accept a new input 33 usec after recognizing the previous one. This time includes the above processing.

3.4.6 Error Termination - Faulty Input

A BCE that has detected a faulty data input will immediately terminate the reception and will enter the Wait state. Before it enters the Wait State, it also performs the following tasks:

1. Into the high half of the BCE Status Register (bits 0 to 15) the BCE or's in the following pattern.



WHERE:

M = 1 if there was a mismatch between the received interface unit address and the IUAR.

P = 1 if there was a detected parity error

SEV = the SEV bits from the input with the S and V bits inverted. Thus, a valid SEV pattern -101- forms 000 in the above pattern.

IUA = the actual Interface Unit address from the input.

S = 1 if the input had a command sync.

2. The BCE's Program Exception bit is set to 0 (error)

3. The BCE's BCE-MSC Indicator bit is set to 1.

4. BCE Local Store register A1 is left with a value equal to the number of inputs not yet received (i.e., the number of inputs not written to memory).

5. BCE Local Store register A0 is left pointing to the last halfword to receive good data. If the error is on the first input, A0 points to the beginning of the buffer.

6. The BCE's Busy/Wait bit is reset to Wait.

3.4.7 Error Termination - Time Out

A BCE that has waited the prescribed period of time for a data input will also error terminate to the Wait State in exactly the same fashion that is used on faulty inputs except that the appropriate BCE Status Register bit (either 25, 26, or 27) is set to 1.

3.4.8 Reception of Intermediate Words

After successful reception of the first word of a sequence, a BCE will continue to look into its MIA Buffer for additional data inputs until it has accepted the number indicated in the original Receive Data instruction. As with the first input, the BCE will not wait indefinitely. Unlike the first word, however, the time out detection procedure is normally fixed and is not adjustable by the programmer. This time is typically one additional IOP cycle, 16.5 usec. If the BCE does not find data in the specified time, the BCE error terminates with Status Register bit 26 set to 1.

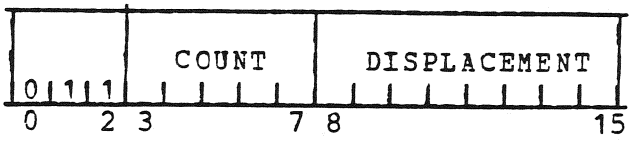
The only exception to the above error termination is if a timeout occurs and the number of inputs left to be received is an integer multiple of 512. If it is, the BCE will set a timer to MTO (as in the acceptance of the first input) and continue waiting. This procedure allows a BCE to recognize the inter-block gaps that occur between every 512 words of an input stream from the Mass Memory or Display Units. A timeout after this point has has been reached will cause the BCE to error terminate with Status Register bit 27 set to 1.

Upon recognition of a data input before time runs out, a BCE will perform the same error checks applied to the first input. Any discrepancy regardless of the BCE's mode, will cause the BCE to error terminate.

Data that meets all the above conditions is saved by the BCE for transmission to memory, as described in paragraph 3.4.5.

RECEIVE DATA SHORT

#RDS



INSTRUCTION FORMAT

#RDS COUNT, DISP.

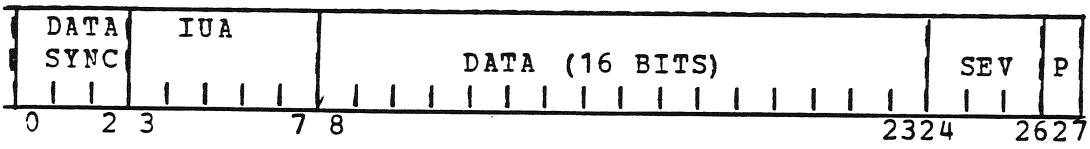
NOTES: Count has range of 0 to 31. (1 less than number of transfers)

Displacement has range of 0 to 255.

DESCRIPTION

This instruction directs the Bus Control Element (BCE) to accept a number of input words from its associated MIA, do a variety of checks on these words, and assemble them into 32 bit memory fullwords and place them in memory.

As received by the MIA each input word has the following format:



P = PARITY

IUA = Interface Unit Address

Bits 0-7 and 24-27 are used only in the validity checks; bits 8-23 represent the 16 bits of data that are to be saved in main memory.

Execution of this instruction is as described in paragraph 3.4.2 and in general proceeds as follows:

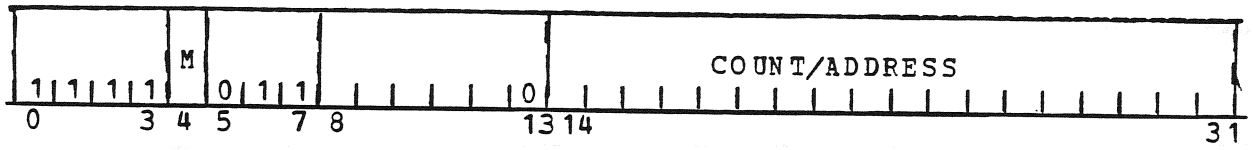
1. The input transfer count is extracted from the instruction. This count represents 1 less than the number of input words to be accepted, and these represent the number of 16 bit main memory halfwords accepted. Thus a transfer count of Ø corresponds to acceptance of 1 input. A transfer count of 31 corresponds to acceptance of 32 inputs.

2. The main memory address for the first input word is computed from the sum of the present contents of the current BCE's base register and the displacement field, bits 8 thru 15.

This displacement may have any value in the range 0 to 255. Note that this first address is the main memory halfword address for the first piece of data, and may be either even or odd.

3. The BCE monitors its MIA Buffer for data inputs. The procedure used is as described in Paragraph 3.4.2. If any errors are detected, the BCE terminates the instruction and enters the Wait State. The BCE's Program Exception Bit, BCE-MSC Indicator bit, and Status Register are also set to indicate the source of the problem.

4. Upon successful reception of all specified data words, and their transfer to memory, the BCE increments its Program Counter by 1 and begins the next instruction.



M	EFFECTIVE TRANSFER COUNT	INSTRUCTION FORMAT	
0	Bits 14 thru 31 of instruction (Count Field)	#RDLI	Count
1	Bits 14 thru 31 of fullword addressed by address field plus 2 x BCE #	#RDL	Address

DESCRIPTION

Execution of this instruction is the same as that for Receive Data Short, with the following exceptions:

1. The displacement from the base is zero. The base must point to the beginning of the buffer.

2. The effective count of input words to be received can range from 0 to 262143, and may be specified by either bits 14 thru 31 of the instruction (#RDLI) or by bits 14 thru 31 of the main storage fullword addressed by bits 14 thru 31 of the instruction (#RDL). In the second case, the least significant bit of the address (the halfword selection) is ignored. In either case the effective transfer count is one less than the number of words to be received.

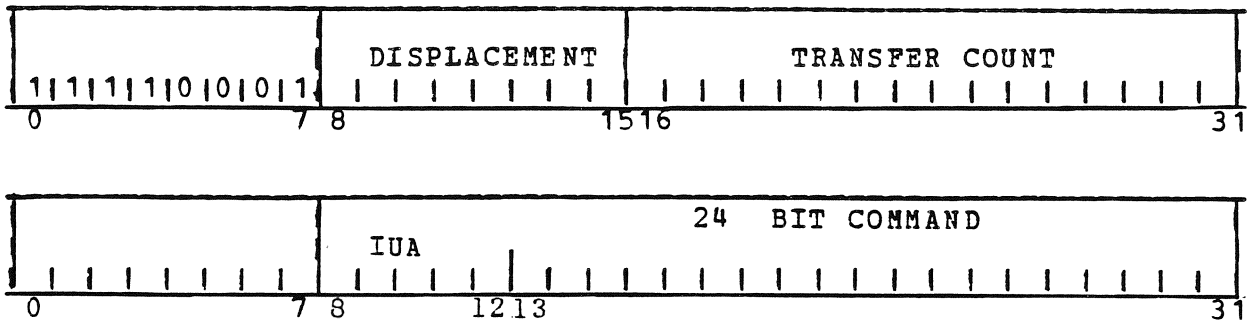
3. After completion of the instruction, the BCE's program counter is incremented by 2.

4. The setup time for #RDL is at least 49.5 usec, and may be longer if contention for memory causes the access of the word containing the transfer count to take longer than 16.5 usec.

For the #RDL instruction the address of the transfer count includes the number of the BCE executing the instruction. This allows many BCE's to execute the same program while at the same time allowing each BCE to receive a different number of inputs. Note that twice the BCE number is a fullword index.

MESSAGE IN #MIN

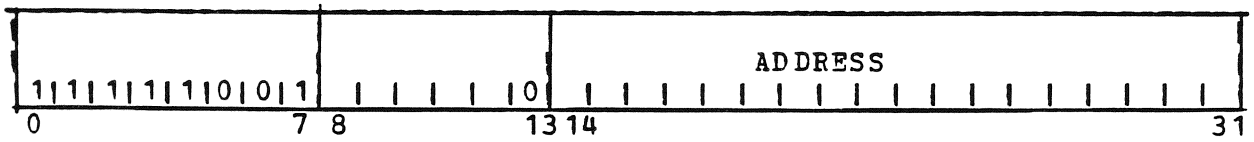
FORMAT (2 Words)



INSTRUCTION FORMAT (Bit 4 = 0)

#MIN	Displacement, Transfer Count
#MINC	IUA,COMMAND

FORMAT (1 Word)



INSTRUCTION FORMAT (Bit 4 = 1)

#MIN@	Address

NOTE: The displacement and transfer count are found at address +2XBCE#. The command is found at address +48+2XBCE#.

DESCRIPTION

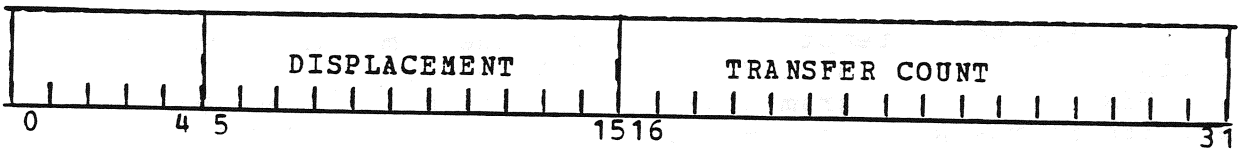
This instruction initiates the transmission of a command to a subsystem, and the acceptance of a stream of data returning in response from that subsystem. As such, it performs in one instruction approximately the same functions as a Transmit Command (#CMDI or #CMD) followed by a Receive Data (#RDS, #RDLI, or #RDL). The command followed by the returning data is termed a "message".

Message In comes in two formats, either of which allows independent specification of the command to be sent, the number of input words to be accepted, and the location of the data buffer in memory where the incoming data should be stored. The first #MIN format consists of a two word instruction. The second word contains all 24 bits of the command to be sent. The first word contains an 8

bit Base relative displacement and a 16 bit Transfer Count. The Displacement may be any positive integer between 0 and 255, and is added to the present contents of the Base Register to form the address of the data buffer. The Transfer Count may be any positive integer between 0 and 65535, and represents one less than the actual number of data half-words to be received.

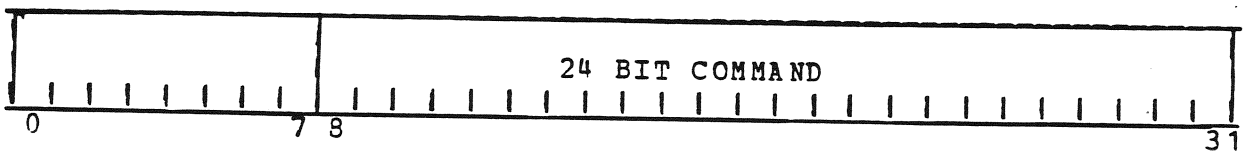
The second #MIN format allows automatic indexing by BCE number into two tables providing basically the same information found in the first format. Each table has 24 entries, one per BCE. Twice the number of the BCE executing this instruction is used as an index so that the table entries are each fullwords.

The first table starts at ADDRESS + 2, and contains the Displacement and Transfer Count. For BCE i the table entry at ADDRESS + 2 x i is formatted as:



This allows a range of displacement between 0 and 2047 and a Transfer Count between 0 and 65535.

The second table starts at ADDRESS + 50, and contains the 24 bit command. For BCE i the table entry at ADDRESS + 48 + 2 x i is formatted as:



Execution of a #MIN depends on the current mode of the BCE -- Command or Listen. In Command mode a BCE is assumed master of the bus connected to its MIA. Consequently, to receive data from some subsystem on that bus, the BCE must first issue a command telling the subsystem what type of data to return. The 24 bit command field in the #MIN contains the command. After issuing the command the BCE prepares to accept returning data.

In Listen mode, a BCE is not master of its bus, and must rely on some other BCE in another IOP to issue the command to the subsystem. A BCE in Listen mode that executes a #MIN thus does not transmit the command part of the #MIN, but instead goes directly into a MIA input monitor loop. However, like a Receive Data instruction, a #MIN executed in Listen mode will not expect to see data. Instead it will wait until it sees that some other BCE/IOP has issued a command and only then prepare to accept input data. This difference allows multiple BCE/IOPs connected to the same bus to use the same

routine to receive the same data from a single subsystem, but with only one BCE actually transmitting the commands to the subsystem.

An additional difference between the execution of a #MIN in Command or Listen mode is the setting of the BCE's IUAR. In Command mode the BCE's IUAR will be set from bits 8 to 12 of the instruction word containing the command. In Listen mode, the IUAR is unchanged. It is assumed that the BCE IUAR was previously set up to monitor the appropriate subsystem address. See Section 4 for more detail on Listen mode.

In either mode, once the command has been handled the BCE enters the same reception algorithm used for all the Receive Data instructions (see #RDS).

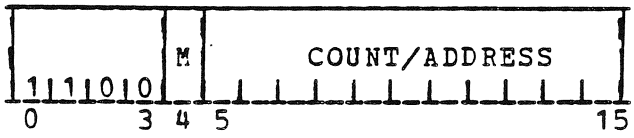
If the BCE attempts to transmit the command word but finds either its transmitter disabled or its MIA busy, the BCE will terminate the #MIN, set its Program Exception Bit to 0 (an error has occurred), set its Status Register Bit 23 to 1, set its BCE-MSC indicator, and enter the Wait State.

3.5 SPECIAL INSTRUCTIONS

The BCE instruction set contains several instructions not falling under any of the previously defined categories. These include an instruction to delay the BCE for specifiable periods of time, and an instruction to cause the BCE to enter the Wait State.

DELAY #DLYI,#DLY

FORMAT



M	Effective Count	INSTR.Format
0	TIMEOUT[1]	#DLYI Timeout
1	(PC+DISP+2XBCE#)[2]	#DLY Address

Notes:

1. Any value between 0 and 2047. This corresponds to delays from 0 to 33.78 millisec.

2. The Lower 18-bits of the fullword addressed by PC(updated)+ Displacement. This allows any count between 0 and 262143, or 0 to 4.325 sec.

DESCRIPTION

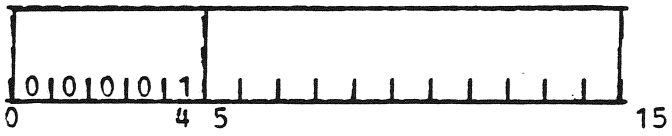
This instruction simply delays the execution of the next instruction. The time period delayed is a function of the Effective Count, with a resolution of 16.5 microseconds per count.

At the end of this delay the program counter is incremented by one, and the next instruction is executed.

PROGRAMMING NOTE

Each count of 1 represents a delay of 16.5 microseconds, the execution time of a BCE micro instruction. Each count of 2 represents a delay of 33 microseconds, the minimum time for a word transmission over a serial bus.

For the #DLY instruction, the address of the word containing the delay includes the number of the BCE executing the instruction. This allows many BCE's to execute the same program while still retaining the capability to delay for different periods. Note that twice the BCE number is a fullword index.



INSTRUCTION FORMAT

#WAT

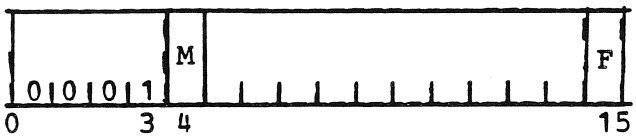
DESCRIPTION

This instruction causes the Bus Control Element (BCE) that is executing it to leave the busy state and enter the wait state. The BCE's Busy/Wait bit (in STAT4) is set to 0 (WAIT). The BCE's Program Counter is incremented by 1, but no further instructions are executed. The BCE is reset to the busy state by the Master Sequence Controller (MSC) only. Once in the Wait State, a BCE performs no actions other than the monitoring of its Busy/Wait bit for a command to re-enter the Busy State.

Paragraph 2.2 describes transitions to and from the Wait State in detail.

PROGRAMMING NOTE

While in the Wait State the BCE alters none of its programmer-visible registers. Thus the CPU is free to change, via PCI/O, any BCE register.



INSTRUCTION FORMAT

#STP FLAG

DESCRIPTION M=0

This instruction initiates execution of a special micro program to perform self tests on the hardware supporting the Bus Control Element that is executing this instruction. These tests include checks of:

- BCE Local Store
- BCE Data Flow Operations
- Ability of BCE to read and write from memory
- MIA wrap capability
- MIA Buffer

A flag of 0 causes all but the last two tests to be performed. A flag of 1 causes all tests to be run.

If an error is detected, the BCE's Program Exception bit is set to 0, bit 22 of the BCE Status Register is set to 1 (Self Test failure). The PC is incremented by 1, and the next instruction is executed.

Successful completion of this instruction causes the BCE to increment its PC by 1 and continue with the next instruction.

DESCRIPTION M=1

This instruction initiates execution of a special micro program to perform self test on the associated MIA Parity Checker. Before execution of this instruction parity must be enabled and the BCE executing this instruction must have its transmitter disabled. If the transmitter is enabled the BCE will set no-go, go to wait and set the BCE Status Register Bit 22 to 1. If the transmitter is disabled the micro program will do a transmit. This will allow data to be sent through the Parity Checking Circuitry, but will not be transmitted on the Bus. If bad parity is detected an External 1 interrupt is generated and the External 1 Status Register will indicate a MIA Parity Error.

Note: It is recommended that this test be run both forcing and not forcing bad parity to the MIA's so the Check Circuitry is fully tested.

(This assembler mnemonic supports only the M=0 OP code.)

4.0 LISTEN MODE

In the Space Shuttle configuration there can be up to five separate CPU/IOP pairs connected to the same set of busses. When running in redundant mode, all CPU/IOPs want to receive exactly the same copies of all input data from relevant sensors so that the resulting outputs should all match. This presents problems. First, since a sensor in normal operation is tied to only one bus, only one IOP at a time can send it the requisite commands to return data. Any attempt by two or more IOPs to send commands over the same bus at the same time results in interference and garbling of the bus signals. Thus, if each IOP is to send a command to the same sensor telling it to return data, there must be some definite time sequence that prevents conflicts. Such sequencing is not only difficult to achieve, but also makes it impossible to guarantee that exactly the same data reaches each CPU/IOP.

The BCE Listening Mode solves this problem by allowing only one BCE/ IOP to place subsystem commands on a bus, but at the same time allow it to inform all other BCE/IOPs connected to the bus that a certain subsystem is to be commanded and that they should set up to handle the returning data. This technique not only eliminates the sequencing problem mentioned above, but also allows all GPC/IOPs to receive exactly the same data at exactly the same time.

The following paragraphs describe Listen Mode, first by example and then in detail. Note that this is not the only method of Listen Mode implementation.

4.1 SAMPLE OPERATION IN LISTEN MODE

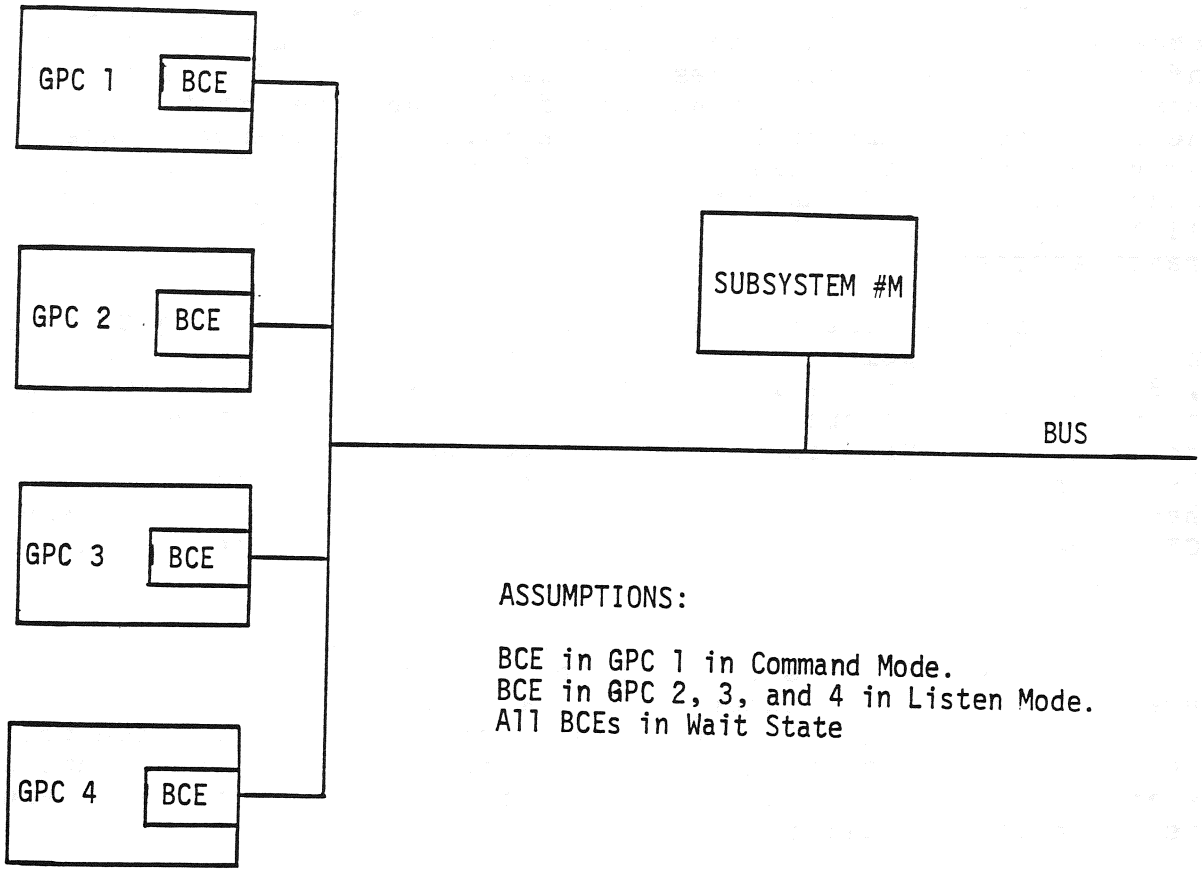
Figure 4.1 diagrams a configuration of redundant IOPs that will use a Listen Mode to accept data from a single subsystem, Subsystem M. Also included is a reference BCE program segment. The BCE in GPC 1 is assumed to be configured in Command Mode. The equivalent BCEs in GPCs 2, 3, and 4 are in Listen Mode. All BCEs are assumed in Wait State initially.

Before a BCE in Listen Mode can accept a command from another BCE in a separate GPC, it can be executing a #WIX instruction -- Wait for Index. This instruction causes a BCE in Listen Mode to monitor its bus for a command from another GPC. The only way a BCE can be executing a #WIX is if its MSC has loaded the PC (via a @LBP or CPU PCO) and started it (via a @SIO) at a memory location containing a #WIX. Once a BCE has been started in this fashion, it will return to the Wait State only when it has either executed a standard #WAT instruction or it has encountered an error. Any attempt by the MSC to change the operation of the BCE after it has been started will be aborted, with error flags set in the MSC's status register.

Since the operational mode of a BCE in an operational sequence will very rarely be changed, the MSC in a GPC will initialize its Listen Mode BCEs once after a reconfiguration, and will leave them alone unless they execute a #WAT or error terminate. For this example

we assume that the BCEs in GPCs 2, 3 and 4 have been initialized to execute a "WIX TABLE" instruction in the near past, and are now watching their bus for a Listen Command.

At some point in time the MSC in GPC 1 will determine that it is time to receive data from Subsystem M and that its BCE is in Command Mode. The MSCs in the other GPCs will also come to the conclusion at about the same time that it is time for Subsystem M to give data, but since their BCEs are in Listen Mode, they do nothing active to get it. To get the needed data, the MSC in GPC 1 initializes its BCE to execute the program at location "START". When the MSC executes a @SIO, the BCE in GPC 1 enters the Busy State and begins execution at location START. The first instruction then tells the BCE to place on the bus a "Listen Command". This command is meant for all BCEs on the bus that are in Listen Mode and are executing a #WIX. It is distinguished from all other traffic on the bus by a unique Interface Unit Address (IUA) not used by any subsystem. This IUA is termed a "Common IOP Address", and in binary is 0 1 0 0 0.



REFERENCE PROGRAM SEQUENCE

```
START    #CMDI     IUA = Common IOP Address,
                   Device # = M, Index = 2.
         #DLYI
         #DLYI
PGM2     #MIN      IUA = M, . . .
         #WIX      Table

TABLE    DC        A (PGM0)
         DC        A (PGM1)
         DC        A (PGM2)
                .
                .
                .
```

Figure 4.1. Listen Mode Configuration

When a BCE in Listen Mode detects a word on the bus that has command sync and the Common IOP Address, they accept it and use the information to set themselves to a new program. The information in this word consists of the number of the subsystem that will originate the information, and an index into a table of branch address. The beginning of the table is specified by the #WIX instruction. We assume that all GPCs have the same programs located in memory so that all listening BCEs will reference the same table and pull out the same branch address.

In this case the subsystem number in the Listen Command is M, and the index points to entry 2 in the table. When the BCEs in GPCs 2, 3 and 4 receive the Listen Command, they save the value M in their Interface Unit Address Register, and branch through the table to location PGM2. This is a #MIN instruction, and since the BCEs are in Listen Mode, instructs the BCEs to monitor the bus for the command that the BCE in GPC 1 will eventually send to Subsystem M. When these BCEs receive this command, they will begin monitoring the bus for the returning data.

After the BCE in GPC 1 has sent the Listen Command, it executes some delay instructions to allow the Listening BCEs time to receive the command, interpret it, and get set up in #MIN. Two delay instructions are listed in the sample program segment since both #CMDI and #MIN must start on an even half word boundary. The amount of time delayed must be based on an analysis of how long it takes the Listening BCEs to get set up.

After the delays, the BCE in GPC 1 executes the #MIN instruction. Even though it is the same instruction executed by the Listening BCEs, the fact that the BCE is in Command Mode causes a different execution. The BCE will place on the bus the command to Subsystem M, telling it to return the appropriate data. The BCE will then prepare itself to handle this data.

When Subsystem M places its data on the bus, all four BCEs are prepared to accept it, and consequently receive exactly the same data at exactly the same time.

After successful reception of the data from Subsystem M, all BCEs execute the #WIX instruction. For the command BCE this is equivalent to a #WAT, and the BCE re-enters the Wait State where it will await another command from its own MSC. For the Listening BCEs, the #WIX instruction simply places them back into a loop where they are monitoring the bus for another Listen Command.

The above sequence obviously can be elaborated to include multiple #MINs from the same subsystem, instructions to change the Base or Time Out Register, or do other functions, such as set the BCE-MSC indicator bits to indicate completion of a BCE Listen Program. The extensions, however, are application dependent, and do not change the basic operation described above.

One of the important aspects of this example is that the software stored in each computer is identical. The commanding and listening BCEs use exactly the same instructions to receive the same data from the same subsystem. Only the mode of the BCEs was different.

4.2 INITIALIZATION INTO LISTEN MODE

Listen Mode is a mode of BCE operation, not a state. It affects how a BCE interprets instructions, but does not of itself represent a completely different state of BCE execution.

A BCE is in Listen Mode when its MIA transmitter is disabled and its receiver enabled. It may be changed at any time without harm while a BCE is in Halt state. Once a BCE is in Busy State, changing the mode of that BCE may cause indeterminant BCE actions. It is recommended practice that a BCE be halted before a mode change, and then enabled after the mode change is complete. This will prevent any possible indeterminant BCE activity.

The signals that may change the MIA's status are as follows:

1. CPU PCO's allow selective set/reset of each MIA's transmitter and receiver.

2. XMIT Disable Discrete disables all MIA transmitters.

3. Input discrete 13 disables MIA transmitters 10-13.

4. Input discrete 14 disables MIA transmitters 14-17 and 20-23.

4.3 DIFFERENCES IN INSTRUCTION EXECUTION

A BCE enters the Busy State only from Wait and only when the MSC has executed a @SIO. This transition is independent of the BCE's mode. The distinction between Listen and Command mode becomes significant only after the BCE has entered the Busy State. As demonstrated in the previous example, this distinction reflects itself in how various instructions are executed. Table 4.1 summarizes those BCE instructions that are influenced by mode. Each difference is discussed separately in the following paragraphs.

The Wait for Index instruction (#WIX) was designed for Listen Mode. In Command Mode #WIX is equivalent to a Wait instruction -- a #WAT. In Listen Mode a #WIX places the BCE in a loop where it is monitoring its MIA for a Listen Command (Interface Unit Address = 0 1 0 0 0). When it receives such a command it branches through a table of addresses. The BCE stays in the Busy State during the entire sequence, and will leave the Busy State only if it is halted, it receives a Listen Command that directs it eventually to a #WAT instruction, or it receives a Listen Command, starts executing a program, but is stopped by an error from further execution.

In Listen Mode a BCE's transmitter is not enabled. Consequently, it is not able to transmit either commands or data. This affects the execution of #CMDI, #CMD, #MIN, #MOUT, #TDS, #TDLI, and #TDL.

Any attempt by a BCE to execute a #MOUT, #TDS, #TDLI, or #TDL while in Listen Mode will be met by error terminating the BCE, i.e. setting Status Bit 23, resetting the Program Exception Bit to 0, and entering the Wait State. A BCE in Listen Mode should never be told to transmit data.

The instructions #CMDI, #CMD, and #MIN are not error terminated when executed by a BCE in Listen Mode. Instead, #CMDI and #CMD are treated as no-ops (#CMDI sets the BCE IUAR), no error indicators are set, and the BCE continues with the next instruction. Likewise in #MIN the transmission of the command is suppressed -- the BCE branches into its receive mode. The reason for this difference is to allow the same program segments to be used by BCEs in both Command and Listen Modes to receive the same data as demonstrated in the example of Figure 4.1.

Execution of #RDS, #RDLI, #RDL, and the receive part of #MIN also differ between Command and Listen Mode, primarily in accepting the first data input. In Command Mode, when these instructions enter the receive algorithm, they immediately start waiting for the first data word, and timing out its arrival by decrementing a timer that had been set to the value in the MTO Register. If the first input the BCE finds in the MIA Buffer is a command, the BCE will discard the command, reset the timer, and start looking for data again. This allows the BCE to ignore the command that was sent to the subsystem but was also echoed back by the MIA to the BCE's MIA Buffer.

A BCE in Listen Mode goes through a different sequence. Rather than immediately waiting for data, the BCE waits (indefinitely) for a bus word having command sync, good parity, etc., and an IUA that matches the BCE's IUAR. This command should be the command being sent by the other BCE/IOP in Command Mode to the subsystem, telling it to return data. The BCE uses the arrival of this command as a signal to set its timers and start watching for arrival of the first input of data. This allows the Listening BCE to accurately time out the first data word, and detect if that word is excessively delayed in its arrival.

After arrival of the first word all BCEs use the same reception algorithm.

TABLE 4.1

DIFFERENCES IN INSTRUCTION EXECUTIONS
DUE TO BCE MODE

BCE INSTRUCTION	EXECUTION IN COMMAND MODE	EXECUTION IN LISTEN MODE
#WIX	Enter Wait State	Wait for Listen Command (Stay Busy)
#CMD	Transmit Command	No-operation
#CMDI	Transmit Command	Suppress Command but Load IUAR
#MIN	Transmit Command, Receive Data	Wait for Command, Receive Data
#RDS, #RDLI, #RDL	Receive Data	Wait for Command, Receive Data
#MOUT	Transmit Command, Transmit Data	Set Error Bits, Enter Wait State
#TDS, #TDLI, #TDL	Transmit Data	Set Error Bits, Enter Wait State

APPENDIX A

IOP BCE INSTRUCTION REPERTOIRE

NOTE: See Figure 1.3 and 1.4 for Formats.