


SPERRY UNIVAC  
1100/80 Systems  
**Processor and Storage**  
Programmer Reference

This document contains the latest information available at the time of publication. However, Sperry Univac reserves the right to modify or revise its contents. To ensure that you have the most recent information, contact your local Sperry Univac representative.

Sperry Univac is a division of Sperry Rand Corporation.

AccuScan, FASTERAND, PAGEWRITER, SPERRY UNIVAC, UNISCOPE, UNISERVO, UNIVAC, and  are trademarks of the Sperry Rand Corporation.







## Contents

### Page Status Summary

### Contents

<b>1. Introduction</b>	1-1
<b>1.1. GENERAL</b>	1-1
<b>1.2. 1100/80 SYSTEM CONFIGURATIONS</b>	1-1
1.2.1. Central Processor Unit	1-1
1.2.2. Main Storage	1-5
1.2.3. Input/Output Unit (IOU)	1-5
1.2.4. System Console	1-6
1.2.5. System Transition Unit (STU)	1-6
1.2.6. System Maintenance Unit	1-7
1.2.7. Auxiliary Storage and Peripheral Subsystems	1-7
<b>2. Processing Unit</b>	2-1
<b>2.1. GENERAL</b>	2-1
<b>2.2. CONTROL SECTION</b>	2-1
2.2.1. Control Section Operation	2-1
2.2.2. Instruction Repertoire	2-1
2.2.3. Control Registers	2-2
2.2.4. Data Shift/Complement/Store Operation	2-2
<b>2.3. ARITHMETIC SECTION</b>	2-2
<b>2.4. MAINTENANCE SECTION</b>	2-2
<b>2.5. INPUT/OUTPUT UNIT (IOU)</b>	2-2
<b>3. Storage</b>	3-1
<b>3.1. GENERAL</b>	3-1
<b>3.2. MAIN STORAGE</b>	3-1
3.2.1. Main Storage Addressing	3-1

3.2.2. MSU Address Assignments	3-2
3.2.3. Fixed Address Assignments	3-5
<b>3.3. BUFFER STORAGE</b>	<b>3-7</b>
3.3.1. Set Associative Addressing	3-7
3.3.2. Address Interleave	3-9
<b>3.4. CONTROL STORAGE</b>	<b>3-10</b>
3.4.1. Control Register Selection Designator	3-10
3.4.2. Control Register Address Assignments	3-10
3.4.2.1. Storage for MSR Value - Address 0143	3-11
3.4.2.2. User Index (X) Register - Addresses 0001 - 0017	3-12
3.4.2.3. User Accumulator (A) Registers - Addresses 0014 - 0033	3-13
3.4.2.4. User Unassigned Registers - Addresses 0034 - 0037	3-13
3.4.2.5. EXEC Bank Descriptor Table Pointer Register - Address 0040	3-13
3.4.2.6. Immediate Storage Check Interrupts - Addresses 0041 - 0042	3-13
3.4.2.7. Normal Interrupts - Addresses 0043 - 0044	3-13
3.4.2.8. User Bank Descriptor Table Pointer Register - Address 0045	3-13
3.4.2.9. Bank Descriptor Index Registers - Addresses 0046 - 0047	3-13
3.4.2.10. Quantum Timer - Address 0050	3-13
3.4.2.11. Guard Mode - Addresses 0051 - 0053	3-13
3.4.2.12. Immediate Storage Check Status - Address 0054	3-14
3.4.2.13. Normal Status - Address 0055	3-14
3.4.2.14. Unassigned Registers - Addresses 0056 - 0067	3-14
3.4.2.15. Jump History Stack - Addresses 0070 - 0077	3-14
3.4.2.16. Real-Time Clock Register (RO) - Address 0100	3-14
3.4.2.17. User (R1) Repeat Count Register - Address 0101	3-14
3.4.2.18. User (R2)/Mask Register - Address 0102	3-14
3.4.2.19. User (R2-R5)/Staging Registers (SR1-SR3) - Address 0103 - 0105	3-15
3.4.2.20. User (R6-R9)/J-Registers (JO-J3) - Address 0106 - 0111	3-15
3.4.2.21. User R-Registers (R10-R15) - Addresses 0112 - 0117	3-15
3.4.2.22. Executive (RO) R-Register - Address 0120	3-15
3.4.2.23. Executive (R1) Repeat Count Register - Address 0121	3-15
3.4.2.24. Executive (R2)/Mask Register - Address 0122	3-15
3.4.2.25. Executive (R3-R5)/Staging Registers (SR1-SR3) - Addresses 0123 - 0125	3-15
3.4.2.26. Executive (R6-R9)/J-Registers (JO-J3) - Addresses 0126 - 0131	3-15
3.4.2.27. Executive R-Registers (R10-R15) - Addresses 0132 - 0137	3-15
3.4.2.28. Executive Index Registers (X1-X15) - Addresses 0141 - 0157	3-16
3.4.2.29. Executive Accumulator Registers (AO-A15) - Addresses 0154 - 0173	3-16
3.4.2.30. Executive Unassigned Registers - Addresses 0140, 0174 - 0177	3-16
3.4.2.31. Control Register Protection	3-16
<b>4. Processor</b>	<b>4-1</b>
<b>4.1. ARITHMETIC SECTION</b>	<b>4-1</b>
4.1.1. General Operation	4-1
4.1.1.1. Data Word	4-1
4.1.1.2. Data Word Complement	4-2
4.1.1.3. Absolute Values	4-2
4.1.2. Microprogrammed Control	4-2
4.1.3. Main Adder Characteristics	4-2
4.1.4. Fixed-Point Single- or Double-Precision Add or Subtract Overflow and Carry	4-2

4.1.4.1. Overflow	4-3
4.1.4.2. Carry	4-3
4.1.4.3. Arithmetic Interrupt	4-3
4.1.5. Fixed-Point Division	4-3
4.1.6. Fixed-Point Multiplication	4-4
4.1.7. Floating-Point Arithmetic	4-4
4.1.8. Floating-Point Numbers and Word Formats	4-4
4.1.8.1. Single-Precision Floating-Point Numbers	4-6
4.1.8.2. Double-Precision Floating-Point Numbers	4-6
4.1.8.3. Negative Floating-Point Numbers	4-6
4.1.8.4. Residue	4-7
4.1.9. Normalized/Unnormalized Floating-Point Numbers	4-7
4.1.10. Floating-Point Characteristic Overflow/Underflow	4-7
4.1.10.1. Floating-Point Characteristic Overflow	4-7
4.1.10.2. Floating-Point Characteristic Underflow	4-8
4.1.10.3. Floating-Point Divide Fault	4-8
4.1.11. Fixed-Point to Floating-Point Conversion	4-8
4.1.12. Floating-Point Addition	4-8
4.1.12.1. Double-Precision Floating-Point Addition	4-9
4.1.13. Floating-Point Subtraction (Add Negative)	4-9
4.1.14. Floating-Point Multiplication	4-9
4.1.15. Floating-Point Division	4-9
4.1.16. Floating-Point Zero	4-9
4.1.17. Byte Instructions	4-10
<b>4.2. CONTROL SECTION</b>	4-10
4.2.1. Instruction Word Format	4-10
4.2.2. Instruction Word Fields	4-11
4.2.2.1. Use of the f-Field	4-11
4.2.2.2. Description of the j-Field	4-11
4.2.2.2.1. Use of the j-Field as an Operand Qualifier	4-11
4.2.2.2.2. Use of the j-Field to Specify Character Addressing	4-14
4.2.2.2.3. Use of j-Field as Partial Control Register Address	4-19
4.2.2.2.4. Use of j-Field as Minor Function Code	4-19
4.2.2.3. Uses of the a-Field	4-19
4.2.2.3.1. Use of the a-Field to Reference an A-Register	4-19
4.2.2.3.2. Use of the a-Field to Reference an X-Register	4-20
4.2.2.3.3. Use of the a-Field to Reference an R-Register	4-20
4.2.2.3.4. Use of the a-Field to Reference a Jump Key	4-20
4.2.2.3.5. Use of the a-Field to Reference Halt Keys	4-20
4.2.2.3.6. Use of the a-Field as Minor Function Code	4-20
4.2.2.4. Use of the j- and a-Fields to Specify GRS Control Register Address	4-21
4.2.2.5. Use of the x-Field	4-21
4.2.2.6. Use of the h-Field	4-22
4.2.2.7. Use of the i-Field	4-22
4.2.2.8. Description of the u-Field	4-23
4.2.2.8.1. Use of the u-Field as an Operand Address Designator	4-24
4.2.2.8.2. Use of the u-Field as an Operand Designator	4-24
4.2.2.8.3. Use of the u-Field as a Shift Count Designator	4-24
4.2.2.8.4. Restrictions on the Use of the u-Field	4-25
<b>5. Instruction Repertoire</b>	5-1

<b>5.1. INTRODUCTION</b>	5-1
<b>5.2. LOAD INSTRUCTIONS</b>	5-2
5.2.1. Load A - L,LA 10	5-2
5.2.2. Load Negative A - LN,LNA 11	5-2
5.2.3. Load Magnitude A - LM,LMA 12	5-2
5.2.4. Load Negative Magnitude A - LNMA 13	5-2
5.2.5. Load R - L,LR 23	5-3
5.2.6. Load X Modifier - LXM 26	5-3
5.2.7. Load X - L,LX 27	5-3
5.2.8. Load X Increment - LXI 46	5-3
5.2.9. Double Load A - DL f = 71 j = 13	5-3
5.2.10. Double-Load Negative A - DLN 71,14	5-3
5.2.11. Double Load Magnitude A - DLM 71,15	5-4
<b>5.3. STORE INSTRUCTIONS</b>	5-4
5.3.1. Store A - S,SA 01	5-4
5.3.2. Store Negative A - SN,SNA 02	5-4
5.3.3. Store Magnitude A - SM,SMA 03	5-4
5.3.4. Store R - S,SR 04	5-5
5.3.5. Store Constant Instructions - XX 05; a = 00-07	5-5
5.3.6. Store X - S,SX 06	5-5
5.3.7. Double Store A - DS 71,12	5-5
5.3.8. Block Transfer - BT 22	5-6
<b>5.4. FIXED-POINT ARITHMETIC INSTRUCTIONS</b>	5-6
5.4.1. Add to A - A,AA 14	5-7
5.4.2. Add Negative to A - AN,ANA 15	5-7
5.4.3. Add Magnitude to A - AM,AMA 16	5-7
5.4.4. Add Negative Magnitude to A - ANM,ANMA 17	5-7
5.4.5. Add Upper - AU 20	5-8
5.4.6. Add Negative Upper - ANU 21	5-8
5.4.7. Add to X - A,AX 24	5-8
5.4.8. Add Negative to X - AN,ANX 25	5-8
5.4.9. Multiply Integer - MI 30	5-8
5.4.10. Multiply Single Integer - MSI 31	5-8
5.4.11. Multiply Fractional - MF 32	5-9
5.4.12. Divide Integer - DI 34	5-9
5.4.13. Divide Single Fractional - DSF 35	5-9
5.4.14. Divide Fractional - DF 36	5-10
5.4.15. Double-Precision Fixed-Point Add - DA 71,10	5-10
5.4.16. Double-Precision Fixed-Point Add Negative - DAN 71,11	5-10
5.4.17. Add Halves - AH 72,04	5-10
5.4.18. Add Negative Halves - ANH 72,05	5-10
5.4.19. Add Thirds - AT 72,06	5-11
5.4.20. Add Negative Thirds - ANT 72,07	5-11
<b>5.5. FLOATING-POINT ARITHMETIC</b>	5-11
5.5.1. Floating Add - FA 76,00	5-11
5.5.2. Floating Add Negative - FAN 76,01	5-12
5.5.3. Double-Precision Floating Add - DFA 76,10	5-12
5.5.4. Double-Precision Floating Add Negative - DFAN 76,11	5-13

5.5.5. Floating Multiply - FM	76,02	5-13
5.5.6. Double-Precision Floating Multiply - DFM	76,12	5-14
5.5.7. Floating Divide - FD	76,03	5-15
5.5.8. Double-Precision Floating Divide - DFD	76,13	5-15
5.5.9. Load and Unpack Floating - LUF	76,04	5-16
5.5.10. Double Load and Unpack Floating - DFU	76,14	5-16
5.5.11. Load and Convert to Floating - LCF	76,05	5-16
5.5.12. Double Load and Convert to Floating - DFP, DLCF	76,15	5-17
5.5.13. Floating Expand and Load - FEL	76,16	5-18
5.5.14. Floating Compress and Load - FCL	76,17	5-18
5.5.15. Magnitude of Characteristic Difference to Upper - MCDU	76,06	5-19
5.5.16. Characteristic Difference to Upper - CDU	76,07	5-19
<b>5.6. SEARCH AND MASKED-SEARCH INSTRUCTIONS</b>		5-19
5.6.1. Search Equal - SE	62	5-22
5.6.2. Search Not Equal - SNE	63	5-22
5.6.3. Search Less Than or Equal - Search Not Greater - SLE,SNG	64	5-22
5.6.4. Search Greater - SG	65	5-23
5.6.5. Search Within Range - SW	66	5-23
5.6.6. Search Not Within Range - SNW	67	5-24
5.6.7. Mask Search Equal - MSE	71,00	5-24
5.6.8. Mask Search Not Equal - MSNE	71,01	5-25
5.6.9. Mask Search Less Than or Equal - Mask Search Not Greater - MSLE,MSNG		5-25
5.6.10. Mask Search Greater - MSG	71,03	5-26
5.6.11. Masked Search Within Range - MSW	71,04	5-26
5.6.12. Masked Search Not Within Range - MSNW	71,05	5-27
5.6.13. Masked Alphanumeric Search Less Than or Equal - MASL	71,06	5-27
5.6.14. Masked Alphanumeric Search Greater - MASG	71,07	5-28
<b>5.7. TEST (OR SKIP) INSTRUCTIONS</b>		5-28
5.7.1. Test Even Parity - TEP	44	5-28
5.7.2. Test Odd Parity - TOP	45	5-29
5.7.3. Test Less Than or Equal to Modifier - TLEM	47	5-29
5.7.4. Test Zero - TZ	50	5-29
5.7.5. Test Nonzero - TNZ	51	5-30
5.7.6. Test Equal - TE	52	5-30
5.7.7. Test Not Equal - TNE	53	5-30
5.7.8. Test Less Than or Equal - Test Not Greater - TLE,TNG	54	5-30
5.7.9. Test Greater - TG	55	5-31
5.7.10. Test Within Range - TW	56	5-31
5.7.11. Test Not Within Range - TNW	57	5-31
5.7.12. Test Positive - TP	60	5-32
5.7.13. Test Negative - TN	61	5-32
5.7.14. Double-Precision Test Equal - DTE	71,17	5-32
<b>5.8. SHIFT INSTRUCTIONS</b>		5-33
5.8.1. Single Shift Circular - SSC	73,00	5-34
5.8.2. Double Shift Circular - DSC	73,01	5-34
5.8.3. Single Shift Logical - SSL	73,02	5-34
5.8.4. Double Shift Logical - DSL	73,03	5-35
5.8.5. Single Shift Algebraic - SSA	73,04	5-35
5.8.6. Double Shift Algebraic - DSA	73,05	5-35

5.8.7. Load Shift and Count - LSC	73,06	5-35
5.8.8. Double Load Shift and Count - DLSC	73,07	5-36
5.8.9. Left Single Shift Circular - LSSC	73,10	5-36
5.8.10. Left Double Shift Circular - LDSC	73,11	5-36
5.8.11. Left Single Shift Logical - LSSL	73,12	5-36
5.8.12. Left Double Shift Logical - LDSL	73,13	5-37
<b>5.9. UNCONDITIONAL JUMP INSTRUCTION</b>		5-37
5.9.1. Store Location and Jump - SLJ	72,01	5-37
5.9.2. Load Modifier and Jump - LMJ	74,13	5-38
5.9.3. Allow All Interrupts and Jump - AAIJ	74,07	5-38
<b>5.10. BANK DESCRIPTOR SELECTION INSTRUCTIONS</b>		5-38
5.10.1. Load Bank and Jump - LBJ	07,17	5-38
5.10.2. Load I-Bank Base and Jump - LIJ	07,13	5-39
5.10.3. Load D-Bank Base and Jump - LDJ	07,12	5-39
<b>5.11. CONDITIONAL JUMP INSTRUCTIONS</b>		5-40
5.11.1. Jump Greater and Decrement - JGD	70	5-40
5.11.2. Double-Precision Jump Zero - DJZ	71,16	5-40
5.11.3. Jump Positive and Shift - JPS	72,02	5-40
5.11.4. Jump Negative and Shift - JNS	72,03	5-40
5.11.5. Jump Zero - JZ	74,00	5-41
5.11.6. Jump Nonzero - JNZ	74,01	5-41
5.11.7. Jump Positive - JP	74,02	5-41
5.11.8. Jump Negative - JN	74,03	5-41
5.11.9. Jump - Jump Keys - J,JK	74,04	5-41
5.11.10. Halt Jump - Halt Keys and Jump - HJ,HKJ	74,05	5-42
5.11.11. Jump No Low Bit - JNB	74,10	5-42
5.11.12. Jump Low Bit - JB	74,11	5-42
5.11.13. Jump Modifier Greater and Increment - JMGI	74,12	5-42
5.11.14. Jump Overflow - JO	74,14; a = 0	5-43
5.11.15. Jump Floating Underflow - JFU	74,14; a = 1	5-43
5.11.16. Jump Floating Overflow - JFO	74,14; a = 2	5-43
5.11.17. Jump Divide Fault - JDF	74,14; a = 3	5-43
5.11.18. Jump No Overflow - JNO	74,15; a = 0	5-43
5.11.19. Jump No Floating Underflow - JNFU	74,15; a = 1	5-44
5.11.20. Jump No Floating Overflow - JNFO	74,15; a = 2	5-44
5.11.21. Jump No Divide Fault - JNDF	74,15; a = 3	5-44
5.11.22. Jump Carry - JC	74,16	5-44
5.11.23. Jump No Carry - JNC	74,17	5-44
<b>5.12. LOGICAL INSTRUCTIONS</b>		5-44
5.12.1. Logical OR - OR	40	5-45
5.12.2. Logical Exclusive OR - XOR	41	5-45
5.12.3. Logical AND - AND	42	5-45
5.12.4. Masked Load Upper - MLU	43	5-46
<b>5.13. MISCELLANEOUS INSTRUCTIONS</b>		5-46
5.13.1. Load DR Designators - LPD	07,14	5-46
5.13.2. Store DR Designators - SPD	07,15	5-46
5.13.3. Execute - EX	72,10	5-47

5.13.4. Executive Request - ER	72,11	5-47
5.13.5. Test and Set - TS	73,17; a = 0	5-48
5.13.6. Test and Set and Skip - TSS	73,17; a = 1	5-48
5.13.7. Test and Clear and Skip - TCS	73,17; a = 2	5-48
5.13.8. No Operation - NOP	74,06	5-48
5.13.9. Store Register Set - SRS	72,16	5-49
5.13.10. Load Register Set - LRS	72,17	5-49
5.13.11. Test Relative Address - TRA	72,15	5-49
5.13.12. Increase Instructions - XX	05; a = 10-17	5-50
<b>5.14. BYTE INSTRUCTIONS</b>		<b>5-51</b>
5.14.1. Byte Move - BM	33,00	5-55
5.14.2. Byte Move With Translate - BMT	33,01	5-55
5.14.3. Byte Translate and Compare - BTC	33,03	5-56
5.14.4. Byte Compare - BC	33,04	5-57
5.14.5. Edit - EDIT	33,07	5-57
5.14.5.1. Function Byte		5-58
5.14.5.2. Subfunction Byte		5-60
5.14.6. Byte to Binary Single Integer Convert - BI	33,10	5-61
5.14.7. Byte to Binary Double Integer Convert - BDI	33,11	5-63
5.14.8. Binary Single Integer to Byte Convert - IB	33,12	5-63
5.14.9. Binary Double Integer to Byte Convert - DIB	33,13	5-64
5.14.10. Byte to Single Floating Convert - BF	33,14	5-64
5.14.11. Byte to Double Floating Convert - BDF	33,15	5-65
5.14.12. Single Floating to Byte Convert - FB	33,16	5-65
5.14.13. Double Floating to Byte Convert - DFB	33,17	5-65
5.14.14. Byte Add - BA	37,06	5-66
5.14.15. Byte Add Negative - BAN	37,07	5-66
<b>5.15. EXECUTIVE INSTRUCTION REPERTOIRE</b>		<b>5-66</b>
5.15.1. Prevent All Interrupts and Jump - PAIJ	72,13	5-67
5.15.2. Enable/Disable Dayclock - EDC,DDC	73,14, 11-12	5-67
5.15.3. Select Dayclock - SDC	73,14, 13	5-67
5.15.4. Select Interrupt Locations - SIL	73,15, 00	5-67
5.15.5. Load Breakpoint Register - LBX	73,15, 02	5-67
5.15.6. Store Processor ID - SPID	73,15, 05	5-67
5.15.7. Load Quantum Timer - LQT	73,15, 03	5-68
5.15.8. Load Base - LB	73,15 10	5-68
5.15.9. Load Limits - LL	73,15, 11	5-68
5.15.10. Load Addressing Environment - LAE	73,15, 12	5-68
5.15.11. Store Quantum Time - SQT	73,15, 13	5-68
5.15.12. Load Designator Register - LD	73,15, 14	5-68
5.15.13. Store Designator Register - SD	73,15, 15	5-69
5.15.14. User Return - UR	73,15, 16	5-69
5.15.15. Reset Auto-Recovery Timer - RAT	73,15, 06	5-69
5.15.16. Toggle Auto-Recovery Path - TAP	73, 5, 07	5-69
5.15.17. Store System Status - SSS	73,15, 17	5-69
5.15.18. Diagnostics -	73,14, 14 - 17	5-69
5.15.19. Input/Output Instructions		5-70
<b>6. Input/Output</b>		<b>6-1</b>

<b>6.1. INTRODUCTION</b>	6-1
<b>6.2. FUNCTIONAL CHARACTERISTICS</b>	6-1
6.2.1. Channels	6-3
6.2.2. Subchannels	6-5
<b>6.3. CONTROL OF INPUT/OUTPUT DEVICES</b>	6-5
6.3.1. Input/Output Device Addressing	6-5
6.3.2. States of the Input/Output System	6-6
6.3.3. Condition Codes	6-10
6.3.4. Instruction Format and Channel Address Word	6-10
6.3.5. Instruction Operation	6-17
<b>6.4. I/O INSTRUCTIONS</b>	6-17
6.4.1. Sense Release - SRL 75,00	6-17
6.4.1.1. Byte or Block Multiplexer Channel Operation	6-18
6.4.1.2. Word Channel Operation	6-18
6.4.2. Start I/O Fast Release - SIOF 75,01	6-19
6.4.2.1. Byte or Block Multiplexer Channel Operation	6-19
6.4.2.2. Word Channel Operation	6-20
6.4.3. Test I/O - TIO 75,02	6-20
6.4.3.1. Byte or Block Multiplexer Channel Operation	6-21
6.4.3.2. Word Channel Operation	6-21
6.4.4. Test Subchannel - TSC 75,03	6-21
6.4.4.1. Byte or Block Multiplexer Channel	6-21
6.4.4.2. Word Channel Operation	6-22
6.4.5. Halt Device - HDV 75,04	6-22
6.4.5.1. Byte or Block Multiplexer Channel Operation	6-23
6.4.5.2. Word Channel Operation	6-24
6.4.6. Halt Channel - HCH 75,05	6-24
6.4.6.1. Byte or Block Multiplexer Channel Operation	6-24
6.4.6.2. Word Channel Operation	6-25
6.4.7. Load Channel Register - LCR 75,10	6-25
6.4.7.1. Byte and Block Multiplexer Channel	6-25
6.4.7.2. Word Channel Operation	6-26
6.4.8. Load Table Control Words - LTCW 75,11	6-26
6.4.8.1. Byte and Block Multiplexer Channel	6-26
6.4.8.2. Word Channel Operation	6-27
<b>6.5. EXECUTION OF I/O OPERATIONS</b>	6-27
6.5.1. Channel Command Word	6-28
6.5.2. CCW Completion	6-31
<b>6.6. COMMAND CODE</b>	6-37
6.6.1. Transfer in Channel Command - TIC	6-38
6.6.2. Store Subchannel Status Command - SST	6-39
<b>6.7. DATA TRANSFER</b>	6-39
6.7.1. Format Flags (E, A, B, and C)	6-39
6.7.2. Skip Data - SK	6-40
6.7.3. Data Address Decrement - DAD	6-40
6.7.4. Data Address Lock - DAL	6-40



<b>6.8. CHAINING OPERATIONS</b>	<b>6-40</b>
6.8.1. Data Chaining	6-41
6.8.2. Command Chaining	6-41
6.8.3. EI Chaining (ESI Word Interface Only)	6-42
6.8.4. Truncated Search	6-43
6.8.5. Truncated Search Restrictions	6-44
<b>6.9. INTERRUPT GENERATION FLAGS</b>	<b>6-44</b>
6.9.1. Program Controlled Interruption - PCI	6-44
6.9.2. Monitor - MON (Word Channel Only)	6-45
<b>6.10. STATUS</b>	<b>6-45</b>
<b>6.11. INSTRUCTION STATUS</b>	<b>6-49</b>
<b>6.12. STATUS TABLE</b>	<b>6-50</b>
<b>6.13. STORE SUBCHANNEL STATUS - SST</b>	<b>6-52</b>
<b>6.14. SUBCHANNEL STATUS</b>	<b>6-53</b>
6.14.1. SIOF Device Check (Bit 52) (Byte or Block Multiplexer Channel Only)	6-53
6.14.2. Interface Control Check (Bit 53)	6-53
6.14.3. Channel Control Check (Bit 54)	6-53
6.14.4. Channel Data Check (Bit 55)	6-53
6.14.5. Not Used (Bit 56)	6-54
6.14.6. Program Check (Bit 57)	6-54
6.14.7. Monitor (Bit 58) (Word Channel Only)	6-54
6.14.8. Incorrect Length (Bit 58) (Byte and Block Multiplexer Channels Only)	6-55
6.14.9. Program Controlled Interrupt (Bit 59)	6-55
<b>6.15. DEVICE STATUS</b>	<b>6-55</b>
<b>6.16. DATA CHAINING PRECAUTIONS</b>	<b>6-56</b>
<b>6.17. SUBCHANNEL EXPANSION FEATURE AND CHANNEL BASE REGISTER</b>	<b>6-63</b>
<b>6.18. MASK REGISTER</b>	<b>6-63</b>
<b>6.19. INITIAL LOAD</b>	<b>6-65</b>
<b>6.20. BACK-TO-BACK OPERATION (Word Channel Only)</b>	<b>6-65</b>
<b>6.21. PRIORITIES</b>	<b>6-66</b>
<b>6.22. BASIC PROGRAMMING PROCEDURE</b>	<b>6-66</b>
<b>6.23. PROGRAMMING EXAMPLES</b>	<b>6-67</b>
<b>7. Interrupts</b>	<b>7-1</b>
7.1. INTRODUCTION	7-1

<b>7.2. INTERRUPT SEQUENCE</b>	7-3
7.2.1. Program Status	7-3
7.2.2. Addressing Status	7-4
7.2.3. Interrupt Status	7-4
<b>7.3. INTERRUPT TYPES</b>	7-5
7.3.1. Program Exception Interrupts	7-5
7.3.2. Arithmetic Exception Interrupts	7-6
7.3.3. Program-Initiated Interrupts	7-7
7.3.4. Interprocessor Interrupt	7-8
7.3.5. Clock Interrupts	7-9
7.3.6. Storage Check Interrupts	7-9
7.3.6.1. Immediate Storage Checks	7-9
7.3.6.2. Delayed Storage Check Interrupts	7-10
7.3.6.2.1. SIU/MSU Errors and Internal SIU Errors	7-10
7.3.6.2.2. Storage Check Interrupt Status	7-11
7.3.7. Power Check Interrupt	7-14
7.3.8. Byte status Code	7-14
7.3.9. Multi-Processor-Interrupt Synchronization	7-18
<b>7.4. INPUT/OUTPUT INTERRUPTS</b>	7-18
7.4.1. Machine Check Interrupts	7-18
7.4.2. Normal Interrupts	7-19
7.4.3. Tabled Interrupts	7-22
<b>8. Executive Control</b>	8-1
<b>8.1. GENERAL</b>	8-1
<b>8.2. PROCESSOR STATE</b>	8-1
8.2.1. Designator Register	8-1
<b>8.3. INTRODUCTION TO ADDRESSING</b>	8-7
8.3.1. Main Storage Organization	8-7
8.3.2. Program Segmentation	8-7
8.3.3. General Theory of 1100/80 Addressing	8-7
8.3.4. Bank Descriptor	8-8
8.3.5. Limits	8-8
8.3.6. Control Information	8-8
8.3.7. Bank Descriptor Registers	8-8
8.3.8. Address Generation	8-10
8.3.9. P-Capturing Instructions	8-12
<b>Appendix A. Glossary</b>	A-1
<b>Appendix B. Summary Of Word Formats</b>	B-1
<b>Appendix C. User Instruction Repertoire</b>	C-1
<b>Appendix D. Character Codes</b>	D-1
<b>User Comment Sheet</b>	

## FIGURES

Figure 1-1. SPERRY UNIVAC 1100/81 System Minimum Configuration	1-2
Figure 1-2. SPERRY UNIVAC 1100/82 System Expanded Configuration	1-3
Figure 3-1. Main Storage Mapping of 262K Words	3-3
Figure 3-2. Main Storage Unit Address Assignment	3-4
Figure 3-3. Requester Absolute Address Format	3-9
Figure 4-1. Data Transfers From Storage	4-12
Figure 4-2. Data Transfers to Storage	4-13
Figure 4-3. J-Register Format for Character Addressing Mode	4-15
Figure 4-4. Byte Selected for Valid Combinations of BL and Ob Field Values	4-16
Figure 5-1. J-Register Format	5-51
Figure 6-1. 1100/80 Input/Output Unit	6-2
Figure 6-2. Byte or Block Multiplexer Channel Compared to 1100/80 Word Channel	6-4
Figure 6-3. Block Multiplexer Channel	6-68
Figure 6-4. Word Channel ISI Interface Example CCW List	6-70
Figure 7-1. Format of Guard Mode Interrupt Status	7-6
Figure 7-2. Format of Addressing Exception Interrupt Status	7-7
Figure 7-3. Format of Breakpoint Interrupt Status	7-8
Figure 7-4. Format of Interprocessor Interrupt Status	7-8
Figure 7-5. Format of Immediate Storage Check Interrupt Status	7-10
Figure 7-6. Storage Check Interrupt Status Word	7-13
Figure 7-7. Power Check Interrupt Status	7-14
Figure 8-1. Basic Designator Register States	8-6
Figure 8-2. Bank Descriptor and BDT Pointer Formats	8-9
Figure 8-3. Base Value Selection	8-11

## TABLES

Table 1-1. Fully Supported Configurations	1-4
Table 3-1. MSR Values vs. Module Identification	3-5
Table 3-2. Fixed Address Assignment 0200-0237	3-6
Table 3-3. Fixed Address Assignments 0240-0277	3-7
Table 3-4. Words/Blocks Per Storage Set as a Function of Main Storage/Buffer Capacity	3-8
Table 3-5. MSR Selection	3-10
Table 3-6. GRS Register Assignments 0 Through 63	3-11
Table 3-7. GRS Register Assignments 64 Through 127	3-12
Table 4-1. Instructions that Condition the Carry and Overflow Designators	4-3
Table 4-2. Single-Precision Floating-Point Characteristic Values and Exponent Values	4-5
Table 4-3. Double-Precision Floating-Point Characteristic Values and Exponent Values	4-5
Table 4-4. Explanation of J-Register Fields for Character Addressing Mode	4-15
Table 4-5. Output Ob Values Produced When BL = 0	4-17
Table 4-6. Output Ob Value Produced When BL = 1	4-18
Table 4-7. Output Ob Values Produced When BL = 2	4-18
Table 4-8. Output Ob Values Produced When BL = 3	4-19
Table 4-9. Summary of Use of i-Field	4-23
Table 5-1. Truth Table for Logical OR, XOR, and AND	5-45
Table 5-2. J-Register Increment Field Values	5-53
Table 5-3. Byte Status Word	5-54
Table 5-4. Byte String Sign Codes	5-55
Table 5-5. Function Byte Interpretation	5-58
Table 5-6. Subfunction Byte Interpretation	5-60
Table 5-7. Summary of Staging Register and J-Register Fields	5-62

Table 6-1. Device Addressing	6-7
Table 6-2. Channel, Subchannel, and Device States	6-8
Table 6-3. I/O System Composite State vs Condition Codes	6-11
Table 6-4. I/O Instruction Condition Codes for Byte or Block Channel	6-12
Table 6-5. I/O Instruction Condition Codes for Word Channels	6-14
Table 6-6. MSU Data Format - 36-Bit Format, Forward Operation	6-32
Table 6-7. MSU Data Format - 36-Bit Format, Backward Operation	6-33
Table 6-8. Format Flags vs Type of Channel	6-34
Table 6-9. CCW Flags vs Termination Conditions on Byte or Block Multiplexer Channel	6-35
Table 6-10. CCW Flags vs Termination Conditions on Word Channel	6-37
Table 6-11. CCW Command Code	6-38
Table 6-12. IOU Status	6-47
Table 6-13. IOU Fixed Addresses	6-49
Table 6-14. Byte Data Packing on Abnormal Boundaries	6-58
Table 6-14. Byte Data Packing on Abnormal Boundaries (continued)	6-59
Table 6-14. Byte Data Packing on Abnormal Boundaries (continued)	6-60
Table 6-14. Byte Data Packing on Abnormal Boundaries (continued)	6-61
Table 6-14. Byte Data Packing on Abnormal Boundaries (continued)	6-62
Table 6-15. Scratch Pad Formats for Subchannel Expansion Feature	6-64
Table 6-16. Interrupt Mask Register	6-65
Table 7-1. Interrupt Priority	7-2
Table 7-2. Byte Status Code Definition	7-15
Table 7-3. General Input Format for Byte-to-Floating Instructions	7-16
Table C-1. Mnemonic/Function Code Cross-Reference	C-1
Table C-2. Instruction Repertoire	C-4
Table D-1. Fieldata To ASCII Code Conversion	D-1

## 1. Introduction

### 1.1. GENERAL

This manual provides information on the central processor unit (CPU), main storage unit (MSU), buffer storage (SIU), and input/output unit (IOU) of the SPERRY UNIVAC 1100/80 Systems.

The SPERRY UNIVAC 1100/80 Systems are high-performance, software compatible, extensions to the proven SPERRY UNIVAC 1100 Series Systems. Designed to enhance the efficiency of the SPERRY UNIVAC 1100 Series, the 1100/80 Systems offer dependable and highly effective processing in real-time, demand, and batch modes and excel in multiprocessing applications.

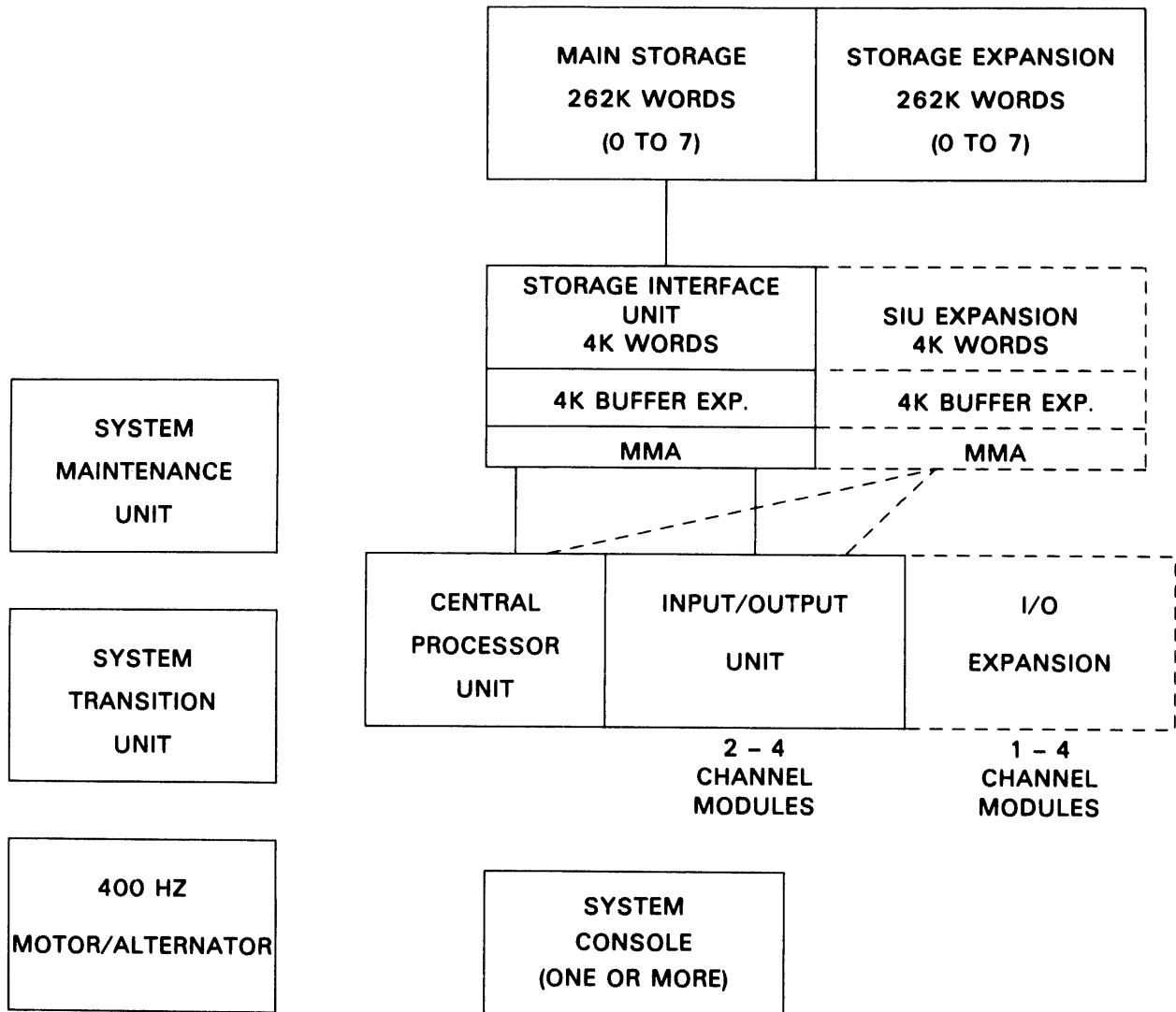
Although the 1100 Series Systems may differ in hardware design, software compatibility is maintained. All components of the 1100/80 Systems (processing units, input/output units, storage units, and peripherals) are controlled by the SPERRY UNIVAC 1100 Series Operating System. Industry standard language processors and application software are provided. The flexible design of the 1100/80 Systems allows the user to select a system to best meet his individual requirements.

### 1.2. 1100/80 SYSTEM CONFIGURATIONS

The basic 1100/81 Processing System (1x1 configuration) consists of two functionally and physically independent units: one CPU and one IOU. The processor organization is intrinsically that of a multitask processor and is designed for operation in a multiprogramming environment. The basic system may be expanded by adding a CPU and/or an IOU up to a total of two CPUs (1100/82 System) and two IOUs (2x2). The basic 1x1 configuration is shown in Figure 1-1. A 2x2 configuration is shown in Figure 1-2. Table 1-1 lists all fully supported configurations.

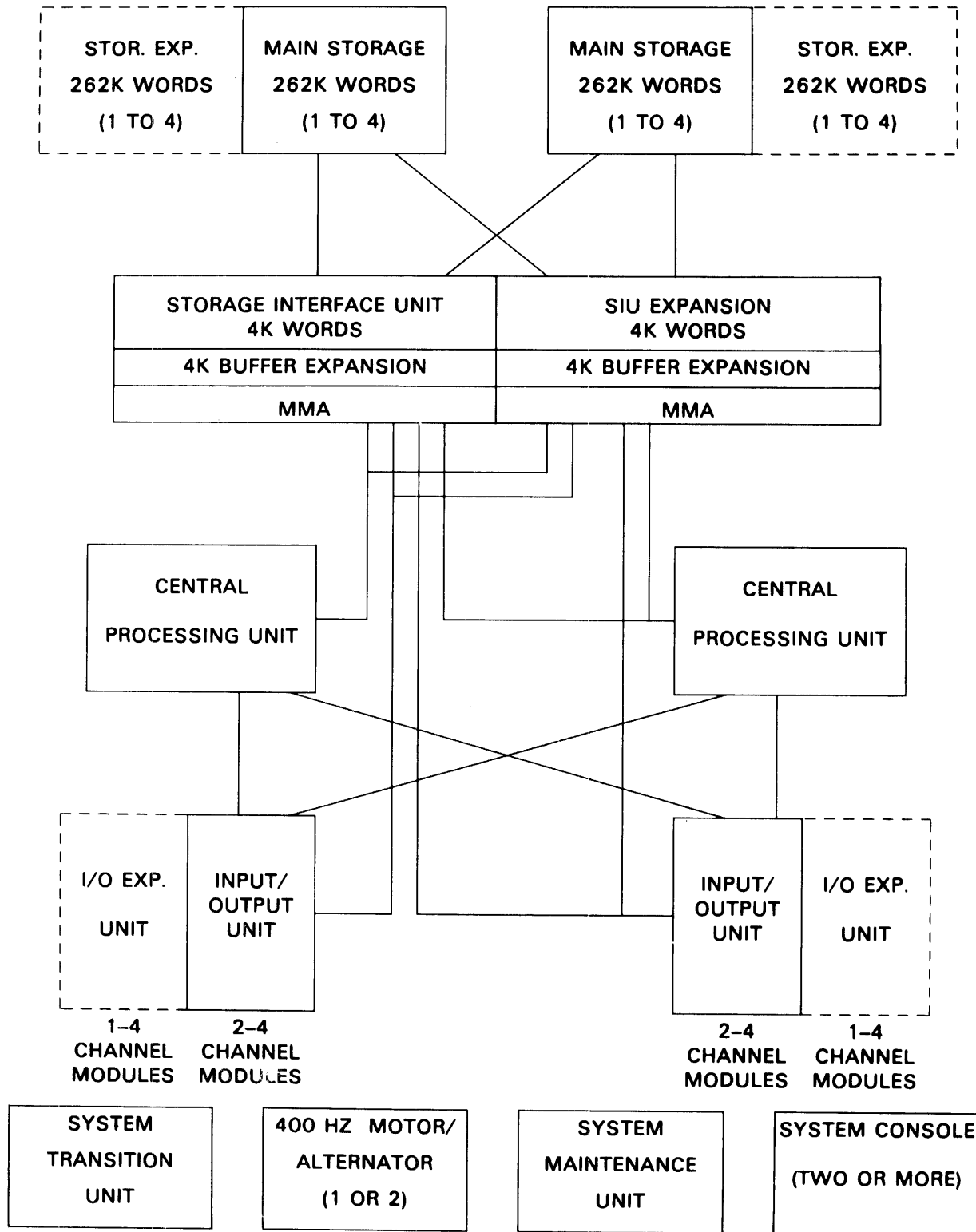
#### 1.2.1. Central Processor Unit

The processing unit is the central element of a large-scale system that is capable of serving both business and scientific applications in batch, demand, and real-time environments. The processing unit provides compatibility with prior 1100 Series Systems at the user object level, depending on internal code selections, peripheral configurations, and software implementation of hardware enhancements and user interfaces.



NOTE: Main storage must contain a minimum of 524K words.

Figure 1-1. SPERRY UNIVAC 1100/81 System Minimum Configuration



NOTE: Main storage must contain a minimum of 1048K words.

Figure 1-2. SPERRY UNIVAC 1100/82 System Expanded Configuration

Table 1-1. Fully Supported Configurations

Units	Configuration			
	1100/81		1100/82	
	1X1	1X2	2X1	2X2
CPU	1	1	2	2
IOU	1	2	1	2
Main Storage (words)	524K - 4194K	524K - 4194K	1048K - 4194K	1048K - 4194K
Storage Interface Units (words)	8K <sup>(1)</sup> - 16K	8K <sup>(1)</sup> - 16K	16K	16K
System Console	1-N*	1-N*	2-N*	2-N*
System Transition Unit	1	1	1	1
System Maintenance Unit	1	1	1	1
Motor/Alternator	1	1-2	1-2	1-2

\* N equals any number required, but it is limited by I/O.

(1) May be one 8K buffer or two 4K buffers.

The basic processing unit consists of the following components:

- A control and arithmetic section that includes fixed-point arithmetic; logical data manipulation;
- Instruction, interrupt, and arithmetic control and control storage;
- Maintenance section which acts as a device and a control during off-line maintenance procedures initiated by the maintenance processor; and
- Interfaces for two input/output units, one storage interface unit, one system maintenance unit, one system transition unit, and the system interrupt network.

The processing unit also has the following features:

- Floating-point instruction control and arithmetic.
- Byte-oriented instruction control and arithmetic.

The processing unit has the following general characteristics:

- A complete set of arithmetic, logical, manipulative, data transfer, and sequence control instructions.



- A comprehensive relative addressing mechanism providing program segmentation and storage protection.
- An absolute addressing range of sixteen million 36-bit words.
- A basic instruction fetch period of 200 nanoseconds.
- A general purpose microprogrammed arithmetic section.

### 1.2.2. Main Storage

The 1100/80 main storage system consists of large capacity storage units plus high speed storage buffers to achieve increased performance from the lower speed main storage. Operation of the buffer is transparent to software, to the extent that software organization affects the miss rate, or percentage of instructions, or operands not located in the buffer when requested.

The basic main storage unit consists of 262K words located in a single cabinet. This can be expanded up to eight storage cabinets, minimum memory is 524K words.

The basic storage interface unit contains 8K words of buffer storage. In addition, a second 4K-word buffer may be added which may be expanded to 8K words giving a maximum buffer size in a system of 16K words.

### 1.2.3. Input/Output Unit (IOU)

The basic 1100/81 System configuration includes one input/output unit (IOU). The IOU controls all transfers of data between the peripheral devices and main storage. Transfers are initiated by a CPU under program control. The IOU includes independent data transfer paths to the CPU and to main storage. The primary mode of I/O transmission is through byte channels with word channels available as an option.

The IOU consists of two sections: a control section and a section containing from two to four input/output channel modules. An IOU expansion allows up to four additional channel modules to be added to the IOU. The word I/O channel option provides four word I/O channel modules and occupies one byte channel module position. A second IOU with identical expansion capabilities can be added to a system.

The control section includes all logic associated with the transfer of function, data, and status words between main storage and the subsystems. It also services I/O requests from either one or both of the CPUs (in a multiprocessor system) and routes interrupts to one of the two processing units. Interrupt routing may be specified by program.

The IOU capabilities are given below.

- Primary mode of I/O transmission through byte channels; word channels are optional.
- Channel transfer rates of:
  - $3.0 \times 10^6$  bytes per second (maximum) on a block multiplexer channel module;
  - $200 \times 10^3$  bytes per second (maximum) on a byte multiplexer channel module;
  - $500 \times 10^3$  words per second aggregate for a word channel module.

- Externally specified index (ESI) and internally specified index (ISI) transfer modes on the word channels.
- Channel buffering
- Interrupt tabling
- Parity generation/checking capability on all ISI channels.

#### 1.2.4. System Console

The system console provides the means for communication with the Executive System. The basic console consists of the following major components:

- The CRT/keyboard consists of a UNISCOPE 100 Display Terminal. The display format is 16 lines with 64 characters per line. The seven-bit ASCII character set, consisting of 95 characters plus the space, is used. The keyboard provides all of the operator controls required for generating data and initiating transfers.
- The incremental printer operates at 30 characters per second and provides a hard copy of console messages. (Five additional incremental printers may be connected to a console.)
- Maintenance interface for remote console operation by means of the system maintenance unit and the Total Remote Assistance Center (TRACE) computer system.
- The fault indicator, located on the incremental printer, provides the operator with a visual indication of a fault condition in a major system component. The actual component and nature of the fault may then be determined from indicators on the operator/maintenance panel on the system transition unit.
- A standard byte multiplexer channel interface.

#### 1.2.5. System Transition Unit (STU)

The STU contains the controls and indicators required for control and assignment of the system units. The functions controlled by the STU are:

- **Power sequencing**  
Controls turn-on/turn-off of main system power and sequencing of ac power in the CPUs, IOUs, and main storage (including buffer storage sections).
- **Partitioning**  
This function provides the ability to assign the individual units to either one of two independent applications or to isolate it from either application for off-line concurrent maintenance. Included in this function is the control for the automatic expansion or compression of the main storage address range for both applications. This operation provides contiguous main storage ranges for either or both applications for any combination of main storage unit assignments.

This function also indicates the operational status of each central complex unit. The state of these status conditions are available to system software for configuration control.

The ability to partition peripheral subsystems is provided by controls on the individual subsystems and, optionally, for byte peripheral subsystems, by software command.

■ Initial Load

This function provides the ability to set MSR (module select register) values, select initial load paths and initiate the initial load operation for either one of two applications.

■ Automatic Recovery

This function provides the system, specified in Application O, with an automatic system recovery capability. When auto-recovery is enabled and the system software does not reset the auto-recovery timer within the preset time interval, the system is cleared, reloaded, and reinitiated. The system provides two recovery paths. The alternate recovery path is automatically initiated when an attempted recovery fails. The function provides for software resetting of the auto-recovery timer and selection of the auto-recovery path to be used by the next auto-recovery attempt.

■ Processor and Input/Output Unit Controls

This function provides the controls and indicators required for manual control of the processors and input/output units.

### 1.2.6. System Maintenance Unit

The system maintenance unit provides for diagnostic checkout and fault isolation by the automatic comparison of maintenance indicators against known correct data and the creation of dumps. The system maintenance unit includes a maintenance processor, communications capability, UNISCOPE 100 Display Terminal, card tester and peripherals.

### 1.2.7. Auxiliary Storage and Peripheral Subsystems

The 1100/80 Systems offer a full range of auxiliary storage and peripheral subsystems to provide the capability to satisfy many requirements. The following list of peripheral equipment is the minimum available with the 1100/80 System. This minimum has been established to ensure an adequate complement for customer engineering and software support.

	Minimum Complement	Alternate
1.	One 0716 Card Reader and one 0776 High Speed Printer Subsystem	
2.	8430/8433/8434 Disk Subsystem with one 5046 Control Unit and two 8430 or two 8433 or two 8434 Disk Storage Units	One disk subsystem as follows: - 5024 Control Unit with two 8425 Disk Storage Units
3.	UNISERVO Magnetic Tape Subsystem with 5042 Control Unit and four UNISERVO 30 Tape Units	UNISERVO Magnetic Tape Subsystem with 5017 Control Unit and four UNISERVO 12/14/16/20 Tape Units



## 2. Processing Unit

### 2.1. GENERAL

The 1100/80 Central Processing Unit (CPU) contains a control section, an arithmetic section, a maintenance section, a General Register Stack, and interfaces through which it is connected to other equipment. The IOU controls all data transfers between peripheral devices and storage. Transfers are initiated by a CPU under program control.

### 2.2. CONTROL SECTION

The control section of the CAU interprets instructions and directs all processor operations except certain I/O operations. It is discussed briefly below and in more detail in 4.2.

#### 2.2.1. Control Section Operation

The program instruction words are sequentially loaded into the control section. Each instruction word is interpreted by the control section which generates the signals necessary to perform the instruction. The instruction words are located in main storage and the data words (operands) are located either in main storage or in the addressable control registers which are part of the control section. The control section includes an address formation segment which generates the absolute main storage addresses to obtain the instruction words.

The instruction word is divided into fields. These fields specify to the control section the function to be performed, which portion of the operand is to be used, a control register, indexing, index register modification, indirect addressing, and an operand address.

#### 2.2.2. Instruction Repertoire

The instruction repertoire includes fixed-point and floating-point arithmetic, logical functions, byte operations, block transfers, comparisons, tests, I/O control, and special purpose instructions. There are over 200 basic instructions in the repertoire. Partial word data transfers and repetitive operations are included in the instruction repertoire. Indexing capability is provided with all instructions. Indirect addressing capability is also provided and is usable to any level with full indexing capability at each level.

Instructions such as data transfers, single-precision fixed-point adds, and certain logical functions, require less than 250 nanoseconds for complete execution. Indexing (18-bit) does not add to the execution time of an instruction. Details of the instruction repertoire are found in Section 5.

### 2.2.3. Control Registers

The 128 addressable control registers in the general register stack (GRS) of the control section are integrated-circuit registers. These control registers are addressed either explicitly or implicitly by the instructions. They fall into four categories: index registers, arithmetic registers, special registers, and unassigned registers.

The control registers are discussed in detail in Section 3.

### 2.2.4. Data Shift/Complement/Store Operation

The CPU includes circuitry which permits the various store instructions to bypass the arithmetic section. This circuitry includes the shifting capability needed for storing partial words in main storage, the sign testing capability needed for the Store Magnitude A instruction, and the complementing capability needed for the Store Negative A and Store Magnitude A instructions.

## 2.3. ARITHMETIC SECTION

All arithmetic computation is microprogram controlled and is performed using the nonaddressable registers of the arithmetic section. These arithmetic processes can be performed in either fixed-point or floating-point mode. Fixed-point arithmetic instructions provide for single-precision, double-precision, half-word, and third-word addition and subtraction, and for fraction and integer multiplication and division. Floating-point instructions provide for both single-precision and double-precision operation. The arithmetic section also performs certain logical operations such as shifting and comparisons. The instruction word may be used to specify the transfer of any chosen portion of a word (half, third, quarter, or sixth) to the arithmetic section. The ability to transfer only the selected portion of a word minimizes the number of masking and shifting operations required.

A shift matrix in the arithmetic section permits the completion of an entire single word shift operation in one main storage cycle time. By use of the matrix, the shift operation can shift a single or double word operand in either direction up to 72 bit positions.

Details on the operation of the arithmetic section are found in 4.1.

## 2.4. MAINTENANCE SECTION

The maintenance section performs all diagnostic tests using its own repertoire of commands. It operates only when the processor is in maintenance mode. In this mode the processing system can be operating either online or offline. When online, the processing system and the maintenance system operate concurrently. In this case the maintenance system is connected to and operating on the byte bus and the processing system operates normally except that the processing operation is suspended whenever the maintenance system needs to use the processor data and control paths for executing a maintenance function.

## 2.5. INPUT/OUTPUT UNIT (IOU)

The IOU is a separate functional entity. I/O activity is initiated when the interpretation of certain instructions by the CPU causes signals to be sent to the IOU. Once an I/O operation is initiated, the IOU and the subsystem control the input and output transfers. The IOU operates with a wide variety of peripheral devices, and it requires minimal attention from the CPU.

Once an I/O operation is initiated by the program, I/O activity is independent of program control. The I/O data flows between main storage and the peripheral subsystem through an I/O channel. Each I/O channel consists of 36 input data lines, 2 input parity lines, 36 output data lines, 2 output parity lines, and various control signal lines. All data word bits are transmitted in parallel to or from the subsystem.

The I/O unit has four interfaces: a storage interface, a processor interface, a control unit-peripheral interface, and a system transition unit interface.

Details of the IOU are presented in Section 6.





## 3. Storage

### 3.1. GENERAL

The storage system comprises up to 4,194,304 words of main storage, up to 16,384 words of high speed buffer storage and 128 words of control registers.

The main storage units (MSUs) provide storage for the instruction and data words. The storage interface unit (SIU) provides high speed buffer storage between the storage units and the processor and between the storage units and the input-output units (IOUs). The 128 addressable control registers in the control section of each CPU provide fast access storage for data and control words.

### 3.2. MAIN STORAGE

A data or instruction word consists of 36 information bits and two parity bits. The two parity bits provide hardware parity checking on each 18-bit segment of the word transferred over the MSU-SIU interface.

Data to or from main storage is transferred in block increments of eight contiguous words. The eight words, comprising four double words (72 data, 4 parity bits) are transferred in four sequential operations. Each double word is written into or read from storage as an 80-bit word (72 data, 8 ECC). The eight error correction code (ECC) bits are generated from the write data. If a single-bit error is detected during a read, the error correcting code is used to correct the data. If multiple errors are detected in the stored data the processor is notified with an Interrupt signal.

A main storage cabinet contains 262K or 524K words. A maximum of eight cabinets may be used in a system. Figures 1-1 and 1-2 show the minimum and expanded system configurations.

#### 3.2.1. Main Storage Addressing

Main storage addressing is continuous from the lowest order address to the highest. Figure 3-1 shows the MSU mapping of 262K words into 8-word block increments. Each block contains eight contiguous words and the blocks are sequential relative to the set addresses. Also shown is the relationship of the set address portion of an absolute address to the blocks stored in an MSU when the buffer capacity is 4096 words. That is, in each 262K module of main storage there are 128 8-word blocks for each set address or, stated differently, 1/128 of the total storage capacity (regardless of size) is assigned to each set address. For an expanded buffer of 8192 words 1/256 of the total storage capacity is assigned to each set address. Set addresses associated with a buffer storage capacity of 4096 words are on a modulo 1024 and for a 8192-word buffer capacity the set addresses are on a modulo 2048.

When a requested absolute address is not in buffer storage the SIU initiates a request to main storage to read out the 8-word block containing the needed address and data. The request is initiated by bussing the 18 bits of the absolute address (shown in the following format) to all the MSUs in the configuration. The identified MSU executes the read cycle.

### 3.2.2. MSU Address Assignments

A system can have one to eight MSUs with each MSU having 262K or 524K words each. A maximum configuration has 4,194,304 words in eight cabinets. The system is capable of addressing 16,777,216 words which is divided into upper and lower address ranges of 8,388,608 words each.

The addressing for the first MSU (0) installed always begins at the mid-address  $2^{23} - 1$  (8,388,607) minus the size of that main storage unit (262K or 524K). The second MSU (1) begins its addressing at  $2^{23}$  if the SIU upper address is present. More MSUs can be added on either side of the mid-address point up to eight MSUs. MSUs added to either address range are assigned to the configuration with their numbers in an increasing order going away from the midpoint address. (See Figure 3-2.) This method of MSU assignment allows using a system of as little as one MSU and expanding it to eight MSUs with no change in the addressing logic. Table 3-1 shows the absolute addresses for MSUs assigned to specific positions in a maximum storage configuration.

This redundancy in the SIU allows concurrent addressing of the upper and lower address ranges and the even and odd addresses within each address range. It is possible to service four requests concurrently.

Note that the minimum SIU configuration is 8K words which can be one 8K buffer or two 4K buffers, with one 524K MSU or two 262K units. However, the system will operate in a degraded mode with one 4K SIU and one 262K MSU.

Figure 3-1. Main Storage Mapping of 262K Words

Sets		MSU Words		Sets		MSU Words	
8K	4K			8K	4K		
0	0	0	- 7	254	126	6147	- 6135
1	1	8	- 15	255	127	6136	- 6143
	2	16	- 23	0	0	6144	- 6151
	(1)				1	6152	- 6159
	3	24	- 31				
	4	32	- 39		(7)		
					127	7160	- 7167
	126	1008	- 1015	(4)	0	7168	- 7175
127	127	1016	- 1023		1	7176	- 7183
(1)							
128	0	1024	- 1031		(8)		
129	1	1032	- 1039				
	2	1040	- 1047	255	127	8184	- 8191
				0	0	8192	- 8199
	(2)			1	1	8200	- 8207
254	126	2032	- 2039		(9)		
255	127	2040	- 2047	2	2	8208	- 8215
0	0	2048	- 2055	3	3	8216	- 8223
	1	2056	- 2063				
	(3)				127	9208	- 9215
127	127	3064	- 3071		(5)		
(2)					0	9216	- 9223
128	0	3072	- 3079		(10)		
	1	3080	- 3087	255	127	10240	- 10247
	(4)						
254	126	4080	- 4087	128	0	261120	- 161127
255	127	4088	- 4095	(64)			
0	0	4096	- 4103				
1	1	4104	- 4111		(128)		
2	2	4112	- 4119				
	(5)			255	127	262136	- 262143
3	3	4120	- 4127				
4	4	4128	- 4135				
126	126	5104	- 5111	NOTE: The numbers in parentheses ( ) show the association of the number of buffer-set increments to each 262K of storage.			
127	127	5112	- 5119				
(3)							
128	0	5120	- 5127				
129	1	5128	- 5135				
	2	5136	- 5143				
	(6)						

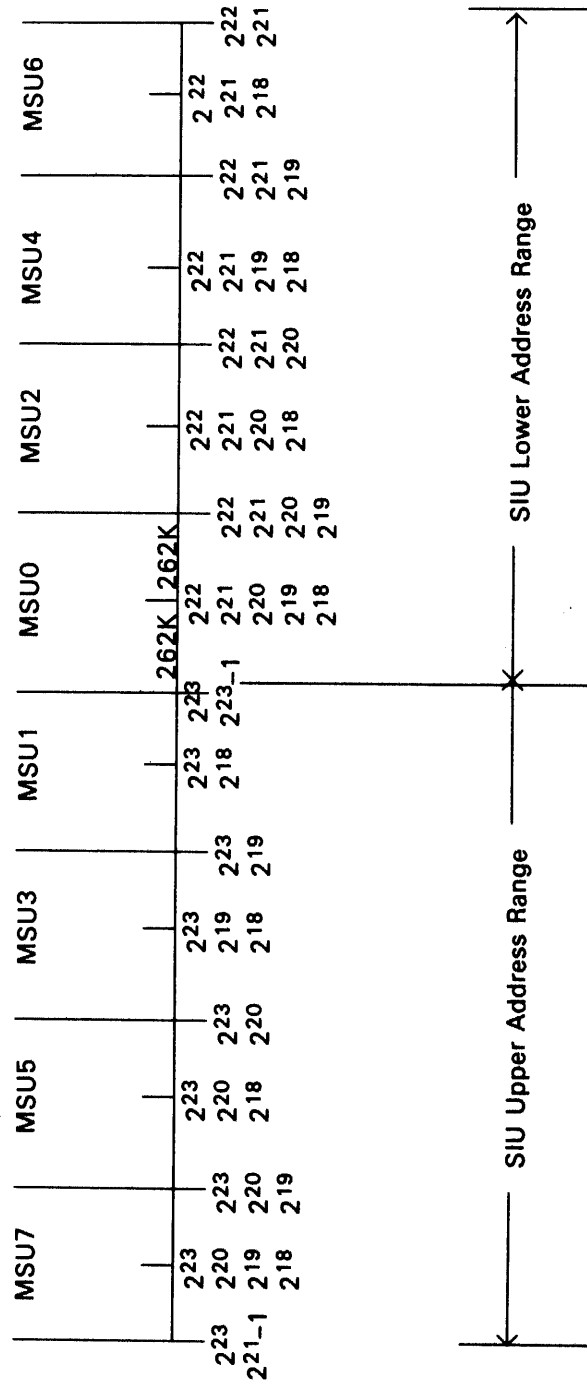


Figure 3-2. Main Storage Unit Address Assignment

Table 3-1. MSR Values vs. Module Identification

Lower Address Range		
MSR Value (Octal)	Address Ranges (Octal)	Physical Module
140	30 000 000 - 30 777 777	MSU6
144	31 000 000 - 31 777 777	
150	32 000 000 - 32 777 777	MSU4
154	33 000 000 - 33 777 777	
160	34 000 000 - 34 777 777	MSU2
164	35 000 000 - 35 777 777	
170	36 000 000 - 36 777 777	MSU0
174	37 000 000 - 37 777 777	

Upper Address Range		
MSR Value (Octal)	Address Ranges (Octal)	Physical Module
200	40 000 000 - 40 777 777	MSU1
204	41 000 000 - 41 777 777	
210	42 000 000 - 42 777 777	MSU3
214	43 000 000 - 43 777 777	
220	44 000 000 - 44 777 777	MSU5
224	45 000 000 - 45 777 777	
230	46 000 000 - 46 777 777	MSU7
234	47 000 000 - 47 777 777	

### 3.2.3. Fixed Address Assignments

The interrupt subroutine entrances and certain status words are assigned fixed locations in main storage as shown in Tables 3-2 and 3-3. The listed addresses are relative to the contents of the 7-bit module select register (MSR) and the position of the SIU Upper/Lower switch. MSR may be manually loaded by pressing the desired combination of the seven MSR switches and the SIU Upper/Lower switch on the system transition unit partitioning panel. When an initial load operation

is performed, the value in the MSR identifies the main storage area in which the incoming data is to be stored. During an ESI-I/O operation the value in the MSR identifies the high order bits of the address of the main storage locations from which the ESI access control words and chain pointer words are obtained.

*Table 3-2. Fixed Address Assignment 0200-0237*

Octal	Decimal	Assignment
200	128	Reserved for Hardware Default
201	129	Unassigned
202	130	Unassigned
203	131	Unassigned
204	132	I/O Normal Status Interrupt
205	133	I/O Tabled Status Interrupt
206	134	I/O Machine Check Interrupt
207	135	Unassigned
210	136	Quantum Timer Interrupt
211	137	Real Time Clock Interrupt
212	138	
213	139	
214	140	
215	141	
216	142	Dayclock Value
217	143	Dayclock Interrupt
220	144	Immediate Storage Check Interrupt
221	145	Invalid Instruction
222	146	Executive Request Interrupt
223	147	Guard Mode Interrupt
224	148	Test and Set Interrupt
225	149	Characteristic Underflow Interrupt
226	150	Characteristic Overflow Interrupt
227	151	Divide Check Interrupt
230	152	Addressing Exception Interrupt
231	153	Breakpoint Interrupt
232	154	Interprocessor Interrupt
233	155	Power Check Interrupt
234	156	Delayed Storage Check Interrupt
235	157	Jump History Stack Interrupt
236	158	Emulation Interrupt
237	159	Unassigned

All fixed addresses are relative to the MSR.

Table 3-3. Fixed Address Assignments 0240-0277

Octal	Decimal	Assignment
240	160	Processor 0 Channel Address Word 0
241	161	Processor 0 Channel Address Word 1
242	162	Unassigned
243	163	Unassigned
244	164	Processor 1 Channel Address Word 0
245	165	Processor 1 Channel Address Word 1
246	166	Unassigned
247	167	Unassigned
250	168	Processor 0 Interrupt Address Word
251	169	Unassigned
252	170	Unassigned
253	171	Unassigned
254	172	Unassigned
255	173	Unassigned
256	174	Unassigned
257	175	Unassigned
260	176	Processor 0 Interrupt Address Word
261	177	Processor 0 Channel Status Word 0
262	178	Processor 0 Channel Status Word 1
263	179	Processor 0 Channel Status Word 2
264	180	Processor 1 Interrupt Address Word
265	181	Processor 1 Channel Status Word 0
266	182	Processor 1 Channel Status Word 1
267	183	Processor 1 Channel Status Word 2

All fixed addresses are relative to the MSR. Addresses 270 through 277 are unassigned.

### 3.3. BUFFER STORAGE

Buffer storage is in the storage interface unit (SIU) and is transparent to users, processors, IOUs, and the operating system. The only access to main storage is through the SIU with absolute addresses. The basic buffer capacity is 4096 word (36 bit + parity bit words) and is expandable in 4096 word increments to 16,384 words.

The storage interface unit, comprises two identical and functionally independent halves, SIU upper and SIU lower. SIU upper provides access to MSUs assigned with absolute addresses  $2^{23}$  (8,388,608) and up; SIU lower provides access to MSUs assigned with absolute addresses  $2^{23} - 1$  (8,388,607) and down. Each SIU half contains a buffer of 4096 words, expandable to 8192 words maximum and a table of the block addresses (called tag) of the words resident in the buffer.

#### 3.3.1. Set Associative Addressing

Both the buffer and the absolute address space in main storage are divided into sets. Each set in the buffer is associated with a corresponding set of absolute addresses in main storage. Each buffer set is functionally a 4-entry content-addressable memory, where each entry represents a block (8 words) of main storage in an associated absolute address space set. A block is eight contiguous words beginning at an address divisible by eight. Each buffer set contains four 8-word blocks (32 words) and its contents could be any four blocks from a like numbered storage set. For example, buffer set 1 could contain blocks 4104-4111, 8-15, 6152-6159, 2056-2063 from storage set 1.

Figure 3-1 shows the relationship of main storage blocks to the 4K and 8K buffer sets. For a 4K buffer 1/128 of all storage blocks are associated with a given buffer set. For an 8K buffer 1/256 of all storage blocks are associated with a given buffer set. For example, main storage blocks associated with the 4K and 8K buffer sets are on a modulo 1024 and 2048, respectively, as follows:

4K Buffer			8K Buffer		
Set 0	Set 1	Set 127	Set 0	Set 1	Set 255
0 - 7	8 - 15	1016-1023	0 - 7	8 - 15	2040-2047
1024-1031	1032-1039	2040-2047	2048-2055	2056-2063	4088-4095
2048-2055	2056-2063	3064-3071	4096-4103	4104-4111	6136-6143
3072-3079	3080-3087	4088-4095	6144-6151	6152-6159	8184-8191
etc.	etc.	etc.	etc.	etc.	etc.

Table 3-4 shows the relationship of the number of buffer sets to the number of words and 8-word blocks in a storage set for the 1100/80 main storage capacities available.

Table 3-4. Words/Blocks Per Storage Set as a Function of Main Storage/Buffer Capacity

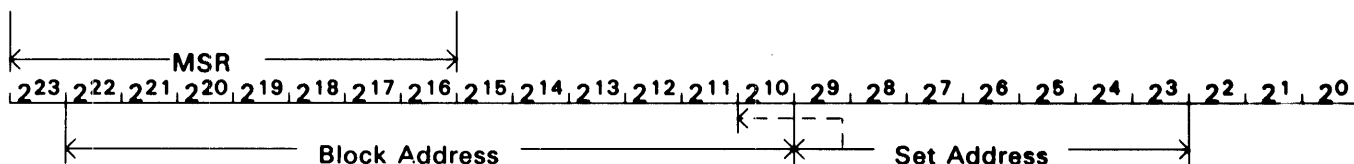
Main Storage Capacity (Words)	4K Buffer (128 Sets)		8K Buffer (256 Sets)	
	Words Per Storage Set	Blocks Per Storage Set	Words Per Storage Set	Blocks Per Storage Set
262144*	2048*	256*	1024*	128*
524288	4096	512	2048	256
786432	6144	768	3072	384
1048576	8192	1024	4096	512
1310720	10240	1280	5120	640
1572864	12288	1536	6144	768
1835008	14336	1792	7168	896
2097152	16384	2048	8192	1024
2359296	18432	2304	9216	1152
2621440	20480	2560	10240	1280
2883584	22528	2816	11264	1408
3145728	24576	3072	12288	1536
3407872	26624	3328	13312	1664
3670016	28672	3584	14336	1792
3932160	30720	3840	15350	1920
4194304	32768	4096	16384	2048

\* Degraded mode only



### 3.3.2. Address Interleave

When the SIU contains an upper and lower half then each half may be configured as an even and odd segment. When a requesters absolute address (contents of Figure 3-3 sans MSR) is sent to the SIU, bit 23 selects the upper or lower half and bit 0 selects the odd or even segment in that SIU half. The set address initiates the reading of four block addresses from the tag and the data from the buffer. A comparison is made between the requested block address and the four tag block addresses and if the requested block is resident, the data is sent to the requester; if not, bits 20 through 3 with MSR appended is sent to main storage. (See 3.2.1.) The MSR values are listed in Table 3-5.



- 2<sup>23</sup> = 0 Selects lower address range 2<sup>23</sup> - 1 and down (SIU Lower)
- = 1 Selects upper address range 2<sup>23</sup> and up (SIU Upper)
- 2<sup>22</sup> - 2<sup>10</sup> - Block address
- 2<sup>9</sup> - 2<sup>3</sup> - Set addresses - 0 - 127 (4096 word basic buffer)
- 2<sup>10</sup> - 2<sup>3</sup> - Set addresses - 0 - 255 (8192 word expanded buffer)
- 2<sup>2</sup> - 2<sup>1</sup> - Selects 1 of 4 odd or even words (00-11)
- 2<sup>0</sup> - 0 = Even word segment
- 1 = Odd word segment

**NOTE:**

MSR is appended by the SIU when an MSU request is made.

*Figure 3-3. Requester Absolute Address Format*

Table 3-5. MSR Selection

MSU Granularity (Millions <sub>10</sub> of Words)				Number Of Words (Millions <sub>10</sub> ) Available To SIU Half	SIU Upper Bits						SIU Lower Bits					
2	1	1/2	1/4		23	22	21	20	19	18	23	22	21	20	19	18
		X		1/4	1	0	0	0	0	0	0	1	1	1	1	1
	X		X	1/2	1	0	0	0	0	1	0	1	1	1	1	0
		X		3/4	1	0	0	0	1	0	0	1	1	1	0	1
	X	X	X	1	1	0	0	0	1	1	0	1	1	1	0	0
		X		1 1/4	1	0	0	1	0	0	0	1	1	0	1	1
	X		X	1 1/2	1	0	0	1	0	1	0	1	1	0	1	0
		X		1 3/4	1	0	0	1	1	0	0	1	1	0	0	1
X	X	X	X	2	1	0	0	1	1	1	0	1	1	0	0	0

## NOTE:

Bit 16 & 17 of MSR will always be zero in the normal (non-debug) case regardless of SIU half selection.

This optional redundancy in the SIU allows concurrent addressing of the upper and lower address ranges and the even and odd addresses within each address range. It is possible to service four requests concurrently.

### 3.4. CONTROL STORAGE

The control section of the CPU includes a general register stack (GRS) comprising 128 addressable control registers that can be independently referenced in parallel with main storage. Each control register stores a word consisting of 36 information bits. The control registers are addressable by the a-, and x-fields of the instruction word and by the value U developed in the index subsection of the CPU's control section. The details of control register addressing are explained in Section 4. Table 3-6 and 3-7 summarize the control register address assignments.

#### 3.4.1. Control Register Selection Designator

The 128 addressable control registers include one set of registers for use by the user program and another set for use by the Executive program. The control register selection designator (D6) in the designator register defines which set of registers is addressed by the a- and x-designators of an instruction word. When D6 = 0, the user program set of control registers is addressed; when D6 = 1, the Executive program set of control registers is addressed. The contents of D6 has no effect on the choice of a control register for any particular value of U.

#### 3.4.2. Control Register Address Assignments

Operand addresses  $0_8$  through  $177_8$  are assigned to the control registers. The following paragraphs define the various uses and related address assignments for the control registers.

### 3.4.2.1. Storage for MSR Value – Address 0143

During Initial Load of the system the value in the module select register (MSR) is loaded into GRS by the hardware. This one time load makes the MSR value available for referencing by software.

Table 3-6. GRS Register Assignments 0 Through 63

Octal	Decimal	Register Assignment
0000	0	Initial Load MSR Value → X3
0001	1	User X1
.	.	.
0011	9	User X9
0012	10	User X10
0013	11	User X11
0014	12	User X12/A0
0015	13	User X13/A1
0016	14	User X14/A2
0017	15	User X15/A3
0020	16	User A4
0021	17	User A5
.	.	.
0033	27	User A15
0034	28	Unassigned
.	.	.
0037	31	Unassigned
0040	32	Exec BDT Pointer
0041	33	Immed. Stor. Check Program Return Address
0042	34	Immed. Stor. Check Designator Register
0043	35	Normal Program Return Address
0044	36	Normal Designator Register
0045	37	User BDT Pointer
0046	38	E 0 0— 0  BDI0  E 2 0— 0  BDI2
0047	39	E 1 0— 0  BDI1  E 3 0— 0  BDI3
0050	40	Quantum Timer
0051	41	Guard Mode Program Return Address
0052	42	Guard Mode Designator Register
0053	43	Guard Mode Interrupt Status
0054	44	Immed. Stor. Check Status
0055	45	Normal Status
0056	46	Unassigned
0057	47	Unassigned
.	.	.
0067	55	.
0070	56	Jump History Stack
.	.	.
0077	63	Jump History Stack

\* Note that locations 060 through 067 are used as temporary working storage locations by the processor, and their contents are therefore unpredictable. Delay Storage Checks are classed as normal interrupts.

Table 3-7. GRS Register Assignments 64 Through 127

Octal	Decimal	Register Assignment
0100	64	Real Time Clock
0101	65	User R1/Repeat Count
0102	66	User R2/Mask Register
0103	67	User R3/Staging Register 1
0104	68	User R4/Staging Register 2
0105	69	User R5/Staging Register 3
0106	70	User R6/J0
0107	71	User R7/J1
0110	72	User R8/J2
0111	73	User R9/J3
0112	74	User R10
.	.	.
0117	79	User R15
0120	80	Exec R0
0121	81	Exec R1/Repeat Count
0122	82	Exec R2/Mask Register
0123	83	Exec R3/Staging Register 1
.	.	.
0137	95	Exec R15
0140	96	Unassigned
0141	97	Exec X1
.	.	.
0153	107	Exec X11
0154	108	Exec X12/A0
.	.	.
0157	111	Exec X15/A3
0160	112	Exec A4
.	.	.
0173	123	Exec A15
0174	124	Unassigned
.	.	.
0177	127	Unassigned

### 3.4.2.2. User Index (X) Register – Addresses 0001 – 0017

The index registers, referred to as X-registers provide the programmer with address modification capability (indexing).

An index register contains a modifier field (Xm), which is used to modify the operand address (indexing), and an increment field (Xi), which is used to modify the modifier field (automatic incrementation). If designator register bit 7 (D7) and the i-bit of an instruction are one, 24-bit index register mode is specified. In this mode, Xm is the lower 24 bits of the index register (bits 23-0), and Xi is the upper 12 bits of the index register (bits 35-24). In all other cases, 18-bit index register mode is selected. In this mode, Xm is the lower 18 bits of the index register (bits 17-0), and Xi is the upper 18 bits of the index register (bits 35-18).

### 3.4.2.3. User Accumulator (A) Registers – Addresses 0014 – 0033

The A-registers store arithmetic operands and results. The actual computation or logical function is performed in the arithmetic section and the results are stored in the A-register or registers specified by the instruction. Four of the A-registers (addresses 014<sub>8</sub> – 017<sub>8</sub>) overlap registers assigned as X-registers. This affords additional versatility in the use of A-registers and X-registers.

### 3.4.2.4. User Unassigned Registers – Addresses 0034 – 0037

Two of these unassigned registers (0034<sub>8</sub> and 0035<sub>8</sub>) serve as an extension of the set of user A-registers when D6 = 0 and an instruction which requires more than one user A-register is being performed. All four of these unassigned registers can serve as general purpose registers.

### 3.4.2.5. EXEC Bank Descriptor Table Pointer Register – Address 0040

The word at this location is read when the Executive bank descriptor pointer is specified.

### 3.4.2.6. Immediate Storage Check Interrupts – Addresses 0041 – 0042

When an interrupt occurs, these registers temporarily store the captured program return address and designators, respectively.

### 3.4.2.7. Normal Interrupts – Addresses 0043 – 0044

When an interrupt occurs, these registers store the normal captured program return address and designators, respectively.

### 3.4.2.8. User Bank Descriptor Table Pointer Register – Address 0045

The word at this location is read when the user bank descriptor pointer is specified.

### 3.4.2.9. Bank Descriptor Index Registers – Addresses 0046 – 0047

The control register at address 046<sub>8</sub> is used as a holding register for bank descriptors 0 and 2. The register at address 047<sub>8</sub> is used as a holding register for bank descriptors 1 and 3.

### 3.4.2.10. Quantum Timer – Address 0050

When an interrupt occurs the captured quantum timer value is stored in this register.

### 3.4.2.11. Guard Mode – Addresses 0051 – 0053

When a Guard Mode Fault interrupt occurs: the program return address is captured in address 0051<sub>8</sub>, the designators are captured at address 0052<sub>8</sub> and the status is captured at address 0053<sub>8</sub>.

#### 3.4.2.12. Immediate Storage Check Status – Address 0054

When an Immediate Storage Check interrupt occurs the status is stored in this register.

#### 3.4.2.13. Normal Status – Address 0055

Address 0055 stores all processor generated interrupt status except Immediate Storage Check status and Guard Mode status.

#### 3.4.2.14. Unassigned Registers – Addresses 0056 – 0067

The processor uses 0060<sub>8</sub> through 0067<sub>8</sub> as temporary working storage.

#### 3.4.2.15. Jump History Stack – Addresses 0070 – 0077

The jump history stack consists of eight general register locations (070 to 077) that hold recent 24-bit absolute jump instruction addresses. Bit 35 of each entry contains a pass flag indicating whether the entry was stored on an odd or even pass through the stack. Entry stacking is activated by a LBRX instruction, according to the conditions specified in the breakpoint register. Unless terminated by one of these conditions, the process continues in a wraparound manner, and older entries are subsequently overwritten by new entries.

#### 3.4.2.16. Real-Time Clock Register (RO) – Address 0100

The contents of the lower half (bit positions 17–00) of the real-time clock (RTC) register are decreased by one every 200 microseconds, independent of program control or supervision. A Real-Time Clock interrupt occurs if the RTC value in the lower half of the RTC register is zero when a decrementation cycle is initiated. The upper half (bit positions 35–18) of the RTC register should not be used.

#### 3.4.2.17. User (R1) Repeat Count Register – Address 0101

The contents of the repeat counter register define the number of times a repeated instruction is executed. During execution of a repeated instruction the contents of the lower half of the repeat count register are decreased by one each time the repeated instruction is executed. If an interrupt occurs during the sequence of repeated executions of an instruction, the repeat sequence is suspended to process the interrupt, and the current count is left in R1. The repeated sequence may be resumed after the interrupt has been processed. The final value of the count after the repeat sequence terminates is always available in R1. If the contents of the repeat count register is zero, the repeated instruction is not executed and the execution of the next instruction is initiated. Zero is defined as all zeros or all ones in the lower half of the word (bit positions 17–00); the upper half (bit positions 35–18) of the repeat count register should not be used.

#### 3.4.2.18. User (R2)/Mask Register – Address 0102

The bits in the mask register specify the fields of operands to be operated upon in certain instructions. A logical **AND** is performed with the operand and the mask and/or its complement. The portions of the operand so selected are then used in the instruction operation.

#### 3.4.2.19. User (R2–R5)/Staging Registers (SR1–SR3) – Address 0103 – 0105

The three staging registers are used for holding operand information and operation status for byte instruction execution.

#### 3.4.2.20. User (R6–R9)/J-Registers (J0–J3) – Address 0106 – 0111

When designator register bit 4 (D4) is one, j-field values of 4 through 7 specify registers R6 through R9, respectively, instead of partial-word selections. These registers provide character addressing and indexing in a manner that is similar to, and in addition to, the word indexing function of the X-registers.

#### 3.4.2.21. User R-Registers (R10–R15) – Addresses 0112 – 0117

These registers are unassigned and serve as general purpose registers. When D6 = 0, each of these registers can be implicitly addressed by one of the values  $12_8$  through  $17_8$  in the a-field of a Load R or Store R instruction.

#### 3.4.2.22. Executive (R0) R-Register – Address 0120

This register is unassigned and serves as a general purpose register. When D6 = 1, this register is implicitly addressed when the a-field of a Load R or Store R instruction equals zero.

#### 3.4.2.23. Executive (R1) Repeat Count Register – Address 0121

This register has the same function and format as the user R1 repeat count register when D6 = 1.

#### 3.4.2.24. Executive (R2)/Mask Register – Address 0122

This register performs the same function as the user R2 mask register when D6 = 1.

#### 3.4.2.25. Executive (R3–R5)/Staging Registers (SR1–SR3) – Addresses 0123 – 0125

These registers perform the same function as the user SR1 – SR3 staging registers when D6 = 1.

#### 3.4.2.26. Executive (R6–R9)/J-Registers (J0–J3) – Addresses 0126 – 0131

These registers perform the same function as the user J0 – J3 registers when D6 = 1.

#### 3.4.2.27. Executive R-Registers (R10–R15) – Addresses 0132 – 0137

These registers are unassigned and serve as general purpose registers. When D6 = 1, each of these registers can be implicitly addressed by one of the values  $12_8$  through  $17_8$  in the a-field of a Load R or Store R instruction.

#### **3.4.2.28. Executive Index Registers (X1–X15) – Addresses 0141 – 0157**

When  $D6 = 1$ , these registers perform the same functions as the user index registers.

#### **3.4.2.29. Executive Accumulator Registers (A0–A15) – Addresses 0154 – 0173**

When  $D6 = 1$ , these registers perform the same function as the user A–registers.

#### **3.4.2.30. Executive Unassigned Registers – Addresses 0140, 0174 – 0177**

When  $D6 = 1$ , these registers are used in the same manner as the unassigned registers at addresses  $034_8 - 037_8$ .

#### **3.4.2.31. Control Register Protection**

When operating in guard mode ( $D2 = 1$  and  $D6 = 0$ ) a Guard Mode interrupt will occur if an attempt is made to execute a privileged (Executive) instruction or to store data into an Executive GRS location.



## 4. Processor

The 1100/80 Central Processor Unit (CPU) comprises an arithmetic section, control section, maintenance section, general register stack, and interfaces for communicating with other units in the system.

The arithmetic and control sections are discussed in this section. The general register stack (GRS) is discussed in Section 3. A brief discussion of the maintenance section is in Section 2.

### 4.1. ARITHMETIC SECTION

#### 4.1.1. General Operation

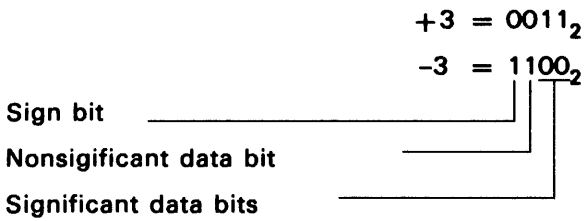
During the execution of logical and arithmetic instructions the following steps are performed:

1. Transfer input data from instruction word specified storage locations or control registers to input registers in the arithmetic section. During the transfer, the input data are processed by the main control section to provide absolute values.
2. Perform the arithmetic operations of addition, subtraction (add negative), multiplication, division, byte manipulation, skip detection, etc., as specified by the instruction word.
3. Transfer final results from the arithmetic section to temporary holding registers, general register storage (GRS), or indicate skip condition.

##### 4.1.1.1. Data Word

The highest order binary bit represents the sign of the value contained in the remaining bit positions. If the sign bit contains a zero, the word is positive and 1's in the remaining bit positions represent significant data. If the sign bit contains a one, the word is negative and 0's in the remaining bit positions represent significant data. A binary data word containing all zeros is referred to as positive zero (+0). A binary data word containing all ones is referred to as negative zero (-0).

Example: (assume a 4-bit word length)



#### 4.1.1.2. Data Word Complement

The ones complement of any binary arithmetic data word is obtained when all zeros in the word are changed to ones and all the ones are changed to zeros. An arithmetic data word of positive value, when complemented, becomes a negative value; and a negative value, when complemented, becomes a positive value.

#### 4.1.1.3. Absolute Values

The absolute value of an arithmetic number is the magnitude of the number regardless of the sign.

<u>Example:</u>	Binary Value	Absolute Value
	001110 (+14)	001110 (14)
	110001 (-14)	001110 (14)

#### 4.1.2. Microprogrammed Control

The 1100/80 arithmetic section consists of three major parallel data paths. The main adder, the shifter, and the multiplier. The cycle time of the main adder and the shifter are both 150 nanoseconds. The multiplier is designed to have a 4-bit add and shift cycle time of 100 nanoseconds. The main adder and shifter are extensively microprogram-controlled; however, the multiplier repeat cycles are self-initiated.

#### 4.1.3. Main Adder Characteristics

The "main adder" of the processor arithmetic section performs single- or double-precision adds or subtracts, or logical operations.

#### 4.1.4. Fixed-Point Single- or Double-Precision Add or Subtract Overflow and Carry

In fixed-point arithmetic, the execution of certain instructions can result in an overflow or a carry condition. During execution, D1 and D0 are cleared to zeros; the overflow and carry conditions set bits D1 and D0, respectively, in the designator register. These bits can be sensed by certain other instructions. Each of these designators, when set to one, remains in the "set" condition until the next time any one of the instructions in Table 4-1 is executed or until the Load Designator Register instruction is executed.

#### 4.1.4.1. Overflow

An "overflow" condition is detected when one of the ten instructions in Table 4-1 is executed and the numeric value of the result obtained exceeds the maximum numeric value that can be contained in the register holding the final result. Under this condition the resulting sign will be incorrect, an overflow enable is generated and sent to control and designator D1 is set.

Table 4-1. Instructions that Condition the Carry and Overflow Designators

Function Code (Octal)	Instruction
f = 14, j = 00-17	Add to A
f = 15, j = 00-17	Add Negative to A
f = 16, j = 00-17	Add Magnitude to A
f = 17, j = 00-17	Add Negative Magnitude to A
f = 20, j = 00-17	Add Upper
f = 21, j = 00-17	Add Negative Upper
f = 24, j = 00-17	Add to X
f = 25, j = 00-17	Add Negative to X
f = 71, j = 10	Double-Precision Fixed-Point Add
f = 71, j = 11	Double-Precision Fixed-Point Add Negative

#### 4.1.4.2. Carry

A "carry" condition is detected when an arithmetic carry is generated out of the sign bit position as the result of the addition of two numeric values. The sign combinations which could set designator DO to a 1-bit indicating that a carry has occurred.

#### 4.1.4.3. Arithmetic Interrupt

The arithmetic section cannot cause a system interrupt. But, when an arithmetic fault occurs, it generates a fault condition signal which allows the control section to set the appropriate designator bit. Other processor conditions in conjunction with those arithmetic fault conditions determines whether or not control generates an interrupt.

#### 4.1.5. Fixed-Point Division

The process of dividing one fixed-point number by another consists of transferring the numbers to the arithmetic section, performing a series of trial subtractions to form a quotient and a remainder, transferring the properly signed quotient to a register and, if the remainder is to be saved, transferring the properly signed remainder to another register. All divide operations use the main adder and shifter.

#### 4.1.6. Fixed-Point Multiplication

The arithmetic section contains a fast multiplier unit to handle multiplications. The main adder and shifter are used only in the beginning and ending cycles for input and output data adjustments.

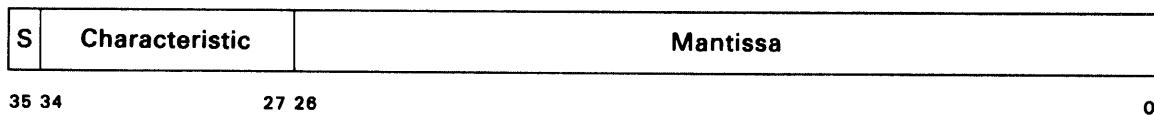
#### 4.1.7. Floating-Point Arithmetic

Floating-point arithmetic handles the scaling problems which arise in computations involving numbers which vary widely in range. In floating-point arithmetic, the numbers are represented in a special format so that the computer can automatically handle the scaling.

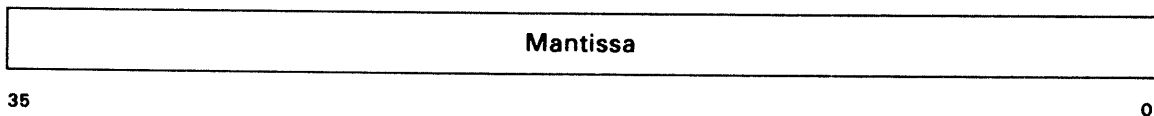
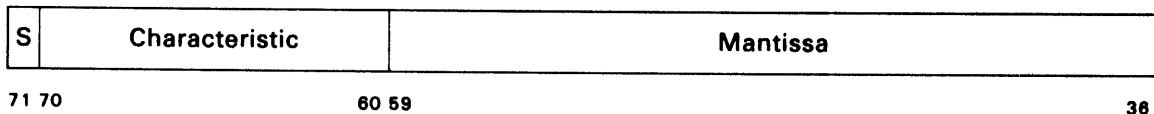
#### 4.1.8. Floating-Point Numbers and Word Formats

Floating-point numbers in the instructions are represented in single-precision format as a 27-bit fractional quantity multiplied by the appropriate power of two, or in the double-precision format as a 60-bit fractional quantity multiplied by the appropriate power of two. The power of two is called the exponent. In machine representation, the exponents are biased to make them lie in the range of positive numbers or zero. These biased exponents are called characteristics. The fractional part is referred to as the mantissa. The two format types, single-precision and double-precision, are as follows.

##### *Single-Precision Floating-Point Format*



##### *Double-Precision Floating-Point Format*



An explanation of the sign bit, characteristic, and mantissa follows:

- **SIGN** – The sign bit expresses the sign (S) of the numerical quantity represented by the floating-point number.
  - If S = 0, the numerical quantity is positive (+).
  - If S = 1, the numerical quantity is negative (-).
- **CHARACTERISTIC** – The characteristic represents both the numerical value and the sign of the exponent.

1. **Single-Precision Characteristic** - The 8-bit characteristic of a single-precision floating-point number represents an exponent value in the range +127 through -128. The characteristic is formed by adding a bias of +128 ( $200_8$ ) to the exponent. Table 4-2 shows the range of characteristic values and corresponding exponent values.

Table 4-2. Single-Precision Floating-Point Characteristic Values and Exponent Values

Decimal Values		Octal Values	
Characteristic	Unbiased Exponent	Characteristic	Unbiased Exponent
255	+127	377	+177
128	000	200	000
000	-128	000	-200

2. **Double-Precision Characteristic** - The 11-bit characteristic of a double-precision floating-point number represents an exponent value in the range +1023 through -1024. The characteristic is formed by adding a bias of +1024 ( $2000_8$ ) to the exponent. Table 4-3 shows the range of characteristic values and the corresponding exponent values.

Table 4-3. Double-Precision Floating-Point Characteristic Values and Exponent Values

Decimal Values		Octal Values	
Characteristic	Unbiased Exponent	Characteristic	Unbiased Exponent
2047	+1023	3777	+1777
1024	0000	2000	0000
0000	-1024	0000	-2000

- **MANTISSA** - The mantissa portion of a floating-point number represents the fractional part of the number. In the instructions the fractional part is normalized so that the absolute values represented are greater than or equal to 1/2 but less than one. Zero cannot be represented in this range and it is considered to be normalized as it stands. The binary point of a

floating-point number is assumed to lie between the last bit of the characteristic and the first bit of the mantissa. The mantissa of a single-precision floating-point number contains 27 bits; for a double-precision floating-point number, the mantissa contains 60 bits. The mantissa need not be normalized for all instructions.

#### 4.1.8.1. Single-Precision Floating-Point Numbers

A single-precision floating-point number can be derived from a positive decimal number as follows:

Example                      Given number =  $+12_{10}$

$$+12_{10} = 1100_2 = .1100_2 \times 10_2 + 4$$

- Sign = + = 0
- Characteristic = exponent + bias  
=  $00\ 000\ 100_2 + 10\ 000\ 000_2$   
=  $10\ 000\ 100_2$
- Mantissa =  $.110\ 000\ \dots\ 000_2$
- The format for the floating-point number is as shown (sign included):

Sign	Characteristic	Mantissa
0	10 000 100	1100 ..... 0

= 204600000000<sub>8</sub>

35 34
27 26
0

#### 4.1.8.2. Double-Precision Floating-Point Numbers

A double-precision floating-point number can be derived from a positive decimal number following the same steps that were used for single-precision with these two exceptions:

- A bias value of  $2000_8$  is added to the exponent to form the characteristic. For single-precision the value is  $200_8$ .
- The single-precision floating-point number for  $-12_{10}$  (including sign) is  $573\ 177\ 777\ 777_8$ .

#### 4.1.8.3. Negative Floating-Point Numbers

A floating-point number can be derived to represent a given negative number as follows:

- Represent the given number as a positive floating-point number.
- Form the ones complement of the entire positive floating-point number.

Example Given number =  $-12_{10}$

- The single-precision floating-point number for  $+12_{10}$  (including sign) is  $204\ 600\ 000\ 000_8$ .
- The single-precision floating-point number for  $-12_{10}$  (including sign) is  $573\ 177\ 777\ 777_8$ .

#### 4.1.8.4. Residue

When a single-precision floating-point add or add negative operation is performed, the result consists of two single-precision floating-point numbers. One of the numbers represents the algebraic sum and the other number is the residue.

When the two 36-bit input operands for an Add or Add Negative instruction are transferred to the arithmetic section, their characteristics are examined, and the mantissa of the input operand with the smaller characteristic is right-shifted a number of bit positions equal to the difference between the characteristics. The bits shifted out of the 36-bit arithmetic register are saved in an auxiliary register. The portion of the mantissa saved in the auxiliary register is used to form the residue and it is not included in the algebraic addition. After completion of the addition and any shifting necessary to normalize the sum, the sum and the residue are packed into single-precision floating-point format and transferred to two consecutive registers.

#### 4.1.9. Normalized/Unnormalized Floating-Point Numbers

A floating-point number is normalized when the leftmost bit of the mantissa is not identical to the sign bit or when all bits of the mantissa are identical to the sign bit. A floating-point number is unnormalized when all bits of the mantissa are not sign bits and the leftmost bit of the mantissa is identical to the sign bit.

All floating-point operations produce a normalized result when the input operands are normalized. The sums produced by Floating Add and Floating Add Negative instructions and the result produced by the Load And Convert To Floating instruction are always normalized regardless of whether or not the input operands are normalized. When either or both input operands are not normalized, the result obtained may be less accurate than if normalized input operands had been used.

Normalized input operands must be used for the Floating Multiply, Divide, Compress And Load, and Expand And Load instructions. If normalized input operands are not used for these instructions, the results are undefined.

#### 4.1.10. Floating-Point Characteristic Overflow/Underflow

Floating-point characteristic overflow/underflow occurs when the characteristic does not lie in the range representable in the number of bits allowed for the characteristic.

When any of the Floating-Point Add, Add Negative, Multiply, Divide or Load And Convert instructions or the Compress And Load instruction are performed, overflow or underflow may occur.

##### 4.1.10.1. Floating-Point Characteristic Overflow

Single-precision floating-point characteristic overflow occurs when the 8-bit characteristic of the resultant most significant single-precision floating-point word represents a number greater than  $377_8$  and the associated mantissa is not zero.

Double-precision floating-point characteristic overflow occurs when the 11-bit characteristic of the resultant double-precision floating-point number represents a number greater than  $3777_8$  and the associated mantissa is not zero.

When overflow is detected, the action taken depends on designator bit D20. Designator bit D22 is always set.

#### 4.1.10.2. Floating-Point Characteristic Underflow

Single-precision floating-point characteristic underflow occurs when the resultant floating-point word represents a negative number and the associated mantissa of the result is not zero. This means that the exponent of the result is less than  $-200_g$ , thus the attached sign (positive - because absolute value is used) changes due to the borrow. If the characteristic of the residue (Floating Add, Floating Add Negative), remainder (Floating Divide), or the least significant single-precision word of the product (Floating Multiply) represents a negative number, this fact by itself does not result in underflow. Instead, the residue, remainder, or least significant word of the product is cleared to all zero bits or set to all one bits (to reflect the appropriate sign).

Double-precision floating-point characteristic underflow occurs when the 11-bit characteristic of the result represents a negative number, i.e., the exponent of the result is less than  $2000_g$ , the mantissa of the result is not zero, and designator D5 is cleared.

When underflow is detected, designator D21 is always set and the action taken by the processor depends on the state of designator D20.

#### 4.1.10.3. Floating-Point Divide Fault

For single- or double-precision floating-point division a divide fault condition will be detected when the mantissa of the divisor is zero. The action taken depends on designator D8 (for single-precision floating-point division only) and designator D20. Designator D23 is always set.

#### 4.1.11. Fixed-Point to Floating-Point Conversion

Conversion of a fixed-point number to floating-point number is performed in the arithmetic section. The first input operand contains a characteristic (biased exponent) which defines the location of the binary point for the fixed-point number with respect to the standard position of the binary point for a floating-point number. The second input operand is the signed fixed-point number to be converted.

The conversion process consists of transferring the two operands to the arithmetic section, shifting the fixed-point number, if necessary, to position its bits as the mantissa for a normalized floating-point number. Modify the characteristic to reflect the magnitude and direction of the normalizing shift. Pack the shifted fixed-point number (mantissa) and the modified characteristic in floating-point format. Load the packed results in a register (conversion to single-precision floating-point format) or into two consecutive registers (conversion to double-precision floating-point format).

#### 4.1.12. Floating-Point Addition

The process of adding two floating-point numbers consists of loading the numbers into the arithmetic section, determining the difference between the characteristics of the two numbers, shifting (right) the mantissa of the number having the smaller characteristic, adding the mantissas, combining the results in floating-point format, and transferring the resulting floating-point numbers to GRS.

The input operands for floating-point addition need not be normalized numbers. For single-precision addition, the sum (most significant word produced) is always a normalized number. The residue word



may or may not be a normalized number. For double-precision addition, the sum is always a normalized number.

#### 4.1.12.1. Double-Precision Floating-Point Addition

The steps performed for double-precision floating-point addition are similar to those for the single-precision addition with these six differences:

1. Each of the two operands occupy two 36-bit registers in the arithmetic section. In single-precision addition both operands are contained in two 36-bit registers.
2. The mantissa sum can contain a maximum of 60 bits in double-precision addition instead of 27 bits as in single-precision addition.
3. The bits that are shifted out of the right end of the 36-bit register when the operands are lined up prior to addition are lost. There is no residue.
4. Double-precision characteristic overflow occurs when the characteristic is greater than  $3777_8$  and the mantissa is not zero.
5. Double-precision underflow occurs when the exponent is less than  $-2000_8$  and the mantissa is not zero. In single-precision the value is  $-200_8$ .
6. The sum is stored in two consecutive registers, A and A + 1. No residue is stored.

#### 4.1.13. Floating-Point Subtraction (Add Negative)

Floating-point subtraction (both single-precision and double-precision) uses the same routine as for the Floating-point Add operation.

#### 4.1.14. Floating-Point Multiplication

The process of multiplying two floating-point numbers consists of loading normalized input operands into the arithmetic section, unpacking, multiplying the mantissas, adding the characteristics, packing the results into floating-point format, and transferring the result to GRS. The results obtained for all cases in which either or both input operands are not normalized numbers are undefined.

#### 4.1.15. Floating-Point Division

The process of dividing one floating-point number by another consists of loading the normalized input operands into the arithmetic section, unpacking, dividing one mantissa by the other, subtracting the characteristics, packing the results into floating-point format, and transferring the result to GRS. The results obtained for all cases in which either or both input operands are not normalized numbers are undefined.

#### 4.1.16. Floating-Point Zero

Floating-point zero can be defined as a floating-point number having all mantissa bits identical to the sign bit.

#### 4.1.17. Byte Instructions

This class of instructions is designed to permit transference, translation, comparison, and arithmetic computation of data in the form of predetermined bit patterns (e.g., half words, third words, quarter words, and sixth words) referred to as bytes.

There are a total of 15 distinct instructions that perform the various multiword (byte string) operations noted above. These instructions may be arranged under three functional groups:

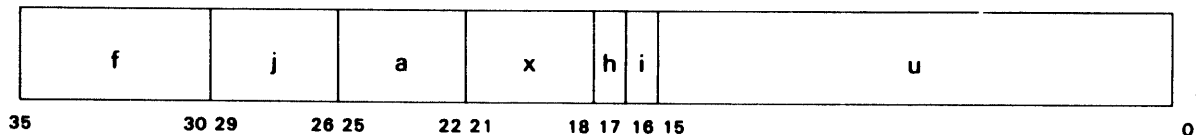
1. instructions that involve byte transfers and manipulations between one storage location and another.
2. instructions that permit the mutual transference and manipulation of data among storage and various control and arithmetic registers, and
3. instructions that perform decimal arithmetic addition and subtraction operations.

Twelve of the byte instructions are performed in the arithmetic section. The remaining three instructions (33,00 – Byte Move, 33,01 – Byte Move with Translate, and 33,07 – Edit) are performed in the main control section.

### 4.2. CONTROL SECTION

#### 4.2.1. Instruction Word Format

During the running of a program in the SPERRY UNIVAC 1100/80 Processor, instructions are transferred from main storage locations to the control section of the Central Processing Unit (CPU). The instructions are transferred from sequentially addressed main storage locations until the sequence is broken by the program or interrupted by the control section's reaction to some special condition or event. Each instruction is a coded directive to the control section; the control section initiates a sequence of steps necessary to perform the particular operation prescribed by the instruction. The 36-bit instruction word, illustrated below, is subdivided into seven fields.



where:

f = Function Code

j = Operand Qualifier, Character Addressing, partial Control Register Address, or Minor Function Code

a = A-, X-, or R-register; Channel number, Jump Key or Stop Keys number; Minor Function Code; partial Control Register Address

x = Index Register

- h** = Index Register Incrementation Control
- i** = Indirect Addressing Control, Base Register Suppression Control, 24-Bit Indexing Control, or Operand Basing Selector
- u** = Operand Address or Operand Base

#### 4.2.2. Instruction Word Fields

The following paragraphs describe the manner in which the CPU's control section reacts to the contents of each of the seven fields of an instruction word.

##### 4.2.2.1. Use of the f-Field

The f-field is used to define the basic operation to be performed for all legal values of f less than or equal to  $70_8$  (except  $07_8$ ,  $33_8$ , and  $37_8$ ). When the f-field is  $07_8$ ,  $33_8$ ,  $37_8$  or greater than  $70_8$ , the f- and j-fields are combined to form a 10-bit field used to define the basic operation. For eleven of these f, j combinations, the value in the a-field is used to define variations of the basic operation. All function codes are defined in Section 5 and listed in Appendix C.

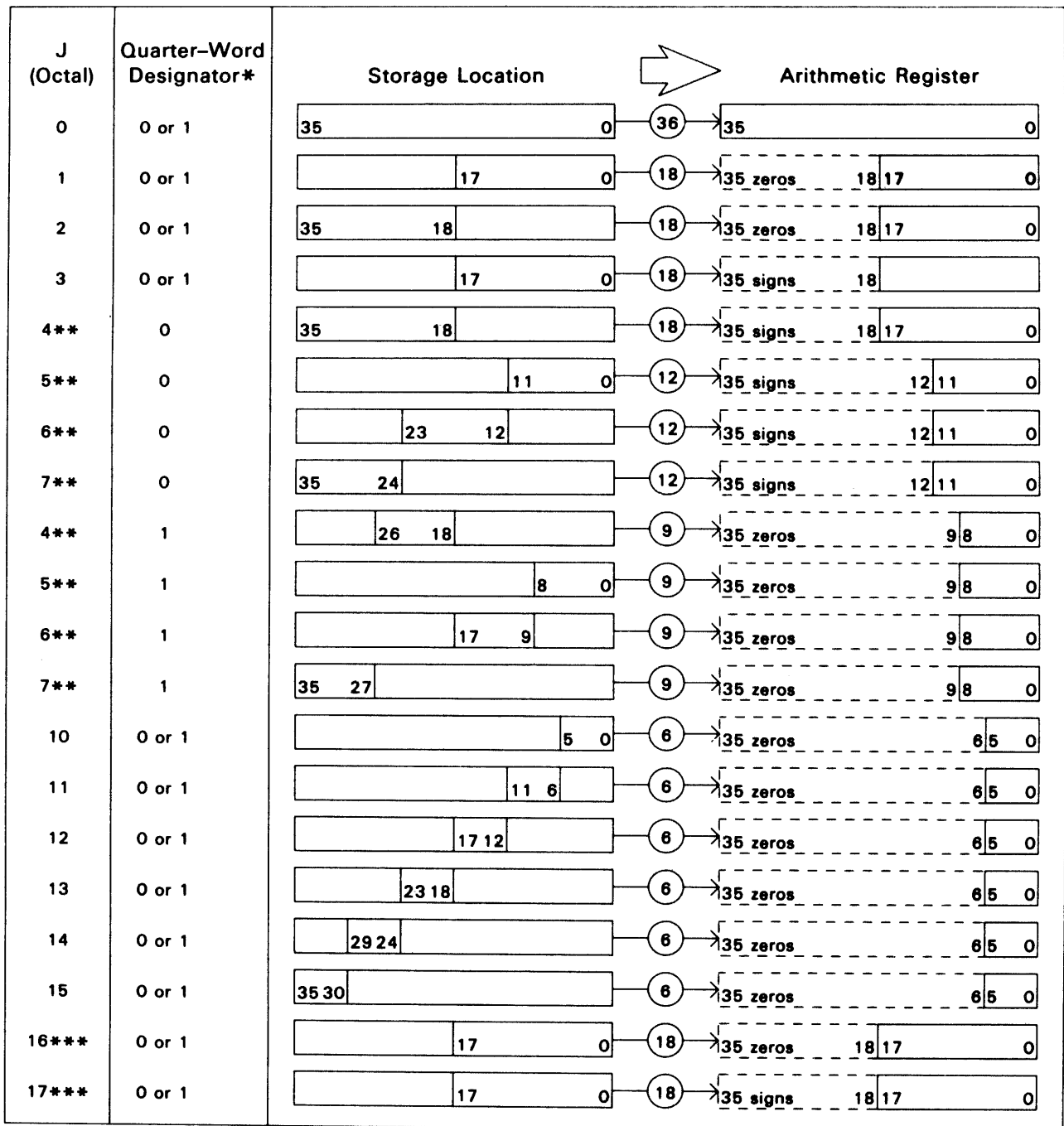
##### 4.2.2.2. Description of the j-Field

When f is less than  $70_8$  (except  $07_8$ ,  $33_8$  and  $37_8$ ), the j-field is used as an operand qualifier or to identify a J-register used in the character addressing mode. When f is equal to  $70_8$ , the j-field is used as part of a control register address. When f is  $07_8$ ,  $33_8$ ,  $37_8$  or greater than  $70_8$ , the j-field and the f-field are used to define a basic operation, and, in this instance, the j-field operates as a minor function code.

##### 4.2.2.2.1. Use of the j-Field as an Operand Qualifier

When the f-field of an instruction contains a value in the range  $01_8$  through  $67_8$  (except  $07_8$ ,  $33_8$  and  $37_8$ ) and  $D4 = 0$ , the j-field is used as an operand qualifier which specifies the data transfer pattern to or from main storage except as specified in 4.2.2.2.2.

The j-field can contain values ranging from  $00_8$  through  $17_8$ . Each value except  $4_8$  through  $7_8$  determines a specific data transfer pattern. Each of the j-field values  $4_8$  through  $7_8$  may specify either of two different data transfer patterns, or character addressing with the choice dependent on the contents of the quarter word mode selector ( $D10$ ) and the character addressing mode selector ( $D4$ ) of the designator register (See 8.2.1). If  $D4 = 1$ , character addressing is specified and each of the j-field values  $4_8$  through  $7_8$  specify a J-register as explained in 4.2.2.2.2. Figures 4-1 and 4-2 illustrate all the possible data transfer patterns which can be specified by the j-field when  $D4 = 0$ .

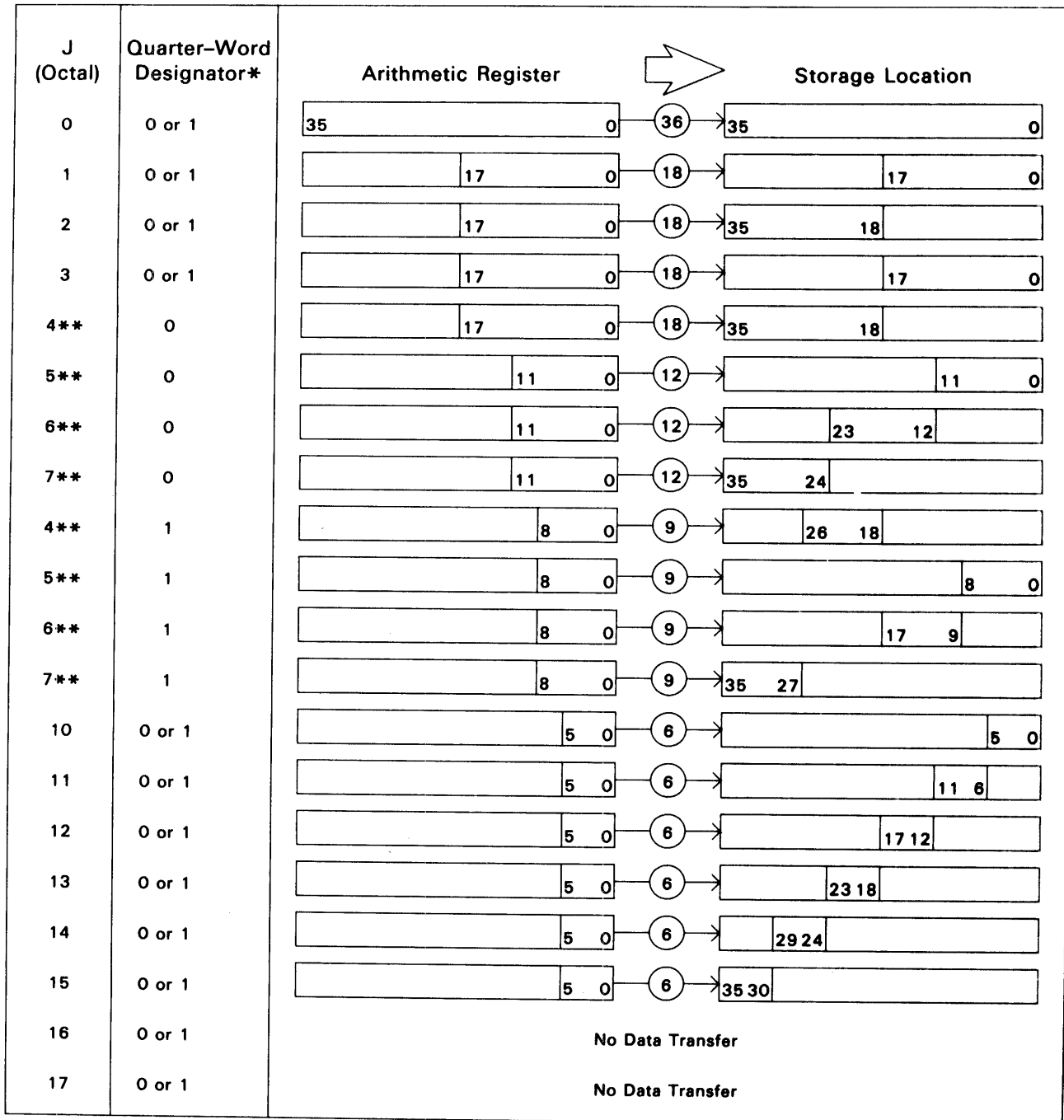


\* The Quarter-Word Designator bit (D10) is held in the designator register.

\*\* Character Addressing Designator bit (D4) will imply J-Register usage for instruction codes less than f = 70 (except 07, 33, 37) for character or byte manipulation. D4 overrides D10.

\*\*\* If x = 0, the h, i, and u are transferred. If x is not equal to zero, then u + (X<sub>x</sub>) is transferred.

Figure 4-1. Data Transfers From Storage



\* The Quarter-Word Designator bit (D10) is held in the designator register.

\*\* Character Addressing Designator bit (D4) will imply J-Register usage for instruction codes less than f = 70 (except 07, 33, 37) for character or byte manipulation. D4 overrides D10.

Figure 4-2. Data Transfers to Storage

■ Operand qualification when  $f = 10_8$  through  $67_8$  (except  $33_8$  and  $37_8$ )

These instructions require the transfer of a full 36-bit word or a partial word to the arithmetic section.

1. If  $j = 00_8$ , the full 36-bit word addressed by U is transferred to the arithmetic section.
2. If  $j = 01_8$  through  $15_8$  and U specifies a main storage location ( $U \geq 200_8$ ), a partial word is transferred to the arithmetic section. In the arithmetic section, the partial word is extended to a full 36-bit word either by zero fill or by sign bit fill from the leftmost bit position of the partial word, as illustrated in Figure 4-1.
3. If  $j = 16_8$  or  $17_8$ , an 18-bit partial word is transferred to the arithmetic section. Details on the formation of this partial word and its extension are given in 4.2.2.8.2.

When  $j = 01_8$  through  $15_8$  and U specifies a control register ( $U \leq 177_8$ ), the  $j$ -field is treated as if it contained  $00_8$  and the full 36-bit word is transferred from the control register to the arithmetic section.

■ Operand qualification for store and block transfer instructions

The full 36-bit word in the control register specified by the  $a$ -field (see 4.2.2.3.1) is transferred to a nonaddressable register in the data shift/complement/store section ( $f=01_8$  through  $04_8$  and  $06_8$ ). The nonaddressable register is cleared to 0 when  $f=05_8$ .

1. If  $j = 00_8$ , the full 36-bit word is transferred from the nonaddressable register to the location (main storage or control register) specified by U.
2. If  $j = 01_8$  through  $15_8$  and U specifies a main storage location ( $U \geq 200_8$ ), a partial word is transferred from the least significant bit positions of the nonaddressable register to the main storage location as shown in Figure 4-2. The contents of the remaining bit positions of the main storage location are not changed. Partial word writes of a third word, quarter word, or sixth word increase the storage cycle time to 200 nanoseconds.
3. If  $j = 16_8$  or  $17_8$ , data is never transferred from the nonaddressable register to any storage location (main storage or control register).

When  $j = 01_8$  through  $15_8$  and U specifies a control register ( $U \leq 177_8$ ), the  $j$ -field is treated as if it contained  $00_8$ , and the full 36-bit word is transferred to the control register.

#### 4.2.2.2.2. Use of the $j$ -Field to Specify Character Addressing

When the  $f$ -field of an instruction contains a value in the range  $01$  through  $67_8$  (except  $07_8$ ,  $33_8$ , and  $37_8$ ),  $D4$  (the character addressing mode selector) = 1, and  $j = 4_8$ ,  $5_8$ ,  $6_8$ , or  $7_8$ , the character addressing mode is specified. When the character addressing mode is specified, a  $j$ -field value of 4, 5, 6, or 7 specifies  $J0$ ,  $J1$ ,  $J2$ , or  $J3$ , respectively, in the GRS, as the register defining character or byte size, the position of the byte within a word, and other details. When  $D6 = 0$ , the  $J$ -register is selected from the set of four  $J$ -registers at GRS locations  $106$  through  $111_8$ . When  $D6 = 1$ , the  $J$ -register is selected from the set of four  $J$ -registers at GRS locations  $126$  through  $131_8$ . The format of a  $J$ -register word as used in the character addressing mode is shown in Figure 4-3 and explained in Table 4-4.

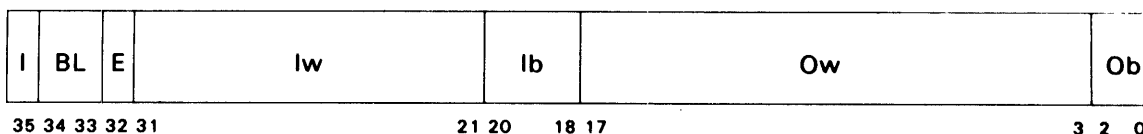


Figure 4-3. J-Register Format for Character Addressing Mode

Table 4-4. Explanation of J-Register Fields for Character Addressing Mode

Bit Positions	J-Register Field Identifier	Interpretation
35	I	<p>The I-bit of the J-Register in conjunction with the h-bit of the instruction, specifies whether or not the contents of the Ow and Ob-fields are modified when the instruction is performed as follows:</p> <ul style="list-style-type: none"> <li>■ I = 0 or h = 0 specifies no J register modification*</li> <li>■ I = h = 1 specifies modification of Ow and Ob by lw and lb, respectively.</li> </ul>
34,33	BL	<p>Specifies the byte length, as follows:</p> <ul style="list-style-type: none"> <li>■ BL = 0 specifies a 9 bit byte</li> <li>■ BL = 1 specifies an 18 bit byte</li> <li>■ BL = 2 specifies a 6 bit byte</li> <li>■ BL = 3 specifies a 12 bit byte</li> </ul>
32	E	<p>Specifies the bit used to extend the byte to 36 bits, if necessary, as follows:</p> <ul style="list-style-type: none"> <li>■ E = 0 specifies extension with 0 bits</li> <li>■ E = 1 specifies extension with the high order bit of the byte</li> </ul>
31-21	lw	<p>lw specifies the increment (or decrement) in words. lb specifies the increment (or decrement) in bytes. If I = 0, or h = 0 these two values are ignored.</p>
20-18	lb	<p>If I = 1 and h = 1, the values in the lw and lb-fields are added to the values in the Ow and Ob-fields, and the sums are stored in the Ow and Ob-fields after the initial values in these two fields are used to form the absolute address of a word and select a byte within the word.</p>
17-3	Ow	<p>The offset in words. This value is used to form the relative address U and the absolute addresses SI and SD.</p>

Table 4-4. Explanation of J-Register Fields for Character Addressing Mode (continued)

Bit Positions	J-Register Field Identifier	Interpretation
2-0	Ob	The offset in bytes. This value is used to select a particular byte within the selected word. The valid values of Ob for the possible values of BL are shown in Figure 4-4. Other byte selections are not defined.

\* If  $l = 0$ ,  $h = 1$  in the instruction word specifies index register modification when  $x \neq 0$ .

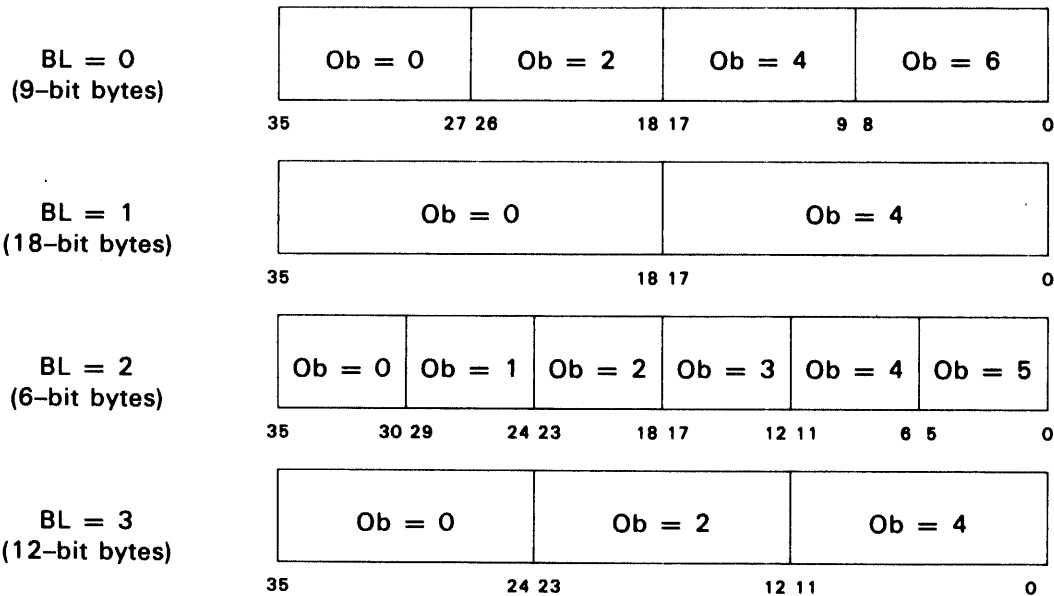


Figure 4-4. Byte Selected for Valid Combinations of BL and Ob Field Values

The additions performed when  $l = 1$  and the  $h$ -bit of the instruction = 1 are symbolized by

$$Ob + lb \rightarrow Ob$$

and

$$Ow + lw \rightarrow Ow$$

The values in the  $Ow$ - and  $Ob$ -fields are always treated as positive values in these additions. The high order bit in the  $lw$ -field (bit 31 of the specified J-register) is applied as the sign of both the  $lw$ - and  $lb$ -fields. If this sign is a zero bit, forward modification of  $Ow/Ob$  is performed. Forward modification permits incrementing the  $Ow$ - or  $Ob$ -field value (or both) to produce new  $Ow$ - and  $Ob$ -field values to select any desired byte in lower order bit positions of the same word or select any desired byte in any word having a higher address. If the sign bit applied to the  $lw$ - and  $lb$ -fields is a one bit, backward modification of  $Ow/Ob$  is performed. Backward modification permits decrementing the  $Ow$ - and  $Ob$ -field value (or both) to produce new  $Ow$ - and  $Ob$ -field values to select



any desired byte in higher order bit position of the same word or select any desired byte in any word having a lower address.

The result produced for the addition  $Ob + Ib \rightarrow Ob$  is dependent on the two values used as inputs, the sign in the  $lw$ -field and the value in the  $BL$ -field, as shown in Tables 4-5 through 4-8. The valid combinations of  $Ob$  and  $Ib$  are shown in these tables. The result produced when any other combination is used is undefined.

The addition  $Ow + lw \rightarrow Ow$  is performed in an 18-bit ones complement subtractive adder after extending the 15-bit  $Ow$ -field to 18 bits with three high order zero bits and extending the 11-bit  $lw$ -field to 18 bits with seven high order bits identical to the sign bit in the  $lw$ -field. A carry or borrow generated in the addition  $Ob + Ib \rightarrow Ob$  also enters the  $Ow + lw \rightarrow Ow$  addition. The sum is stored in the  $Ow$  field of the specified  $J$ -register after the initial value in the  $Ow$  field is used to form the relative and absolute addresses needed for the instruction.

If the value in the  $Ow$  field is modified by adding a positive  $lw$  value to produce an 18-bit sum greater than  $077777_8$  or by adding a negative  $lw$  value to produce a negative 18-bit sum, the 15-bit value stored in  $Ow$  is undefined. Producing a negative 18-bit sum is a common programming error which can be avoided by choosing an artificially high-initial value for  $Ow$  and reducing the initial value of  $Xm$  by a like amount.

If the value  $U$  produced by the addition  $u + Xm + Ow$  (see 4.2.2.5) for an instruction which specifies the character addressing mode is less than  $200_8$ , a register in the GRS is not referenced. Instead, the values  $SI$  and  $SD$  are produced, and if  $U$  passes the storage limits test, an attempt is made to reference main storage.

Table 4-5. Output  $Ob$  Values Produced When  $BL = 0$

BL = 0 (9-Bit Bytes)	Number Of Bytes Forward Or Backward	Valid Ib Value	Valid Input Ob Values			
			0	2	4	6
Forward Modification ( $J_{31} = 0$ )	0	0	0	2	4	6
	1	2	2	4	6	$0_C$
	2	4	4	6	$0_C$	$2_C$
	3	6	6	$0_C$	$2_C$	$4_C$
Backward Modification ( $J_{31} = 1$ )	0	7	0	2	4	6
	1	5	$6_B$	0	2	4
	2	3	$4_B$	$6_B$	0	2
	3	1	$2_B$	$4_B$	$6_B$	0

For valid  $Ob/Ib$  input combinations

C = Carry (+ 1) to  $Ow + lw \rightarrow Ow$  addition

B = Borrow (- 1) to  $Ow + lw \rightarrow Ow$  addition

Table 4-6. Output Ob Value Produced When BL = 1

BL = 1 (18-Bit Bytes)	Number Of Bytes Forward Or Backward	Valid lb Value	Valid Input Ob Values	
			0	4
Forward Modification (J <sub>31</sub> = 0)	0	0	0	4
	1	4	4	0 <sub>C</sub>
Backward Modification (J <sub>31</sub> = 1)	0	7	0	4
	1	3	4 <sub>B</sub>	0

For Valid Ob/lb Input Combinations

C = Carry (+ 1) to Ow + lw → Ow addition

B = Borrow (- 1) to Ow + lw → Ow addition

Table 4-7. Output Ob Values Produced When BL = 2

BL = 2 (6-Bit Bytes)	Number Of Bytes Forward Or Backward	Valid lb Value	Valid Input Ob Values					
			0	1	2	3	4	5
Forward Modification (J <sub>31</sub> = 0)	0	0	0	1	2	3	4	5
	1	1	1	2	3	4	5	0 <sub>C</sub>
	2	2	2	3	4	5	0 <sub>C</sub>	1 <sub>C</sub>
	3	3	3	4	5	0 <sub>C</sub>	1 <sub>C</sub>	2 <sub>C</sub>
	4	4	4	5	0 <sub>C</sub>	1 <sub>C</sub>	2 <sub>C</sub>	3 <sub>C</sub>
	5	5	5	0 <sub>C</sub>	1 <sub>C</sub>	2 <sub>C</sub>	3 <sub>C</sub>	4 <sub>C</sub>
Backward Modification (J <sub>31</sub> = 1)	0	7	0	1	2	3	4	5
	1	6	5 <sub>B</sub>	0	1	2	3	4
	2	5	4 <sub>B</sub>	5 <sub>B</sub>	0	1	2	3
	3	4	3 <sub>B</sub>	4 <sub>B</sub>	5 <sub>B</sub>	0	1	2
	4	3	2 <sub>B</sub>	3 <sub>B</sub>	4 <sub>B</sub>	5 <sub>B</sub>	0	1
	5	2	1 <sub>B</sub>	2 <sub>B</sub>	3 <sub>B</sub>	4 <sub>B</sub>	5 <sub>B</sub>	0

For valid Ob/lb combinations

C = Carry (+ 1) to Ow lw → Ow addition

B = Borrow (- 1) to Ow lw → Ow addition

Table 4-8. Output Ob Values Produced When BL = 3

BL = 3 (12-Bit Bytes)	Number Of Bytes Forward Or Backward	Valid Ib Value	Valid Input Ob Values		
			0	2	4
Forward Modification (J <sub>31</sub> = 0)	0	0	0	2	4
	1	2	2	4	0 <sub>C</sub>
	2	4	4	0 <sub>C</sub>	2 <sub>C</sub>
Backward Modification (J <sub>31</sub> = 1)	0	7	0	2	4
	1	5	4 <sub>B</sub>	0	2
	2	3	2 <sub>B</sub>	4 <sub>B</sub>	0

For valid Ob/Ib combinations

C = Carry (+ 1) to Ow lw → Ow addition

B = Borrow (- 1) to Ow lw → Ow addition

#### 4.2.2.2.3. Use of j-Field as Partial Control Register Address

When  $f = 70_8$ , the most significant bit of the j-field is ignored by the hardware, and the three low-order bits are combined with the contents of the a-field to form a 7-bit control register address.

#### 4.2.2.2.4. Use of j-Field as Minor Function Code

When  $f = 07_8, 33_8, 37_8$ , or  $71_8$  through  $76_8$ , the value in the j-field is a minor function code designator. An explanation of the details of each of these instructions is given in Section 5 they are summarized in Appendix C.

#### 4.2.2.3. Uses of the a-Field

The contents of the a-field of an instruction word has a number of uses. The exact use is dependent on the instruction being performed and, in many cases on the contents of the designator register.

##### 4.2.2.3.1. Use of the a-Field to Reference an A-Register

For most of the instructions, the value in the a-field references one of the A-registers. When the A-, X-, and R-register set selector, D6, is equal to 0, each value in the range  $00_8$  through  $17_8$  in the a-field references one of the user A-registers in the range of control register addresses  $14_8$  through  $33_8$ , respectively. When  $D6 = 1$ , each value in the range  $00_8$  through  $17_8$  in the a-field references one of the Executive A-registers in the range of control register addresses  $154_8$  through  $173_8$  respectively. In some instructions, the value in the a-field references two or three A-registers. When two or three A-registers are referenced, the value in the a-field explicitly references register Aa, and implicitly references registers Aa+1 and Aa+2.

The unassigned control registers at addresses  $34_8$ ,  $35_8$ ,  $174_8$  and  $175_8$  can be used as extensions of the two sets of 16 A-registers. For example, when  $a = 17_8$  and the instruction requires referencing three A-registers ( $Aa$ ,  $Aa+1$ , and  $Aa+2$ ) then:

- If  $D6 = 0$ , the last user A-register (address  $33_8$ ) is referenced for  $Aa$ , the first user unassigned control register at address  $34_8$  is referenced for  $Aa+1$ , and address  $35_8$  is referenced for  $Aa+2$ .
- If  $D6 = 1$ , the last Executive A-register at address  $173_8$  is referenced for  $Aa$ , the following Executive unassigned control register at address  $174_8$  is referenced for  $Aa+1$ , and address  $175_8$  is referenced for  $Aa+2$ .

#### 4.2.2.3.2. Use of the a-Field to Reference an X-Register

For certain instructions, the value in the a-field references one of the X-registers. When  $D6 = 0$ , each value in the range of  $01_8$  through  $17_8$  in the a-field references one of the user X-registers in the range of control register addresses  $01_8$  through  $17_8$  respectively; if  $a = 0_8$ , the user nonindexing X-register at control register address  $000_8$  is referenced. When  $D6 = 1$ , each value in the range of  $01_8$  through  $17_8$  in the a-field references one of the Executive X-registers in the range of control register addresses  $141_8$  through  $157_8$  respectively; if  $a = 0_8$ , the Executive nonindexing X-register at control register address  $140_8$  is referenced.

#### 4.2.2.3.3. Use of the a-Field to Reference an R-Register

For certain instructions, the value in the a-field references one of the R-registers. When  $D6 = 0$ , each value in the range of  $00_8$  through  $17_8$  in the a-field references one of the user R-registers at control register addresses  $100_8$  through  $117_8$ , respectively. When  $D6 = 1$ , each value in the range of  $00_8$  through  $17_8$  in the a-field references one of the Executive R-registers at control register addresses  $120_8$  through  $137_8$ , respectively.

#### 4.2.2.3.4. Use of the a-Field to Reference a Jump Key

For a Jump Key instruction, each value in the range of  $01_8$  through  $17_8$  in the a-field references one of the 15 select jump control circuits in the CPU. These circuits may be individually set and cleared via switches on the operator/maintenance panel on the STU.

#### 4.2.2.3.5. Use of the a-Field to Reference Halt Keys

For a Halt Keys and Jump instruction, each of the four bit positions in the a-field references one of the four select stop control circuits in the CPU. These circuits may be individually set and cleared via switches on the STU.

#### 4.2.2.3.6. Use of the a-Field as Minor Function Code

The value in the a-field specifies a particular variation of the basic operation initiated by the  $f$ ,  $j$  combination of the following instructions:

- Load Breakpoint Register/Store Jump Stack,
- Load Processor State Register,

- Initiate Interprocessor Interrupt/Enable Second Day Clock/Enable Day Clock/Disable Day Clock,
- Test and Set/Test and Set and Skip/Test and Clear and Skip,
- Jump Overflow/Jump Floating Underflow/Jump Floating Overflow/Jump Divide Fault,
- Jump No Overflow/Jump No Floating Underflow/Jump No Floating Overflow/Jump No Divide Fault,

#### 4.2.2.4. Use of the j- and a-Fields to Specify GRS Control Register Address

For the Jump On Greater And Decrement instruction, the values in the j-field and a-field combine to form a 7-bit address (the leftmost bit of the j-field is ignored). The 7-bit address specifies which one of the 112 addressable GRS control registers is to be used as the counter for the instruction.

#### 4.2.2.5. Use of the x-Field

An indexing operation which utilizes a ones complement subtractive adder occurs for every instruction. If the A-, X-, and R-register set selector, D6, is equal to 0, each x-field value in the range  $01_8$  through  $17_8$  references one of the user X-registers at control register addresses  $01_8$  through  $17_8$ , respectively. If  $D6 = 1$ , each x-field value in the range  $01_8$  through  $17_8$  references one of the Executive X-registers at control register addresses  $141_8$  through  $157_8$ , respectively. When the value in the x-field is not zero, the value in the Xm-field of the X-register specified by the x-field is added to the extended contents of the u-field to form the relative operand address or an operand. This indexing operation is symbolized by the notation:  $u + X_m = U$  except for instructions which specify the character addressing mode (see 4.2.2.2.2) and for most byte instructions. (See 4.1.17.) In these cases it is symbolized by the notation  $u + X_m + O_w = U$ .  $X_m$  is an 18-bit field unless 24-bit indexing is specified.

When the value in the x-field is zero, no index register is referenced. However, an indexing operation does occur. It consists of adding an 18-bit half word of all zero bits to the extended u-field value to form the relative operand address or operand. This indexing operation is symbolized by the notation:  $u + 0 = U$  or  $u + 0 + O_w = U$ .

An indexing operation never produces a U value consisting of all one bits. This applies when U is a relative address and also when U is extended with zero bits ( $j = 16_8$ ) or with sign bits ( $j = 17_8$ ) for use as an immediate operand.

Example:                    If  $j \neq 16$  or  $17_8$ ,  $u = 000001_8$ , and  $X_m = 777776_8$

then

$$u + X_m = U = 000001_8 + 777776_8 = 000000_8$$

Example:                    If  $f = 10 - 67_8$  (except  $33$  and  $37_8$ ),  $j = 16$  or  $17_8$ ,

$$i = 0, u = 177777_8, \text{ and } X_m = 600000_8,$$

then

$$u + X_m = U = 177777_8 + 600000_8 = 000000_8$$

Example: If  $f = 10 - 67_8$  (except  $33$  and  $37_8$ ),  $j = 16$  or  $17_8$ .

$h = i = 1$ ,  $u = 177777_8$ , and  $x = 0$ .

then

$h, i, u + 0 = U = 777777_8 = 0 = 000000_8$

#### 4.2.2.6. Use of the h-Field

If the  $x$ -field of an instruction contains a nonzero value, the  $h$ -bit determines whether or not the contents of the  $X$ -register specified by the  $x$ -field of the instruction or the  $Ow$ - and  $Ob$ -fields of an instruction specifying the character addressing mode are modified.

After the indexing operation is complete, if  $h = 1$ , and  $x \neq 0$  for an instruction which does not specify the character addressing mode, or an instruction which specifies the character addressing mode and a  $J$ -register containing  $l = 0$  (see 4.2.2.2.2), the contents of the  $X_i$ -field of the specified  $X$ -register are added to the contents of the  $X_m$ -field of the same register and the sum is stored back in the  $X_m$ -field. The process is  $X_m + X_i \rightarrow X_m$ . If  $D7$  or  $i = 0$ , the addition is performed in an 18-bit ones complement subtractive adder. If  $D7 = i = 1$ , the addition is performed in a 24-bit ones complement subtractive adder.

The only time index register modification produces an output of  $-0$  occurs when both inputs are  $-0$ ; that is,  $-0 + -0 = -0$ .

After the indexing operation is complete for an instruction which specifies the character addressing mode and a  $J$ -register containing  $l = 1$ , if  $h = 1$ , the contents of the  $lw$ - and  $lb$ -fields of the  $J$ -register are added to the contents of the  $Ow$ - and  $Ob$ -fields, respectively, as explained in 4.2.2.2.2. In this case  $X_m$  is not modified.

The modification of  $X_m$  or of  $Ow$  and  $Ob$  are performed without increasing instruction execution time.

If  $h = 0$ , neither  $X_m$  nor the  $Ow$ - or  $Ob$ -field is modified; the  $l$ -bit is ignored in this case.

#### 4.2.2.7. Use of the i-Field

The  $i$ -field can be used to specify normal addressing, indirect addressing, absolute addressing, or to extend the  $u$ -field of an instruction.

If  $i = 1$  and  $D7 = 0$ , indirect addressing occurs for all instructions except when  $f = 01_8$  through  $06_8$ ,  $10_8$  through  $32_8$ ,  $34_8$  through  $36_8$ ,  $40_8$  through  $67_8$  and  $j = 16_8$  or  $17_8$ . For the exception,  $x \neq 0$  is also a required condition for indirect addressing.

Indirect addressing will not occur if:

- $i = 0$
- $f = 01_8$  through  $06_8$ ,  $10_8$  through  $32_8$ ,  $34_8$  through  $36_8$ ,  $40_8$  through  $67_8$ ,  $j = 16_8$  or  $17_8$  and  $x = 0$ . (Then the  $i$ -field is used as an extension of the  $u$ -field.)
- $D7 = 1$  (Then  $i = 1$  specifies absolute addressing,  $i = 0$  specifies normal address generation of  $u + X_m$ .)

The above cases are summarized in Table 4-9.

Table 4-9. Summary of Use of i-Field

i	D7	All Instructions	Exceptions	
			f is less than 70 (Except 07, 33 and 37) and j=16 or 17 and: x ≠ 0	x = 0
0	0	Normal Addressing	Operand is (u + Xm)	Operand is h, i, u
0	1	Normal Addressing		
1	0	Indirect Addressing	Indirect Addressing	
1	1	Absolute Addressing (Xm = 24 Bits)	Operand is (u + Xm)	

When indirect addressing is specified, it is initiated after calculating the relative address and the absolute address in the index subsection, even if  $U \leq 177_8$ . The contents of bit positions 21 through 0 of the main storage location addressed are transferred to the control section of the CPU, where they replace the x-, h-, i-, and u-field values of the current instruction. The modified instruction is then performed just as if the whole instruction word were initially obtained in its modified form from main storage. Indexing and index register incrementation (if specified) are performed in the normal manner for both the original and the modified instruction. If the modified instruction also specifies indirect addressing, the whole process of indirect addressing is repeated. The repetition or cascading of indirect addressing continues until the modified instruction contains a 0 bit in the i-field, or contains all 0 bits in the x-field for the f, j combinations which lead to the use of i to extend the u-field, at which time indirect addressing ceases and the modified instruction is performed.

If  $f = 01_8$  through  $67_8$  (except  $07_8$ ,  $33_8$ , and  $37_8$ ),  $j = 16_8$  or  $17_8$ , and  $x = 0$  in an instruction as it is initially obtained from main storage or as it is modified as a result of an indirect addressing operation, indirect addressing does not occur even if  $i = 1$ . In this case, the i-field is used as an extension of the u-field.

#### 4.2.2.8. Description of the u-Field

The ultimate use of the u-field depends on the values in the f and j-fields of the instruction.

For most f, j combinations, u is used to form an operand address. The indexed extension of the value in the u-field of the instruction is used as the relative address of a main storage location or as the address of a GRS location.

For certain f, j combinations, the indexed extension of the value in the u-field of the instruction (or of a modified instruction in the case of indirect addressing) is used as the operand for some instructions, as a count in the case of shift instructions. For other f, j combinations, the value in the u-field has no effect on the result of the instruction.

#### 4.2.2.8.1. Use of the u-Field as an Operand Address Designator

When the value in the u-field of an instruction is an operand address designator because of the f, j combination or the specifying of indirect addressing, the 16-bit u value is extended to 18 bits with two high order zero bits to form one input to the index adder.  $X_m$  is the other input. U, the 18-bit output of the index adder, is used as the relative address of a main storage location if  $U \geq 200_8$ .

If  $U < 200_8$ , U is normally used as the absolute address of a GRS location. However, if  $U < 200_8$  and the instruction specifies indirect addressing, a jump to address, or the address for an Execute instruction (see 5.13.3), U is the relative address of a main storage location rather than the absolute address of a GRS location.

For any given u-field value, a value can be chosen for the  $X_m$  portion of the specified index register which will produce any desired value of U in the range  $000000_8$  through  $777776_8$ . (It is not possible to produce the value  $777777_8$ .)

Certain instructions use U to reference both U and  $U+1$  as a double-length (72-bit) word. In this case, U is the address of the most significant 36 bits and  $U+1$  is the address of the least significant 36 bits.

#### 4.2.2.8.2. Use of the u-Field as an Operand Designator

The value in the u-field of an instruction (or a modified instruction) is an operand ingredient rather than an operand address ingredient if indirect addressing is not specified and

- $f = 07_8$  and  $j = 14_8$
- $f = 10_8$  through  $67_8$  (except  $33_8$  and  $37_8$ ) and  $j = 16_8$  or  $17_8$ ; or
- $73_8$  and  $j = 00_8$  through  $05_8$  or  $10_8$  through  $13_8$  (all shift instructions).

When the value in the u-field of an instruction (or a modified instruction resulting from an indirect addressing sequence) is an operand designator, the 16-bit value in the u-field is extended to 18 bits to provide one of the inputs to the index adder for an indexing operation. This 18-bit value normally consists of 0 bits in the two leftmost bit positions and the 16-bit value from the u-field in the remaining bit positions. However, if  $f = 10_8$  through  $67_8$  (except  $33_8$  and  $37_8$ ),  $j = 16_8$  or  $17_8$ , and  $x = 0$ , the bits in the h and i-fields are used in the two leftmost bit positions in place of the 0 bits. When h and i are both 1 bits and they are used to extend a u-field whose value is all 1 bits, the output of the index adder is all 0 bits rather than all 1 bits.

The 18-bit index adder output is normally sent to the arithmetic section where it is extended to become a 36-bit operand by 0-bit fill ( $j = 16_8$ ) or by filling with bits identical to the leftmost bit of the index adder output ( $j = 17_8$ ).

#### 4.2.2.8.3. Use of the u-Field as a Shift Count Designator

The value in the u-field of an instruction (or a modified instruction) is a shift count designator if  $f = 73_8$  and  $j = 00$  through  $05_8$  or  $10$  through  $13_8$ . In these cases the 16-bit u-value is extended to 18 bits with high order zero bits and added to  $X_m$  to form the 18-bit value U. The appropriate low order bits of U are used as the shift count.



#### 4.2.2.8.4. Restrictions on the Use of the u-Field

When indirect addressing is not specified, certain instructions require the value in the u-field to be zero. These instructions are:

- Initiate Interprocessor Interrupt
- Enable Day Clock
- Disable Day Clock

If this restriction is violated, the results produced are undefined.



## 5. Instruction Repertoire

### 5.1. INTRODUCTION

This section describes the operation performed by each instruction in the 1100/80 user repertoire. These descriptions are grouped by types of instructions.

An introduction to each group presents information that is common to all instructions in the group. The detailed descriptions of the individual instruction have the following format:

- Instruction name                      Mnemonic code                      Octal function code
- Symbolic description of the operation performed by the instruction. The symbology used is defined in Appendix A.
- Textual description of the operation performed by the instruction.
- Sequentially numbered notes which provide special information related to the instruction, if appropriate.

For all instructions, any possible value may be used in the a-, x-, h-, i-, and u-fields unless an exception to this rule is stated in the notes. Any possible value may be used in the j-field except when j is a minor-function-code designator or when an exception is stated in the notes.

If the value of the j-field is 016 and 017 (an immediate operand specification) and the value of the x-field is zero, the h-bit, i-bit, and u-field make up the 18-bit operand. If the h- and i-bits are one and the value of the u-field is 0177777, however, the resulting operand is zero, not all ones. A negative zero can be generated as an immediate operand only by load negative instructions using x-, h-, i-, and u-fields of zero.

If the value of the a-field of the instruction is 017 (A15) and the instruction makes use of more than one arithmetic register (A+1 or A+2), those registers are located at GRS location 034 and 035, or 0174 and 0175, depending on the value of D6. If automatic index register incrementation occurs, the value of Aa or Xa are not affected. However, the value of U or U+1 (if U < 0200) or A+1 (for two pass instructions, which require both U and U+1) may be affected; if Xx is referenced as one of these operands, the updated index value is used.

## 5.2. LOAD INSTRUCTIONS

The single-precision load instructions transfer data to the arithmetic section where a 36-bit word is always formed. The 36-bit word is then transferred to the register specified by the a-field of the instruction. Single-precision data-word transfers from storage to the arithmetic section is controlled by the value in the j-field.

For the double-precision load instructions, the j-field is a minor function code and full 72-bit data transfers result.

### 5.2.1. Load A - L,LA 10

$(U) \rightarrow A$

The contents of U is transferred under j-field control to the arithmetic section and then to Aa.

### 5.2.2. Load Negative A - LN,LNA 11

$-(U) \rightarrow A$

The contents of U is transferred under j-field control to the arithmetic section. The ones complement of the value in the arithmetic section is transferred to Aa.

### 5.2.3. Load Magnitude A - LM,LMA 12

$| (U) | \rightarrow A$

The contents of U is transferred under j-field control to the arithmetic section. If the sign bit (bit 35) of the value in the arithmetic section is a 1 bit, it is complemented; if the sign bit is a 0 bit, it is not complemented. The final value (always positive) is transferred from the arithmetic section to Aa.

For j-field values 3-7 (quarter word not set) and 17 sign bit 35 controled by sign extention.

1. This instruction is the same as Load A (see 5.2.1) for  $j = H1, H2, Q1-Q4, \text{ or } S1-S6$ .

### 5.2.4. Load Negative Magnitude A - LNMA 13

$- | (U) | \rightarrow A$

The contents of U is transferred under j-field control to the arithmetic section. If the sign bit (bit 35) of the value in the arithmetic section is a 0 bit, it is complemented; if the sign bit is a 1 bit, it is not complemented. The final value (always negative) is transferred from the arithmetic section to Aa.

For j-field values 3-7 (quarter word not set) and 17 sign bit 35 controled by sign extention.

1. This instruction may be used to load  $-0$  into an A-register by using  $j = 16_g \text{ or } 17_g$ , and  $x = h = i = u = 0$ .
2. This instruction is the same as Load Negative A (see 5.2.2) for  $j = H1, H2, Q1-Q4, \text{ or } S1-S6$ .

**5.2.5. Load R - L,LR 23**

$$(U) \rightarrow R_a$$

The contents of U is transferred under j-field control to the arithmetic section and then to the Ra-register specified by the a-field.

1. If the processor is in user mode, an attempt to Load RO causes a Guard Mode interrupt.

**5.2.6. Load X Modifier - LXM 26**

$$(U) \rightarrow X_{a_{17-0}}; X_{a_{35-18}} \text{ unchanged}$$

The contents of U is transferred under j-field control to the arithmetic section; the low-order 18 bits of the value in the arithmetic section is transferred to the lower half (bits 17-0) of the X-register specified by the a-field; the upper half (bits 35 through 18) of the X-register remains unchanged.

1. This instruction loads only the low-order 18 bits of the specified X-register even if  $D7 = i = 1$  to specify 24-bit indexing.

**5.2.7. Load X - L,LX 27**

$$(U) \rightarrow X_a$$

The contents of U is transferred under j-field control to the arithmetic section and then to the X-register specified by the a-field.

**5.2.8. Load X Increment - LXI 46**

$$(U) \rightarrow X_{a_{35-18}}; X_{a_{17-0}} \text{ unchanged}$$

The contents of U is transferred under j-field control to the arithmetic section; the low-order 18 bits of the value in the arithmetic section is transferred to the upper half (bits 35-18) of the X-register specified by the a-field. The lower half (bits 17-0) of the X-register remains unchanged.

1. This instruction loads the full high-order 18 bits of the specified X-register even if  $D7 = i = 1$  to specify 24-bit indexing.

**5.2.9. Double Load A - DL  $f = 71$   $j = 13$** 

$$(U,U+1) \rightarrow A,A+1$$

The contents of U and U+1 are transferred to the arithmetic section and then to Aa and Aa+1, respectively.

**5.2.10. Double-Load Negative A - DLN 71,14**

$$-(U,U+1) \rightarrow A,A+1$$

The contents of U and U+1 are transferred to the arithmetic section where the 72-bit value is complemented and then transferred to Aa and Aa+1, respectively.

### 5.2.11. Double Load Magnitude A - DLM 71,15

$$|(U,U+1)| \rightarrow A,A+1$$

The contents of U and U+1 are transferred to the arithmetic section. If the sign bit (bit 35) of U is a 1 bit, the 72-bit value in the arithmetic section is complemented; if the sign bit is a 0 bit, the 72-bit value is not complemented. The final value (always positive) is transferred from the arithmetic section to Aa and Aa+1.

## 5.3. STORE INSTRUCTIONS

The single-precision store instructions transfer data from a control register specified by the a-field to the storage location or control register addressed by U. Exceptions to this are the Store Constant instructions. (See 5.3.5.)

Single-precision data-word transfers to storage are controlled by the j-field. If  $j = 16_8$  or  $17_8$ , no data is stored. A Guard Mode interrupt will occur, however, if  $U < 200_8$  and an Executive register is specified in user mode, or if  $U \geq 0200_8$  and a storage-limits or write-protection violation occurs.

Indexing, index incrementation/decrementation, and indirect addressing function normally in all cases.

### 5.3.1. Store A - S,SA 01

$$(A) \rightarrow U$$

The contents of Aa is transferred under j-field control to location U.

1. If  $j = 16_8$  or  $17_8$ , no data is stored.

### 5.3.2. Store Negative A - SN,SNA 02

$$-(A) \rightarrow U$$

The complement of the value of Aa is transferred under j-field control to location U.

1. If  $j = 16_8$  or  $17_8$ , no data is stored.

### 5.3.3. Store Magnitude A - SM,SMA 03

$$|(A)| \rightarrow U$$

If the sign bit (bit 35) of the value of Aa is one, the value is complemented. The final value (always positive) is transferred under j-field control to location U.

1. If  $j = 16_8$  or  $17_8$ , no data is stored.

**5.3.4. Store R - S,SR 04** $(Ra) \rightarrow U$ 

The contents of the Ra-register specified by the a-field is transferred under j-field control to location U.

1. If  $j = 16_8$  or  $17_8$ , no data is stored.

**5.3.5. Store Constant Instructions - XX 05; a = 00-07** $\text{Constant} \rightarrow U$ 

A constant value specified by the a-field is transferred under j-field control to location U. The following octal constant values may be stored:

SZ	a = 0	000000 000000	Zero
SNZ	a = 1	777777 777777	Ones
SP1	a = 2	000000 000001	Plus One
SN1	a = 3	777777 777776	Minus One
SFS	a = 4	050505 050505	Fielddata Blanks
SFZ	a = 5	606060 606060	Fielddata Zeros
SAS	a = 6	040040 040040	ASCII Blanks
SAZ	a = 7	060060 060060	ASCII Zeros

**5.3.6. Store X - S,SX 06** $(Xa) \rightarrow U$ 

The contents of the Xa-register specified by the a-field is transferred under j-field control to location U.

1. If  $j = 16_8$  or  $17_8$ , no data is stored.

**5.3.7. Double Store A - DS 71,12** $(A,A+1) \rightarrow U,U+1$ 

The contents of Aa and Aa+1 are transferred to locations U and U+1, respectively.

### 5.3.8. Block Transfer - BT 22

$(Xx + u) \rightarrow Xa + u$ , repeat  $k$  times;  $k$  = the initial count in the repeat count register

A source word is transferred under  $j$ -field control to the arithmetic section, and then under  $j$ -field control to a destination word-location. The repeat count is decreased by 1. The source-to-destination transfer step is repetitively performed until the repeat count has been decreased to 0. The  $x$ -field specifies the  $X$ -register used with the  $u$ -field to determine the effective source word-address. The  $a$ -field specifies the  $X$ -register used in determining the effective destination word-address.

1. A word containing the desired repeat count in the rightmost 18-bit positions must be loaded in the repeat count register (R1) before performing the Block Transfer instruction.
2. If the initial repeat count is  $\pm 0$ , no data is transferred. If  $-0$ , then  $+0$  is written into bits 17-0 of the repeat count.
3. If  $j = 16_8$  or  $17_8$ , no data is transferred; however, the repeat count is decreased to zero.
4. If the  $x$ -field is zero, no data is transferred. The contents of the  $X$ -register specified by the  $a$ -field remain unchanged regardless of the contents of the  $a$ - and  $h$ -fields.
5. If an interrupt occurs before the repeat count has decreased to zero, the termination pass occurs at the conclusion of the currently active data transfer. The remnant repeat count is stored in R1. When the interrupt is honored, the captured P value is the address of the Block Transfer instruction or the address of the Execute instruction which led to the Block Transfer instruction. Thus, this address can be preserved and, when the interrupt has been processed, it is possible to return to the Block Transfer instruction and continue executing this instruction at the point where it was terminated for the interrupt. If the Block Transfer instruction was entered by means of an Execute instruction, the  $h$ -field of the Execute instruction must be zero so that, when the program returns to the Execute instruction, the effective U address will again lead to the Block Transfer instruction. If the Block Transfer instruction specifies indirect addressing ( $i = 1$ ), the  $h$ -field must be zero to enable the program to return to the same effective U address and complete the Block Transfer instruction in the event of an interrupt.
6. If there is no indirect addressing ( $i = 0$ ), the  $h$ -field is normally one. If  $h = 0$ , no incrementation/decrementation of the index registers occurs. When  $h = 0$ , the source and destination addresses are the initial contents of the index registers used repetitively for every transfer performed. Thus, no more than one data transfer is effectively performed.
7. If the  $x$ -field is not zero, but the  $a$ -field is zero, the  $a$ -field references index register zero (X0), and proper operation occurs.

### 5.4. FIXED-POINT ARITHMETIC INSTRUCTIONS

The fixed-point arithmetic instructions perform integer or fractional addition, subtraction, multiplication, and division. In a single-precision arithmetic instruction, the transfer of data from location U in storage to the arithmetic section is under the control of the contents of the  $j$ -field of the instruction. For double-precision and parallel half-word and third-word arithmetic operations, the value in the  $j$ -field is a minor-function code.

For all arithmetic instructions, indexing, index incrementation/decrementation, and indirect addressing function normally.



The overflow and carry designators are set according to the results of the operation for all add and add-negative instructions except add and add-negative halves and thirds.

The sign of the result is determined by the rules of algebra except for add and add-negative instructions where both operands are zero. In this case, the result is positive zero, except for add instructions where both operands are negative zero, and add-negative instructions where the minuend (Aa) is negative zero and the subtrahend (U) is positive zero.

#### 5.4.1. Add to A - A,AA 14

$$(A) + (U) \rightarrow A$$

The contents of U is transferred under j-field control to the arithmetic section. The 36-bit value in the arithmetic section is added algebraically to the contents of Aa. The sum is stored in Aa.

#### 5.4.2. Add Negative to A - AN,ANA 15

$$(A) - (U) \rightarrow A$$

The contents of U is transferred under j-field control to the arithmetic section. The 36-bit value in the arithmetic section is subtracted algebraically from the contents of Aa. The difference is stored in Aa.

#### 5.4.3. Add Magnitude to A - AM,AMA 16

$$(A) + |(U)| \rightarrow A$$

The contents of U is transferred under j-field control to the arithmetic section. If the sign bit (bit 35) of the 36-bit value in the arithmetic is one, the value is complemented; if the sign bit is zero, the value is not complemented. The final 36-bit value in the arithmetic section (always positive) is added algebraically to the contents of Aa. The sum is stored in Aa.

Only valid for j = 3-7, 17.

1. This instruction is the same as Add To A (see 5.4.1) for j = H1, H2, Q1-Q4, or S1-S6.

#### 5.4.4. Add Negative Magnitude to A - ANM,ANMA 17

$$(A) - |(U)| \rightarrow A$$

The contents of U is transferred under j-field control to the arithmetic section. If the sign bit (bit 35) of the 36-bit value in the arithmetic section is one, the value is complemented; if the sign bit is zero, the value is not complemented. The final 36-bit value in the arithmetic section (always positive) is subtracted algebraically from the contents of Aa. The difference is stored in Aa.

Only valid for j = 3-7, 17.

1. This instruction is the same as Add Negative To A (see 5.4.2) for j = H1, H2, Q1-Q4, or S1-S6.

**5.4.5. Add Upper - AU 20**

$$(A) + (U) \rightarrow A+1$$

The contents of U is transferred under j-field control to the arithmetic section. The 36-bit value in the arithmetic section is added algebraically to the contents of Aa. The sum is stored in Aa+1. The contents of U and Aa remain unchanged.

**5.4.6. Add Negative Upper - ANU 21**

$$(A) - (U) \rightarrow A+1$$

The contents of U is transferred under j-field control to the arithmetic section. The 36-bit value in the arithmetic section is subtracted algebraically from the contents of Aa. The difference is stored in Aa+1. The contents of U and Aa remain unchanged.

**5.4.7. Add to X - A,AX 24**

$$(Xa) + (U) \rightarrow Xa$$

The contents of U is transferred under j-field control to the arithmetic section. The 36-bit value in the arithmetic section is added algebraically to the contents of the Xa-register specified by the a-field. The sum is stored in the Xa-register specified by the a-field.

**5.4.8. Add Negative to X - AN,ANX 25**

$$(Xa) - (U) \rightarrow Xa$$

The contents of U is transferred under j-field control to the arithmetic section. The 36-bit value in the arithmetic section is subtracted algebraically from the contents of the Xa-register specified by the a-field. The difference is stored in the Xa-register specified by the a-field.

**5.4.9. Multiply Integer - MI 30**

$$(A) \times (U) \rightarrow A,A+1$$

The contents of U is transferred under j-field control to the arithmetic section. The contents of Aa is multiplied algebraically by the 36-bit value in the arithmetic section, producing a 72-bit product. The most significant 36 bits of the product (including sign bits) are stored in Aa. The least significant 36 bits of the product are stored in Aa+1.

1. Bit positions 71 and 70 of the product are always sign bits. The product of any two 35-bit positive integers cannot exceed a 70-bit positive integer.

**5.4.10. Multiply Single Integer - MSI 31**

$$(A) \times (U) \rightarrow A$$

The contents of U is transferred under j-field control to the arithmetic section. The contents of Aa is multiplied algebraically by the 36-bit value in the arithmetic section, producing a 72-bit product. The least significant 36 bits of the product are stored in Aa. The most significant 36 bits of the product are lost.

1. The 36-bit result stored in Aa does not represent the product as a signed number if the leftmost 37 bits of the 72-bit product formed in the arithmetic section are not identical.

#### 5.4.11. Multiply Fractional - MF 32

$$(A) \times (U) \rightarrow A, A+1$$

The contents of U is transferred under j-field control to the arithmetic section. The contents of Aa is multiplied algebraically by the 36-bit value in the arithmetic section, producing a 72-bit product which is shifted left circularly one bit position. The leftmost 36 bits of the shifted product, including the sign bit, are stored in Aa. The rightmost 36 bits are stored in Aa+1.

1. This instruction performs an operation identical to the Multiply Integer instruction (see 5.4.9) except that the 72-bit result of the multiplication process is shifted left circularly one bit position prior to storing it in Aa and Aa+1.
2. The rightmost bit of the result in Aa+1 is a sign bit and it is identical to the leftmost bit of the result in Aa.

#### 5.4.12. Divide Integer - DI 34

$$(A, A+1) \div (U) \rightarrow A; - \text{ remainder} \rightarrow A+1$$

The contents of U is transferred under j-field control to the arithmetic section. The 72-bit signed number in Aa and Aa+1 are divided algebraically by the 36-bit value in the arithmetic section. The 36-bit signed quotient is stored in Aa. The remainder retains the sign of the dividend (the leftmost bit of the initial contents of Aa) and is stored in Aa+1.

1. The absolute value of the 72-bit signed dividend (Aa, Aa+1) should be less than the absolute value of the divisor (j-determined portion of U) multiplied by  $2^{35}$ . If this relationship is not satisfied and D20 is zero, Aa and Aa+1 are cleared to zero and D23 is set to one. If this relationship is not satisfied and D20 is one, Aa and Aa+1 remain unchanged, D23 is set to one, and a Divide Check interrupt results. This includes the case in which the divisor equals zero.

#### 5.4.13. Divide Single Fractional - DSF 35

$$(A) \div (U) \rightarrow A+1$$

The contents of U is transferred under j-field control to the arithmetic section. The contents of Aa is divided algebraically by the 36-bit value in the arithmetic section. The 36-bit signed quotient is stored in Aa+1. The remainder is lost. The contents of Aa remain unchanged.

1. The absolute value of the dividend (Aa) should be less than the absolute value of the divisor (j-determined portion of U). If this relationship is not satisfied and D20 is zero, Aa+1 is cleared to zero and D23 is set to one. If this relationship is not satisfied and D20 is one, Aa+1 remains unchanged, D23 is set to one, and a Divide Check interrupt results. This includes the case in which the divisor equals zero.
2. This instruction performs an operation like that of divide integer except that the quotient appears to be shifted one bit to the right.

**5.4.14. Divide Fractional - DF 36**

$$(A,A+1) \div (U) \rightarrow A; \text{ remainder} \rightarrow A+1$$

The contents of U is transferred under j-field control to the arithmetic section. The 72-bit signed number in Aa and Aa+1 are divided algebraically by the 36-bit value in the arithmetic section. The 36-bit signed quotient is stored in Aa. The remainder retains the sign of the dividend (the leftmost bit of the original contents of Aa) and is stored in Aa+1.

1. The absolute value of the leftmost half of the dividend (Aa) should be less than the absolute value of the divisor (j-determined portion of U). If this relationship is not satisfied and D20 is zero, Aa and Aa+1 are cleared to zero and D23 is set to one. If this relationship is not satisfied and D20 is one, Aa and Aa+1 remain unchanged, D23 is set to one, and a Divide Check interrupt results. This includes the case in which the divisor equals zero.
2. This instruction performs an operation identical to divide integer except that the quotient appears to be shifted one bit to the right.

**5.4.15. Double-Precision Fixed-Point Add - DA 71,10**

$$(A,A+1) + (U,U+1) \rightarrow A,A+1$$

The 72-bit signed number from U and U+1 are added algebraically to the 72-bit signed number from Aa and Aa+1. The 72-bit sum is stored in Aa and Aa+1.

**5.4.16. Double-Precision Fixed-Point Add Negative - DAN 71,11**

$$(A,A+1) - (U,U+1) \rightarrow A,A+1$$

The 72-bit signed number from U and U+1 are subtracted algebraically from the 72-bit signed number from Aa and Aa+1. The 72-bit difference is stored in Aa and Aa+1.

**5.4.17. Add Halves - AH 72,04**

$$(A)_{35-18} + (U)_{35-18} \rightarrow A_{35-18}$$

$$(A)_{17-0} + (U)_{17-0} \rightarrow A_{17-0}$$

The contents of each half (18-bit portion) of U is added algebraically to the contents of the corresponding half of Aa. The sums are stored in the corresponding halves of Aa.

1. There is no interaction between the upper and lower halves of the operands. A carry from bit position 17 is propagated to bit 0 rather than bit 18. A carry from bit position 35 is propagated to bit 18 rather than bit 0.

**5.4.18. Add Negative Halves - ANH 72,05**

$$(A)_{35-18} - (U)_{35-18} \rightarrow A_{35-18}$$

$$(A)_{17-0} - (U)_{17-0} \rightarrow A_{17-0}$$

The contents of each half (18-bit portion) of U is subtracted algebraically from the contents of the corresponding half of Aa. The differences are stored in the corresponding halves of Aa.

1. There is no interaction between the upper and lower halves of the operands. A borrow from bit position 17 is propagated to bit 0 rather than bit 18. A borrow from bit position 35 is propagated to bit 18 rather than bit 0.

#### 5.4.19. Add Thirds - AT 72,06

$$(A)_{35-24} + (U)_{35-24} \rightarrow A_{35-24};$$

$$(A)_{23-12} + (U)_{23-12} \rightarrow A_{23-12};$$

$$(A)_{11-0} + (U)_{11-0} \rightarrow A_{11-0}$$

The contents of each third (12-bit portion) of U is added algebraically to the contents of the corresponding third of Aa. The sums are stored in the corresponding thirds of Aa.

1. A carry from bit position 11, 23, or 35 are propagated to bit 0, 12, or 24, respectively, rather than to bit 12, 24, or 0.

#### 5.4.20. Add Negative Thirds - ANT 72,07

$$(A)_{35-24} - (U)_{35-24} \rightarrow A_{35-24};$$

$$(A)_{23-12} - (U)_{23-12} \rightarrow A_{23-12}$$

$$(A)_{11-0} - (U)_{11-0} \rightarrow A_{11-0}$$

The contents of each third (12-bit portion) of U is subtracted algebraically from the contents of the corresponding third of Aa. The differences are stored in the corresponding thirds of Aa.

1. A borrow from bit position 11, 23, or 35 are propagated to bit 0, 12, or 24, respectively, rather than to bit 12, 24, or 0.

### 5.5. FLOATING-POINT ARITHMETIC

Floating-point arithmetic operations allow for efficient computation involving numerical data with a wide range of magnitudes. Indexing, index incrementation/decrementation, and indirect addressing function normally in all floating-point arithmetic instructions.

The greatest precision is obtained in floating-point arithmetic operations when the floating-point input operands are normalized numbers. Certain floating-point operations produce undefined results if normalized input operands are not used. The supporting notes indicate which instructions are affected.

#### 5.5.1. Floating Add - FA 76,00

$$(A) + (U) \rightarrow A; \text{ residue} \rightarrow A+1 \text{ if } D17 = 1$$

The single-precision floating-point number from location U is added to the single-precision floating-point number from Aa. The resulting sum is normalized and then stored in single-precision

floating-point format in Aa. If  $D17 = 1$ , the residue in single-precision floating-point format is stored in  $Aa+1$ .

1. The result stored in Aa is a normalized number even if either or both of the input operands are not normalized. No attempt is made to normalize the residue stored in  $Aa+1$ .
2. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.
3. If the mantissa of the most significant word of the result is  $\pm 0$ , the word stored depends on D8.
4. The sign of the most significant word of the result is the sign of the large input operand. The sign of the other operand is assigned to the residue.

### 5.5.2. Floating Add Negative - FAN 76,01

$(A) - (U) \rightarrow A$ ; residue  $\rightarrow A+1$  if  $D17 = 1$

The single-precision floating-point number from location U is subtracted from the single-precision floating-point number from Aa. The resulting difference is normalized and then stored in single-precision floating-point format in Aa. If  $D17 = 1$ , the residue in single-precision floating-point format is stored in  $Aa+1$ .

1. The result stored in Aa is a normalized number even if either or both of the input operands are not normalized. No attempt is made to normalize the residue stored in  $Aa+1$ .
2. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.
3. If the mantissa of the most significant word of the result is  $\pm 0$ , the word stored depends on D8.
4. The Floating Add Negative operation is identical to the Floating Add operation described in 5.5.1 except that the ones complement of the contents of location U is used as the second operand.
5. The sign of the most significant word of the result is the sign of the large input operand. The sign of the other operand is assigned to the residue.

### 5.5.3. Double-Precision Floating Add - DFA 76,10

$(A,A+1) + (U,U+1) \rightarrow A,A+1$

The double-precision floating-point number from locations U and U+1 are added to the double-precision floating-point number from Aa and Aa+1. The resulting sum is normalized and then stored in double-precision floating-point format in Aa and Aa+1.

1. The result stored is a normalized number even if either or both of the input operands are not normalized.
2. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.
3. If the exponent value of the sum is less than -1024 and D5 and D2 are one, a Floating-Point Characteristic Underflow interrupt does not occur. Instead, +0 is stored in Aa and Aa+1. If D20 is zero, D5 is ignored.

4. If the mantissa produced is floating-point zero, the result stored is +0 regardless of the signs and characteristics of the input operands.

#### 5.5.4. Double-Precision Floating Add Negative - DFAN 76,11

$$(A,A+1) - (U,U+1) \rightarrow A,A+1$$

The double-precision floating-point number from locations U and U+1 are subtracted from the double-precision floating-point number from Aa and Aa+1. The resulting difference is normalized and then stored in double-precision floating-point format in Aa and Aa+1.

1. The result stored is a normalized number even if either or both of the input operands are not normalized.
2. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.
3. If the exponent value of the difference is less than -1024 and D5 and D20 are one, a Floating-Point Characteristic Underflow interrupt does not occur. Instead, +0 is stored in Aa and Aa+1. If D20 is one and D5 is zero, the interrupt occurs. If D20 is zero, D5 is ignored.
4. The Double-Precision Floating Add Negative operation is identical to the Double-Precision Floating Add process described in 5.5.3 except that the ones complement of the contents of U and U+1 is used as the second operand.
5. If the mantissa produced is floating-point zero, the result stored is +0, regardless of the signs and characteristics of the input operands.

#### 5.5.5. Floating Multiply - FM 76,02

$$(A) \times (U) \rightarrow A \text{ (and } A+1 \text{ if } D17 = 1)$$

The single-precision floating-point number from Aa is multiplied by the single-precision floating-point number from location U. The resulting double-length product is packed into two single-precision floating-point numbers. The most significant portion of the product in single-precision floating-point format is stored in Aa. If D17 = 1, the least significant portion of the product in single-precision floating-point format is stored in Aa+1.

1. If either or both input operands are not normalized numbers, the results are undefined. The following notes apply only if both input operands are normalized numbers.
2. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.
3. The portion of the product stored in Aa is a normalized number. No attempt is made to normalize the number stored in Aa+1.
4. The algebraic rule for signs applies to the portions of the product stored in Aa and Aa+1.
5. If the mantissa of either or both input operands is zero, the following applies:
  - a. A Floating-Point Characteristic Overflow/Underflow interrupt never occurs, regardless of the values of the characteristics of the input operands.
  - b. If D8 is zero, the result stored in Aa is +0 regardless of the signs of the input operands.

- c. If D8 is one and if the exponent value is in the range  $-128$  through  $+127$ , the most significant product-word will reflect the magnitude of the characteristic produced and the sign produced by the mantissa arithmetic.

If the exponent value of the most significant product-word is greater than  $+127$  or less than  $-128$ , the result stored in Aa is  $\pm 0$ , whichever would reflect the signs of the input operands.

6. The value of D8 has no effect on the least significant product-word. When the mantissa for the least significant product-word is zero, it is packed with the appropriate characteristic. If the characteristic of the residue is less than  $-128$ , the result stored in Aa+1 is  $\pm 0$ , whichever would reflect the signs of the operands.

A characteristic overflow of the most significant word can occur; however, the characteristic of the residue could be in the range 000 through 377. In this case, the result stored in Aa is  $\pm 0$  depending on the algebraic rule of the sign, and the residue is packed with the appropriate characteristic and stored in Aa+1.

7. If the characteristic of the number stored in Aa is greater than or equal to 27 then the characteristic of the number stored in Aa+1 is 27 less than the characteristic in Aa.

#### 5.5.6. Double-Precision Floating Multiply - DFM 76,12

$$(A,A+1) \times (U,U+1) \rightarrow A,A+1$$

The double-precision floating-point number from Aa and Aa+1 multiplied by the double-precision floating-point number from locations U and U+1. The product is normalized and stored in double-precision floating-point format in Aa and Aa+1.

1. If either or both input operands are not normalized numbers, the results are undefined. The following notes apply only if both operands are normalized numbers.
2. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.
3. The result stored in Aa and Aa+1 are always a normalized number.
4. The algebraic rule for signs applies except for the special cases covered in notes 5b and 6.
5. If the mantissa of either or both input operands are zero, the following applies:
  - a. A Floating-Point Characteristic Overflow/Underflow interrupt never occurs, regardless of the values of the characteristics of the input operands.
  - b. The result stored in Aa and Aa+1 are  $+0$  regardless of the signs of the input operands.
6. If the exponent value of the product is less than  $-1024$  and D5 and D20 are one, a Floating-Point Characteristic Underflow interrupt does not occur. Instead  $+0$ , regardless of the signs of the input operands, are stored in Aa and Aa+1. If D20 = 1 and D5 = 0, the interrupt occurs. If D20 = 0, D5 is ignored.



### 5.5.7. Floating Divide - FD 76,03

$$(A) \div (U) \rightarrow A; \text{ remainder} \rightarrow A+1 \text{ if } D17 = 1$$

The single-precision floating-point number from Aa is divided by the single-precision floating-point number from location U. The quotient is stored in Aa in single-precision floating-point format. If D17 = 1, the remainder is stored in Aa+1 in single-precision floating-point format.

1. If either or both input operands are not normalized numbers, the results are not defined. The following notes apply only if both operands are normalized numbers.
2. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.
3. If the mantissa of the divisor (U) is zero, a Divide Check interrupt occurs.
4. The quotient stored in Aa is a normalized number. No attempt is made to normalize the remainder that is stored in Aa+1 when D17 = 1.
5. The algebraic rule for signs applies to the quotient stored in Aa. The sign of the dividend is assigned to the remainder stored in Aa+1.
6. If the mantissa of the dividend (Aa) is zero but not the divisor (U), the following applies:
  - a. A Floating-Point Characteristic Overflow/Underflow interrupt never occurs, regardless of the characteristics of the operands.
  - b. If D8 = 0, the quotient stored in Aa is +0 regardless of the signs of the operands.
  - c. If D8 = 1 and the exponent value of the quotient is greater than +128 or less than -128, the quotient stored in Aa is  $\pm 0$ , whichever would reflect the signs of the input operands.
7. If the exponent value of the remainder is less than -128, the remainder stored in Aa+1 is  $\pm 0$ , whichever would reflect the sign of the dividend from Aa.
8. If the characteristic of the dividend from Aa is greater than or equal to 27, then the characteristic of the number stored in Aa+1 for the remainder is 27 or 26 less than the characteristic of the dividend.

### 5.5.8. Double-Precision Floating Divide - DFD 76,13

$$(A,A+1) \div (U,U+1) \rightarrow A,A+1$$

The double-precision floating-point number from Aa and Aa+1 are divided by the double-precision floating-point number from locations U and U+1. The quotient is stored in Aa and Aa+1 in double-precision floating-point format. The remainder is not retained.

1. If either or both of the input operands are not normalized numbers, the results are undefined. The following notes apply only if both operands are normalized numbers.
2. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.
3. If the mantissa of the divisor is zero, a Divide Check interrupt occurs.
4. The result stored in Aa and Aa+1 are always a normalized number.

5. The algebraic rule for signs applies except for the special cases explained in notes 6b and 7.
6. If the dividend mantissa ( $Aa, Aa+1$ ) is zero and the divisor mantissa ( $U, U+1$ ) is not zero, the following applies:
  - a. A Floating-Point Characteristic Overflow/Underflow interrupt never occurs regardless of the values of the characteristics of the input operands.
  - b. The result stored in  $Aa$  and  $Aa+1$  are  $+0$  regardless of the signs of the operands.
7. If the exponent value of the quotient is less than  $-1024$ , and  $D5$  and  $D20$  are one, a Floating-Point Characteristic Underflow interrupt does not occur. Instead  $+0$ , regardless of the signs of the input operands, are stored in  $Aa$  and  $Aa+1$ . If  $D20 = 1$  and  $D5 = 0$ , the interrupt occurs. If  $D20 = 0$ ,  $D5$  is ignored.

#### 5.5.9. Load and Unpack Floating - LUF 76,04

$| (U)_{34-27} \rightarrow A_{7-0}$ , zero fill;

$(U)_{26-0} \rightarrow A+1_{26-0}$ , sign fill

The single-precision floating-point number from location  $U$  is transferred to the arithmetic section and unpacked. The absolute value of the biased characteristic of the input operand is transferred to bits 7 through 0 of  $Aa$ ; bits 35 through 8 of the  $Aa$  is filled with 0 bits. The mantissa of the input operand is transferred to bits 26 through 0 of  $Aa+1$ ; bits 35 through 27 of  $Aa+1$  are filled with bits identical to the sign of the floating-point number in  $U$ .

1. No attempt is made to normalize the operand.

#### 5.5.10. Double Load and Unpack Floating - DFU 76,14

$| (U, U+1)_{70-60} \rightarrow A_{10-0}$ , zero fill;

$(U, U+1)_{59-36} \rightarrow A+1_{23-0}$ , sign fill;

$(U, U+1)_{35-0} \rightarrow A+2$

The double-precision floating-point number from locations  $U$  and  $U+1$  are transferred to the arithmetic section and unpacked. The absolute value of the biased characteristic of the input operand is transferred to bits 10 through 0 of  $Aa$ ; bits 35 through 11 of  $Aa$  are filled with 0 bits. The leftmost 24 bits of the mantissa,  $(U)_{23-0}$ , are transferred to bits 23 through 0 of  $Aa+1$ ; bits 35 through 24 of  $Aa+1$  are filled with bits identical to the sign of the floating-point number in locations  $U$  and  $U+1$ . The rightmost 36 bits of the mantissas ( $U+1$ ) are transferred to  $Aa+2$ .

1. No attempt is made to normalize the operand.

#### 5.5.11. Load and Convert to Floating - LCF 76,05

$(U)_{35} \rightarrow A+1_{35}$ ;  $[\text{normalized } (U)]_{26-0} \rightarrow A+1_{26-0}$ ;

if  $(U)_{35} = 0$ :  $(A)_{7-0} \pm \text{normalizing count} \rightarrow A+1_{34-27}$ ;

if  $(U)_{35} = 1$ : ones complement of  $[(A)_{7-0} \pm \text{normalizing count}] \rightarrow A+1_{34-27}$

The fixed-point number from location U is sent to the arithmetic section where it is shifted right or left, as required, to normalize it. The normalizing shift count is added to the characteristic from the rightmost eight bits of Aa if a normalizing right-shift is required. It is subtracted from the characteristic if a normalizing left-shift is required. The adjusted characteristic (complemented if U is negative) is packed with the normalized value from U to form a single-precision floating-point number. The packed result is stored in Aa+1. The contents of Aa remain unchanged.

1. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.
2. The 28 leftmost bits from Aa are ignored:  $(Aa)_{7-0}$  must be prebiased.
3. If the resultant mantissa is zero, the following applies:
  - a. If  $D8=0$ , the result stored in Aa is +0.
  - b. If  $D8=1$ , and the resultant characteristic is in the range 000 through 377, the characteristic is packed with the zero mantissa and stored in Aa.
  - c. If  $D8=1$ , and the resultant characteristic is a negative number,  $\pm 0$  is stored in Aa depending on the sign of the input operand.

#### 5.5.12. Double Load and Convert to Floating - DFP, DLCF 76,15

$(U)_{35} \rightarrow A+1_{35}$ ;  $[\text{normalized } (U, U+1)]_{59-0} \rightarrow A+1_{23-0}$  and  $A+2$ ;

if  $(U)_{35} = 0$ :  $(A)_{10-0} \pm \text{normalizing count} \rightarrow A+1_{34-24}$ ;

if  $(U)_{35} = 1$ : ones complement of  $[(A)_{10-0} \pm \text{normalizing count}] \rightarrow A+1_{34-24}$

The double-precision fixed-point number from locations U and U+1 are sent to the arithmetic section where it is shifted right or left, if necessary, to normalize it. The normalizing shift count is added to the characteristic from the rightmost 11 bits of Aa if a normalizing right-shift is required. It is subtracted from the characteristic if a normalizing left-shift is required. The adjusted characteristic (complemented if U is negative) is packed with the normalized value from U and U+1 to form a double-precision floating-point number and the packed result is stored in Aa+1 and Aa+2. The contents of Aa remain unchanged.

1. A Floating-Point Characteristic Overflow/Underflow interrupt may occur.
2. The 25 leftmost bits from Aa are ignored;  $(Aa)_{10-0}$  must be prebiased.
3. If the 72-bit input operand from U and U+1 are  $\pm$ , the result stored is +0 regardless of the sign of the 72-bit operand.
4. If the adjusted characteristic represents a negative number when  $D20$  and  $D5 = 1$ , a Floating-Point Characteristic Underflow Interrupt does not occur. Instead +0, regardless of the sign of the 72-bit operand, is stored.

**5.5.13. Floating Expand and Load - FEL 76,16**

If  $(U)_{35} = 0$ :  $(U)_{35-27} + 1600_8 \rightarrow A_{35-24}$ ;

if  $(U)_{35} = 1$ :  $(U)_{35-27} - 1600_8 \rightarrow A_{35-24}$ ;

$(U)_{26-3} \rightarrow A_{23-0}$ ;

$(U)_{2-0} \rightarrow A + 1_{35-33}$ ;

$(U)_{35} \rightarrow A + 1_{32-0}$

The single-precision floating-point input operand from location U is transferred to the arithmetic section. The three fields of the operand are expanded to form a double-precision floating-point number as follows:

- a. The sign bit is stored in bits 71 and 32 through 0.
- b. The 8-bit characteristic which includes a bias of  $200_8$  is modified to an 11-bit characteristic which includes a bias of  $2000_8$  and it is stored in bits 70 through 60.
- c. The 27-bit mantissa is stored in bits 59 through 33.

The result is transferred to Aa and Aa+1.

1. If the operand is not in the normalized single-precision floating-point format, the results stored are undefined. The following notes apply only if the input operand is a normalized number.
2. If the mantissa of the input operand is  $\pm 0$ , the result stored in Aa and Aa+1 are +0 regardless of the sign of the operand.
3. A Floating-Point Characteristic Overflow/Underflow interrupt will not occur as a result of this instruction.

**5.5.14. Floating Compress and Load - FCL 76,17**

If  $(U)_{35} = 0$ :  $(U)_{35-24} - 1600_8 \rightarrow A_{35-27}$ ;

if  $(U)_{35} = 1$ :  $(U)_{35-24} + 1600_8 \rightarrow A_{35-27}$ ;

$(U)_{23-0} \rightarrow A_{26-3}$ ;

$(U+1)_{35-33} \rightarrow A_{2-0}$

The double precision floating-point operand from locations U and U+1 are transferred to the arithmetic section. The three fields of the operand are compressed to form a single-precision floating-point number as follows:

- a. The sign bit is stored in bit 35.
- b. The 11-bit characteristic which includes a bias of  $2000_8$  is modified to an 8-bit characteristic which includes a bias of  $200_8$  and it is stored in bits 34 through 27.
- c. The 27 leftmost bits of the mantissa (bits 23 through 0 from location U and bits 35 through 33 from location U+1) are stored in bits 26 through 0.

The result is transferred to Aa.

1. The following notes apply only if the operand is a normalized number.
2. If  $D20 = 1$ , a Floating-Point Characteristic Overflow interrupt occurs if the characteristic of the operand is greater than  $+127$ , and a Floating-Point Characteristic Underflow interrupt occurs if the characteristic of the operand is less than  $-128$ .  $D21$  is set when an underflow condition is detected, and  $D22$  is set when an overflow condition is detected.
3. The contents of  $U+1_{32-0}$  are ignored.
4. If the operand is not a normalized number or is equal to  $\pm 0$ , the result stored in Aa is  $+0$  regardless of the characteristic of the input operand.

#### 5.5.15. Magnitude of Characteristic Difference to Upper - MCDU 76,06

$$\left| | (A)_{35-27} - | (U)_{35-27} \right| \rightarrow A+1_{8-0}; \text{ zeros} \rightarrow A+1_{35-9}$$

The absolute value of the characteristic of the single-precision floating-point number from location U is subtracted from the absolute value of the characteristic of the single-precision floating-point number from Aa.

The absolute value of the 9-bit difference is stored in bits 8 through 0 of  $Aa+1$ . Bits 35 through 9 of  $Aa+1$  are zero filled. The contents of Aa is not changed.

1. The mantissas from location U and from Aa are ignored.

#### 5.5.16. Characteristic Difference to Upper - CDU 76,07

$$| (A)_{35-27} - | (U)_{35-27} \rightarrow A+1_{8-0}; \text{ sign bits to } A+1_{35-9}$$

The absolute value of the characteristic of the single-precision floating-point number from location U is subtracted from the absolute value of the characteristic of the single-precision floating-point number from Aa. The 9-bit signed difference is stored in bits 8 through 0 of  $Aa+1$ . Bits 35 through 9 of the  $Aa+1$  are filled with bits identical to the sign of the difference. The contents of Aa is not changed.

1. The mantissas from location U and from Aa are ignored.

### 5.6. SEARCH AND MASKED-SEARCH INSTRUCTIONS

There are six search instructions, each of which compares the contents of either one or two A-registers with the contents of storage locations or control registers. There are eight masked-search instructions, each of which compares contents of predefined bit positions of either one or two A-registers with the contents of the corresponding bit positions of storage locations or control registers.

These are all multistage instructions. The various stages required to perform these instructions are as follows:

- An initial stage
- Repeated test stages (any number from 0 to 262,143)
- Termination stage

If indirect addressing is specified, it proceeds prior to initiation of the first test stage.

The initial stage prepares the control section and the arithmetic section for the test stages. The following steps are performed during the initial stage:

- The P-register is incremented:  $(P) + 1 \rightarrow P$
- The contents of the repeat count register (R1) is transferred to the index subsection.
- The contents of the specified A-registers are transferred to the arithmetic section.
- The contents of the mask register (R2) is transferred to the arithmetic section for a masked-search instruction.

These steps are performed only during the initial stage and are not repeated during the test stages.

The rightmost 18 bit positions of R1 contain the repeat count; that is, the maximum number of test stages to be performed. R1 must be loaded with the desired repeat count prior to initiating a search or masked search instruction. If the initial repeat-count is +0, the termination stage is initiated immediately following the completion of the initial stage; there are no test stages. If the repeat count is not zero, a series of one or more test stages is initiated.

During each test stage, the value U is formed in the index subsection. For the search instructions, an input operand is transferred to the arithmetic section under j-field control. The inputs to the test process are the values obtained using the effective U address and the A-register or registers specified by the instruction.

For the masked-search instructions, the contents of the j-field is a minor-function code. The inputs to the test process are:

- the logical product of the mask from R2 and the input operand addressed by U
- the logical products of the mask and the specified A-registers.

Each bit of the logical product is the logical product of the contents of corresponding bit positions of the two words. The logical product of two bits gives the same results as the Logical **AND**.

The search and masked-search instructions include algebraic and alphanumeric comparisons. During an algebraic comparison, the leftmost bit of each of the 36-bit values are considered to be a sign bit; a positive number is always recognized as being greater than a negative number. During an alphanumeric comparison, the leftmost bit of each of the 36-bit values are considered to be a numeric bit rather than a sign bit.

If the test process shows that the specified conditions are met, the repeat count is decreased by one and the termination stage is initiated. If the specified conditions are not met, the repeat count is decreased by one and examined. If the decreased repeat count is zero, the termination stage is initiated. If the decreased repeat count is not zero, another test stage is normally initiated. It should be noted that if  $x = 0$ ,  $X_i = 0$ , or  $h = 0$ , the same value for U will be formed in each test stage.

As previously indicated, the termination stage is initiated if the initial repeat count is zero, if the repeat count is decreased from one to zero during the test stage, or if the conditions specified by the search or masked-search instruction are detected during a test stage. If an interrupt is detected during either an initial stage or one of the test stages, the termination stage is initiated immediately following that stage. The P-register is reset and the repeat count is stored so that the search instruction can be resumed when the interrupt condition has been satisfied.

The termination stage is used to transfer the current repeat count from the index subsection to the rightmost 18-bit positions of R1. The contents of the P-register may or may not be changed during the termination stage, as follows:

- If the termination stage is entered as a result of detecting that the initial repeat count is zero during the initial stage or detecting that the decreased repeat count is zero during a test stage for which the specified conditions are not detected (no find), the contents of the P-register are not changed during the termination stage. The P-register contains the address of the instruction following the search or masked-search instruction, or the address of the instruction following the Execute instruction which referenced the search or masked-search instruction (next instruction condition).
- If the termination stage is entered as a result of detecting the specified conditions during a test stage (a find has been made), the P-register is incremented during the termination stage:  $(P) + 1 \rightarrow P$ . Since the P-register was also incremented during the initial stage, it now contains the address of the search or masked-search instruction (or the address of the Execute instruction which referenced it) + 2 (skip next instruction condition).
- If the termination stage is entered as a result of recognizing an interrupt, the P-register is decreased by 1 during the termination stage:  $(P) - 1 \rightarrow P$ . This decrease offsets the incrementation of the P-register performed during the initial stage; the P-register now contains the address of the search or masked-search instruction, or the address of the Execute instruction which referenced it. This address can be preserved so that when the interrupt condition has been satisfied, the search or masked-search can be resumed at the point where it was terminated for the interrupt. If the search or masked-search instruction is entered by means of an Execute instruction, the h-field of the Execute instruction should be zero (that is, no incrementation) so that when the program returns to the Execute instruction after an interrupt, the effective U address will again lead to the search or masked-search instruction.

If the search or masked-search instruction specifies indirect addressing (i-field = 1), the h-field should be zero to enable the program to return to the same effective U address and resume the search or masked search after an interrupt.

For equality searches (SE, SNE, MSE, MSNE), +0 does not equal -0; for arithmetic searches (SLE, SG, SW, SNW, MSLE, MSE, MSW, MSNW), +0 is greater than -0; for alphanumeric searches (MASL, MASG), -0 is greater than +0.

When a search or masked-search is resumed after an interrupt, the initial stage is again performed to prepare the control section for the remaining test stages and to transfer the contents of the specified A-register to the arithmetic section for the comparisons performed in the test stages. When  $h = 1$  (that is, index register incrementation is specified), if the a- and x-fields reference the same control register, the contents of that register will have been altered by the index incrementation which occurred before the search or masked search was interrupted. As a result, when the search or masked search is resumed, the value referenced by the a-field to be used in the test stages are no longer the original test value used before the interrupt occurred. Therefore, when  $h = 1$ , the a-field and x-field should not specify the same control register so that the search or masked-search instruction can be resumed in the event of an interrupt.

### 5.6.1. Search Equal – SE 62

Skip NI if  $(U) = (A)$ , else repeat

During the initial stage, the contents of the repeat count register (R1) are transferred to the index subsection, the contents of Aa is transferred to the arithmetic section, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section under j-field control. This value from U is compared with the value from Aa and:

- If  $(U) = (Aa)$ , the termination stage is initiated. This stage stores the remnant repeat count and increments the P-register. (Skip to NI.)
- If  $(U) \neq (Aa)$  and the repeat count is not zero, another test stage is initiated.
- If  $(U) \neq (Aa)$  and the repeat count is zero, the termination stage stores zero as the remnant repeat-count and the P-register is not incremented.

1.  $+0$  is not equal to  $-0$ .

### 5.6.2. Search Not Equal – SNE 63

Skip NI if  $(U) \neq (A)$ , else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa is transferred to the arithmetic section, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section under j-field control. The value from U is compared with the value from Aa and:

- If  $(U) \neq (Aa)$ , the termination stage is initiated. The termination stage stores the remnant repeat count and increments the P-register. (Skip NI.)
- If  $(U) = (Aa)$  and the repeat count is not zero, another test stage is initiated.
- If  $(U) = (Aa)$  and the repeat count is zero, the termination stage is initiated. The termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1.  $+0$  is not equal to  $-0$ .

### 5.6.3. Search Less Than or Equal – Search Not Greater – SLE,SNG 64

Skip NI if  $(U) \leq (A)$ , else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa transferred to the arithmetic section, and the P-register is incremented.



If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section under j-field control. The value from U is compared with the value from Aa and:

- If  $(U) \leq (Aa)$ , the termination stage is initiated. The termination stage stores the remnant repeat count and increments the P-register. (Skip NI.)
- If  $(U) > (Aa)$  and the repeat count is not zero, another test stage is initiated.
- If  $(U) > (Aa)$  and the repeat count is zero, the termination stage is initiated. The termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1.  $+0$  is greater than  $-0$ .

#### 5.6.4. Search Greater – SG 65

Skip NI if  $(U) > (A)$ , else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa is transferred to the arithmetic section, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section under j-field control. The value from U is compared with the value from Aa and:

- If  $(U) > (Aa)$ , the termination stage is initiated. The termination stage stores the remnant repeat count and increments the P-register. (Skip NI.)
- If  $(U) \leq (Aa)$  and the repeat count is not zero, another test stage is initiated.
- If  $(U) \leq (Aa)$  and the repeat count is zero, the termination stage is initiated. The termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1.  $+0$  is greater than  $-0$ .

#### 5.6.5. Search Within Range – SW 66

Skip NI if  $(A) < (U) \leq (A+1)$ , else repeat

During the initial stage the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa and Aa+1 are transferred to the arithmetic section, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section under j-field control. The value from U is compared with the value from Aa and:

- If  $(U) > (Aa)$  and  $(U) \leq (Aa + 1)$ , the termination stage is initiated. The termination stage stores the remnant repeat count and increments the P-register. (Skip NI.)
- If  $(U) \leq (Aa)$  or  $(U) > (Aa + 1)$ , and the repeat count is not zero, another test stage is initiated.
- If  $(U) \leq (Aa)$  or  $(U) > (Aa + 1)$ , and the repeat count is zero, the termination stage is initiated. The termination stage stores zero as the remnant repeat count and the P-register is not incremented.
  1. +0 is greater than -0.
  2. Normally,  $(Aa) < (Aa + 1)$ . However, if  $(Aa) \geq (Aa + 1)$ , there is no value from U which can satisfy the conditions  $(Aa) < (U) \leq (Aa + 1)$ .

#### 5.6.6. Search Not Within Range - SNW 67

Skip NI if  $(U) \leq (A)$  or  $(U) > (A + 1)$ , else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa and Aa+1 are transferred to the arithmetic section, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section under j-field control. The value from U is compared with the value from Aa and:

- If  $(U) \leq (Aa)$  or  $(U) > (Aa + 1)$ , the termination stage is initiated. The termination stage stores the remnant repeat count and increments the P-register. (Skip NI.)
- If  $(U) > (Aa)$  and  $(U) \leq (Aa + 1)$ , and the repeat count is not zero, another test stage is initiated.
- If  $(U) > (Aa)$  and  $(U) \leq (Aa + 1)$ , and the repeat count is zero, the termination stage is initiated. The termination stage stores zero as the remnant repeat count and the P-register is not incremented.
  1. Normally,  $(Aa) < (Aa + 1)$ . If, however,  $(Aa) \geq (Aa + 1)$ , there is no value from U which will not satisfy the conditions  $(U) \leq (Aa)$  or  $(U) > (Aa + 1)$ .
  2. +0 is greater than -0.

#### 5.6.7. Mask Search Equal - MSE 71,00

Skip NI if  $(U) \text{ AND } (R2) = (A) \text{ AND } (R2)$ , else repeat

During the initial stage, the contents of the Repeat Count Register (R1) is transferred to the index subsection, the contents of Aa and R2 are transferred to the arithmetic section, the logical products of the values from Aa and R2 are formed, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section.  $(U) \text{ AND } (R2)$  are compared to  $(Aa) \text{ AND } (R2)$  and:

- If  $(U) \text{ AND } (R2) = (Aa) \text{ AND } (R2)$ , the termination stage is initiated. This stage stores the remnant repeat count and increments the P-register (skip NI).
- If  $(U) \text{ AND } (R2) \neq (Aa) \text{ AND } (R2)$  and the repeat count is not zero, another test stage is initiated.
- If  $(U) \text{ AND } (R2) \neq (Aa) \text{ AND } (R2)$  and the repeat count is zero, the termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1.  $+0$  is not equal to  $-0$ .

#### 5.6.8. Mask Search Not Equal – MSNE 71,01

Skip NI if  $(U) \text{ AND } (R2) \neq (A) \text{ AND } (R2)$ , else repeat.

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa and R2 are transferred to the arithmetic section, the logical products of the values from Aa and R2 are formed, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section.  $(U) \text{ AND } (R2)$  are compared to  $(Aa) \text{ AND } (R2)$  and:

- If  $(U) \text{ AND } (R2) \neq (Aa) \text{ AND } (R2)$ , the termination stage is initiated. This stage stores the remnant repeat count and increments the P-register (skip NI).
- If  $(U) \text{ AND } (R2) = (Aa) \text{ AND } (R2)$  and the repeat count is not zero, another test stage is initiated.
- If  $(U) \text{ AND } (R2) = (Aa) \text{ AND } (R2)$  and the repeat count is zero, the termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1.  $+0$  is not equal to  $-0$ .

#### 5.6.9. Mask Search Less Than or Equal – Mask Search Not Greater – MSLE,MSNG 71,02

Skip NI if  $(U) \text{ AND } (R2) \leq (A) \text{ AND } (R2)$ , else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa and R2 are transferred to the arithmetic section, the logical products of the values from Aa and R2 are formed, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section.  $(U) \text{ AND } (R2)$  are compared to  $(Aa) \text{ AND } (R2)$  and:

- If  $(U) \text{ AND } (R2) \leq (Aa) \text{ AND } (R2)$ , the termination stage is initiated. This stage stores the remnant repeat count and increments the P-register (skip NI).
- If  $(U) \text{ AND } (R2) > (Aa) \text{ AND } (R2)$  and the repeat count is not zero, another test stage is initiated.

- If  $(U) \text{ AND } (R2) > (Aa) \text{ AND } (R2)$  and the repeat count is zero, the termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1. +0 is greater than -0.

#### 5.6.10. Mask Search Greater – MSG 71,03

Skip NI if  $(U) \text{ AND } (R2) > (A) \text{ AND } (R2)$ , else repeat

During the initial stage, the contents of the repeat count register(R1) is transferred to the index subsection, the contents of Aa and R2 are transferred to the arithmetic section, the logical products of the values from Aa and R2 are formed, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section.  $(U) \text{ AND } (R2)$  are compared to  $(Aa) \text{ AND } (R2)$  and:

- If  $(U) \text{ AND } (R2) > (Aa) \text{ AND } (R2)$ , the termination stage is initiated. This stage stores the remnant repeat count and increments the P-register. (Skip NI.)
- If  $(U) \text{ AND } (R2) \leq (Aa) \text{ AND } (R2)$  and the repeat count is not zero, another test stage is initiated.
- If  $(U) \text{ AND } (R2) \leq (Aa) \text{ AND } (R2)$  and the repeat count is zero, the termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1. +0 is greater than -0.

#### 5.6.11. Masked Search Within Range – MSW 71,04

Skip NI if  $(A) \text{ AND } (R2) < (U) \text{ AND } (R2) \leq (A+1) \text{ AND } (R2)$ , else repeat.

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa, Aa+1, and R2 are transferred to the arithmetic section, the logical products of the values from Aa and R2 and the values from Aa+1 and R2 are formed, and the P-register is incremented. If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section. The logical products are compared and:

- If  $(U) \text{ AND } (R2) > (Aa) \text{ AND } (R2)$  and  $(U) \text{ AND } (R2) \leq (Aa+1) \text{ AND } (R2)$  the termination stage is initiated. This stage stores the remnant repeat count and increments the P-register. (Skip NI.)
- If  $(U) \text{ AND } (R2) \leq (Aa) \text{ AND } (R2)$  or  $(U) \text{ AND } (R2) > (Aa+1) \text{ AND } (R2)$  and the repeat count is not zero, another test stage is initiated.
- If  $(U) \text{ AND } (R2) \leq (Aa) \text{ AND } (R2)$  or  $(U) \text{ AND } (R2) > (Aa+1) \text{ AND } (R2)$  and the repeat count is zero, the termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1. Normally,  $(Aa) \text{ AND } (R2) < (Aa+1) \text{ AND } (R2)$ . If, however,  $(Aa) \text{ AND } (R2) \geq (Aa+1) \text{ AND } (R2)$ , no possible value of U will satisfy the search condition.

2. +0 is greater than -0.

### 5.6.12. Masked Search Not Within Range – MSNW 71,05

Skip NI if  $(U) \text{ AND } (R2) \leq (A) \text{ AND } (R2)$  or  $(U) \text{ AND } (R2) > (A+1) \text{ AND } (R2)$ , else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa and R2 are transferred to the arithmetic section, the logical products of the values from Aa and R2 and the values from Aa+1 and R2 are formed, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section. The logical products are compared and:

- If  $(U) \text{ AND } (R2) \leq (Aa) \text{ AND } (R2)$  or  $(U) \text{ AND } (R2) > (Aa+1) \text{ AND } (R2)$  the termination stage is initiated. This stage stores the remnant repeat count and increments the P-register. (Skip NI.)
- If  $(U) \text{ AND } (R2) > (Aa) \text{ AND } (R2)$  and  $(U) \text{ AND } (R2) \leq (Aa+1) \text{ AND } (R2)$  and the repeat count is not zero, another test stage is initiated.
- If  $(U) \text{ AND } (R2) > (Aa) \text{ AND } (R2)$  and  $(U) \text{ AND } (R2) \leq (Aa-1) \text{ AND } (R2)$  and the repeat count is zero, the termination stage stores zero as the remnant repeat count and the P-register is not incremented.

1. Normally,  $(Aa) \text{ AND } (R2) < (Aa+1) \text{ AND } (R2)$ . If, however,  $(Aa) \text{ AND } (R2) \geq (Aa+1) \text{ AND } (R2)$ , every possible value of U will satisfy at least one of the following conditions:

$$(U) \text{ AND } (R2) \leq (Aa) \text{ AND } (R2)$$

$$(U) \text{ AND } (R2) > (Aa+1) \text{ AND } (R2)$$

2. +0 is greater than -0.

### 5.6.13. Masked Alphanumeric Search Less Than or Equal – MASL 71,06

Skip NI if  $(U) \text{ AND } (R2) \leq (A) \text{ AND } (R2)$ , else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa and R2 are transferred to the arithmetic section, the logical products of the values from Aa and R2 are formed, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section.  $(U) \text{ AND } (R2)$  are compared alphanumerically to  $(Aa) \text{ AND } (R2)$ , and:

- If  $(U) \text{ AND } (R2) \leq (Aa) \text{ AND } (R2)$ , the termination stage is initiated. This stage stores the remnant repeat count and increments the P-register. (Skip NI.)

- If  $(U) \text{ AND } (R2) > (Aa) \text{ AND } (R2)$  and the repeat count is not zero, another test stage is initiated.
- If  $(U) \text{ AND } (R2) > (Aa) \text{ AND } (R2)$  and the repeat count is zero, the termination stage stores zero as the remnant repeat count and the P-register is not incremented.
  1. -0 is greater than +0.

#### 5.6.14. Masked Alphanumeric Search Greater – MASG 71,07

Skip NI if  $(U) \text{ AND } (R2) > (A) \text{ AND } (R2)$ , else repeat

During the initial stage, the contents of the repeat count register (R1) is transferred to the index subsection, the contents of Aa and R2 are transferred to the arithmetic section, the logical products of the value from Aa and R2 are formed, and the P-register is incremented.

If the initial repeat count is zero, the next instruction (NI) is initiated.

If the initial repeat count is not zero, the first test stage is initiated. During each test stage, the repeat count is decreased and the contents of U is transferred to the arithmetic section.  $(U) \text{ AND } (R2)$  are compared alphanumerically to  $(Aa) \text{ AND } (R2)$ , and:

- If  $(U) \text{ AND } (R2) > (Aa) \text{ AND } (R2)$ , the termination stage is initiated. This stage stores the remnant repeat count and increments the P-register (skip NI).
- If  $(U) \text{ AND } (R2) \leq (Aa) \text{ AND } (R2)$  and the repeat count is not zero, another test stage is initiated.
- If  $(U) \text{ AND } (R2) \leq (Aa) \text{ AND } (R2)$  and the repeat count is zero, the termination stage stores zero as the remnant repeat count and the P-register is not incremented.
  1. -0 is greater than +0.

### 5.7. TEST (OR SKIP) INSTRUCTIONS

Test instructions are used to read one or more words from storage or control registers and test for certain conditions. The result of the test is used to determine whether the instruction addressed by the incremented contents of the P-register (next instruction) should be performed or skipped.

The next instruction (NI) is always read from storage. If the decision is made to skip NI, it is discarded, the P-register is incremented a second time, and the contents of the P-register is then used to address the following instruction.

Indirect addressing, indexing, and index register incrementation/decrementation operate normally.

#### 5.7.1. Test Even Parity – TEP 44

Skip NI if  $(U) \text{ AND } (A)$  has even parity.

The value from U is transferred to the arithmetic section under j-field control, where it is used with the contents of Aa to form a 36-bit logical product.

If  $(U) \text{ AND } (Aa)$  has an even number of 1 bits, the next instruction (NI) is skipped and the instruction following NI is performed.

If (U)  $\overline{\text{AND}}$  (Aa) has an odd number of 1 bits, NI is performed.

### 5.7.2. Test Odd Parity – TOP 45

Skip NI if (A)  $\overline{\text{AND}}$  (U) has odd parity.

The contents of U is transferred to the arithmetic section under j-field control, where they are used with the contents of Aa to form a 36-bit logical product.

If (U)  $\overline{\text{AND}}$  (Aa) has an odd number of 1 bits, the next instruction (NI) is skipped and the instruction following NI is performed.

If (U)  $\overline{\text{AND}}$  (Aa) has an even number of 1 bits, NI is performed.

### 5.7.3. Test Less Than or Equal to Modifier – TLEM 47

Test Not Greater Than Modifier – TNGM

Skip NI if  $(U)_{17-0} \leq (Xa)_{17-0}$ ; always  $(Xa)_{17-0} + (Xa)_{35-18} \rightarrow Xa_{17-0}$

The contents of U is transferred to the arithmetic section under j-field control. The contents of the index register addressed by the a-field (Xa) is transferred to the arithmetic section. The rightmost 18 bits of the value from U is subtracted from the rightmost 18 bits of the value from Xa (this is performed as if the leftmost 18 bits of each operand were zeros).

If  $(U)_{17-0} \leq (Xa)_{17-0}$  (the sign of the difference is positive), the next instruction is skipped and the instruction following NI is performed.

If  $(U)_{17-0} > (Xa)_{17-0}$  (the sign of the difference is negative), NI is performed.

In either case, the leftmost 18 bits from Xa are added to the rightmost 18 bits from Xa, and the sum is stored in the rightmost 18 bit positions of Xa. The leftmost 18 bit positions of Xa are not changed.

1. If  $a = 0$ , index register zero (XO) is referenced.
2.  $+0$  is less than  $-0$ .
3. Both  $Xa_{17-0}$  and the value from U is considered to be 18-bit numeric values with a positive sign implied.
4. Only the rightmost 18 bits of the value from U are involved in the operation. Values of 0, 1, or 3 in the j-field yield the same results. Values of  $16_8$  or  $17_8$  in the j-field yield the same result.
5. If  $h = 1$  and  $a = x$ , the specified index register is incremented or modified only once.

### 5.7.4. Test Zero – TZ 50

Skip NI if  $(U) = \pm 0$ .

The contents of U is transferred to the arithmetic section under j-field control.

If the value transferred is  $\pm 0$ , the next instruction is skipped and the instruction following NI is performed.

If the value transferred is not  $\neq 0$ , NI is performed.

1. The contents of the a-field are ignored.

#### 5.7.5. Test Nonzero - TNZ 51

Skip NI if  $(U) \neq 0$ .

The contents of U is transferred to the arithmetic section under j-field control.

If the value transferred is not  $\neq 0$ , the next instruction is skipped and the instruction following NI is performed.

If the value transferred is  $\neq 0$ , NI is performed.

1. The contents of the a-field are ignored.

#### 5.7.6. Test Equal - TE 52

Skip NI if  $(U) = (A)$ .

The contents of U is transferred to the arithmetic section under j-field control. The contents of Aa is also transferred to the arithmetic section.

If  $(U) = (Aa)$ , the next instruction is skipped and the instruction following NI is performed.

If  $(U) \neq (Aa)$ , NI is performed.

1.  $+0$  is not equal to  $-0$ .

#### 5.7.7. Test Not Equal - TNE 53

Skip NI if  $(U) \neq (A)$ .

The contents of U is transferred to the arithmetic section under j-field control. The contents of Aa is also transferred to the arithmetic section.

If  $(U) \neq (Aa)$ , the next instruction is skipped and the instruction following NI is performed.

If  $(U) = (Aa)$ , NI is performed.

1.  $+0$  is not equal to  $-0$ .

#### 5.7.8. Test Less Than or Equal - Test Not Greater - TLE,TNG 54

Skip NI if  $(U) \leq (A)$ .

The contents of U is transferred to the arithmetic section under j-field control. The contents of Aa is also transferred to the arithmetic section.

If  $(U) \leq (Aa)$ , the next instruction is skipped and the instruction following NI is performed.



If  $(U) > (Aa)$ , NI is performed.

1.  $+0$  is greater than  $-0$ .

#### 5.7.9. Test Greater - TG 55

Skip NI if  $(U) > (A)$ .

The contents of U is transferred to the arithmetic section under j-field control. The contents of Aa is also transferred to the arithmetic section.

If  $(U) > (Aa)$ , the next instruction is skipped and the instruction following NI is performed.

If  $(U) \leq (Aa)$ , NI is performed.

1.  $+0$  is greater than  $-0$ .

#### 5.7.10. Test Within Range - TW 56

Skip NI if  $(A) < (U) \leq (A+1)$ .

The contents of U is transferred to the arithmetic section under j-field control. The contents of Aa and Aa+1 are also transferred to the arithmetic section.

If  $(Aa) < (U) \leq (Aa+1)$ , the next instruction is skipped and the instruction following NI is performed.

If  $(U) \leq (Aa)$  or  $(U) > (Aa+1)$ , NI is performed.

1.  $+0$  is greater than  $-0$ .
2. Normally,  $(Aa) < (Aa+1)$ . If, however,  $(Aa) \geq (Aa+1)$ , there is no value of U that can satisfy the condition  $(Aa) < (U) \leq (Aa+1)$ .

#### 5.7.11. Test Not Within Range - TNW 57

Skip NI if  $(U) \leq (A)$  or  $(U) > (A+1)$ .

The contents of U is transferred to the arithmetic section under j-field control. The contents of Aa and Aa+1 are also transferred to the arithmetic section.

If  $(U) \leq (Aa)$  or  $(U) > (Aa+1)$ , the next instruction is skipped and the instruction following NI is performed.

If  $(U) > (Aa)$  and  $(U) \leq (Aa+1)$ , NI is performed.

1.  $+0$  is greater than  $-0$ .
2. Normally,  $(Aa) < (Aa+1)$ . If, however,  $(Aa) \geq (Aa+1)$ , every possible value of U will satisfy at least one of the following conditions:

$$(U) \leq (Aa)$$

or

$$(U) > (Aa+1)$$

#### 5.7.12. Test Positive - TP 60

Skip NI if  $(U)_{35} = 0$ .

The contents of U is transferred to the arithmetic section under j-field control.

If the sign bit (bit 35) of the value from U is a 0 bit, the next instruction is skipped and the instruction following NI is performed.

If the sign bit is a 1 bit, NI is performed.

1. The contents of the a-field are ignored.
2. Always skip when  $j = H1, H2, Q1-Q4, \text{ or } S1-S6$ .

#### 5.7.13. Test Negative - TN 61

Skip NI if  $(U)_{35} = 1$ .

The contents of U is transferred to the arithmetic section under j-field control.

If the sign bit (bit 35) of the value from U is a 1 bit, the next instruction is skipped and the instruction following NI is performed.

If the sign bit is a 0 bit, NI is performed.

1. The contents of the a-field are ignored.
2. Never skip when  $j = H1, H2, Q1 - Q4, \text{ or } S1 - S6$ .

#### 5.7.14. Double-Precision Test Equal - DTE 71,17

Skip NI if  $(U, U+1) = (A, A+1)$ .

The contents of U, U+1, Aa, and Aa+1 are transferred to the arithmetic section. U, U+1 and Aa, Aa+1 are 72-bit operands.

If  $(U, U+1) = (Aa, Aa+1)$ , the next instruction is skipped and the instruction following NI is performed.

If  $(U, U+1) \neq (Aa, Aa+1)$ , NI is performed.

1. +0 is not equal to -0.

## 5.8. SHIFT INSTRUCTIONS

Each shift instruction transfers either one or two words to the arithmetic section, moves or shifts the bits of the words, and stores the shifted word or words in one or two control registers.

The following basic types of shifts are provided for both single-word (36-bit input operand) and double-word (two 36-bit words treated as a 72-bit input operand) operations:

### ■ Right circular

For a right-circular shift, a shift count of  $n$  moves the contents of all bit positions of the register holding the input operand  $n$  bit positions to the right. Bits shifted out the right end of the register appear in the leftmost bit positions vacated by the shift.

### ■ Left circular

For a left-circular shift, a shift count of  $n$  moves the contents of all bit positions of the register holding the input operand  $n$  places to the left. Bits shifted out the left end of the register appear in the rightmost bit positions vacated by the shift.

For example: A shift count of 6 for a right-circular shift applied to  $765432101234_8$  as the input operand produces  $347654321012_8$  as the result. The same result is produced using a shift count of 30 for a left-circular shift.

For a single-word circular shift, a shift count of 72 or 36 produces the same result as a shift count of 0 (no shift). A shift count of 37 produces the same effect as a shift count of 1, a shift count of 38 produces the same effect as a shift count of 2, and so on.

### ■ Right logical

For a right-logical shift, a shift count of  $n$  moves the contents of all bit positions of the register holding the input operand  $n$  places to the right. Bits shifted out the right end of the register are lost. The leftmost bit positions vacated by the shift are zero filled.

For example: A shift count of 6 for a right-logical shift applied to  $765432101234_8$  as the input operand produces  $007654321012_8$  as the result.

### ■ Left Logical

For a left-logical shift, a shift count of  $n$  moves the contents of all bit positions of the input operand register  $n$  places to the left. Bits shifted out the left end of the register are lost. The rightmost bit positions vacated by the shift are zero filled.

For example: A shift count of 6 for a left-logical shift applied to  $765432101234_8$  as the input operand produces  $543210123400_8$  as the result.

### ■ Right algebraic

For an algebraic shift (right only, since no left algebraic shift is provided), a shift count of  $n$  moves the contents of all bit positions of the register holding the input operand  $n$  places to the right. Bits shifted out the right end of the register are lost. The bit positions vacated by the shift are filled with bits identical to the leftmost bit (sign bit) of the original input operand.

For example: A shift count of 6 for an algebraic shift applied to  $765432101234_8$  as the input operand produces  $777654321012_8$  as the result.

The two Load Shift and Count instructions are basically left circular shift instructions. The shift count is determined by the configuration of the bits of the input operand. If the two leftmost bits are not identical, the shift count is zero. If the two leftmost bits are identical, the operand is shifted left circular by the minimum amount to position the bits of the input operand so that the two leftmost bits are not identical. The shift count is the count of the number of bit positions shifted. If all bits of an input operand are identical, no amount of circular shifting will position its bits so that the two leftmost bits are not identical. In this instance, the shift count is 35 (single-word operand) or 71 (double-word operand). The shift count is stored in a control register.

For all shift instructions, except the two Load Shift and Count instructions, the input operands are specified by one or two A-registers, and the shift count is specified by bits 6 through 0 of the effective U. Indirect addressing, indexing, and index register incrementation/decrementation operate normally for all shift instructions.

The shift count can be any number between 0 and 72. If a shift count of 73 to 127 ( $111_g$  through  $177_g$ ) is specified, the result produced is undefined. The value in the u-field of the shift instruction and the value of  $X_m$  (if  $x \neq 0$ ) must be chosen accordingly.

For the two Load Shift and Count instructions, the effective U specifies the input operand address just as for the other load instructions. The scaled result is loaded in the specified A-register (A, A+1 for Double Load Shift And Count instruction). The number of shifts required for scaling is stored in the next consecutive register A+1 (or A+2 for Double Load Shift And Count instruction).

#### 5.8.1. Single Shift Circular - SSC 73,00

Shift (A) right circularly U places.

The contents of Aa is transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The value from Aa is shifted right circularly by the number of bit positions specified by the shift count. The shifted value is stored in Aa.

1. The result stored is not defined for shift counts greater than 72.
2. If  $36 \leq n \leq 72$ , a shift count of n produces the same result as a shift count of  $n-36$ .

#### 5.8.2. Double Shift Circular - DSC 73,01

Shift (A, A+1) right circularly U places.

The contents of Aa and Aa+1 are transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The 72-bit value from Aa and Aa+1 is shifted right circularly the number of bit positions specified by the shift count. The shifted value is stored in Aa and Aa+1.

1. The result stored is not defined for shift counts greater than 72.

#### 5.8.3. Single Shift Logical - SSL 73,02

Shift (A) right U places, zero fill.

The contents of Aa is transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The value from Aa is right shifted the number of bit positions specified by the shift count. Bits shifted out of the rightmost bit positions are lost; the vacated leftmost bit positions are zero filled. The shifted value is stored in Aa.

1. The result stored is not defined for shift counts greater than 72.
2. If  $36 \leq U \leq 72$ , the result stored in Aa is +0.

#### 5.8.4. Double Shift Logical – DSL 73,03

Shift (A,A+1) right U places, zero fill.

The contents of Aa and Aa+1 are transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The 72-bit value from Aa and Aa+1 is right shifted the number of bit positions specified by the shift count. Bits shifted out of the rightmost bit positions are lost; the vacated leftmost bit positions are zero filled.

1. The result stored is not defined for shift counts greater than 72.

#### 5.8.5. Single Shift Algebraic – SSA 73,04

Shift (A) right U places, sign fill.

The contents of Aa is transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The value from Aa is right shifted the number of bit positions specified by the shift count. Bits shifted out of the rightmost bit positions are lost; bits identical to the content of bit 35 of the initial value from Aa appear in the vacated leftmost bit positions. The shifted count is stored in Aa.

1. The result stored is not defined for shift counts greater than 72.
2. If  $35 \leq U \leq 72$ , all bits of the result stored in Aa are identical to the leftmost bit of the input operand from Aa.

#### 5.8.6. Double Shift Algebraic – DSA 73,05

Shift (A, A+1) right U places, sign fill.

The contents of Aa and Aa+1 are transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The 72-bit value from Aa and Aa+1 is right shifted the number of bit positions specified by the shift count. Bits shifted out of the rightmost bit positions are lost; bits identical to the contents of bit 35 of the initial value from Aa appear in the vacated leftmost bit positions. The shifted value is stored in Aa and Aa+1.

1. The result stored is not defined for shift counts greater than 72.

#### 5.8.7. Load Shift and Count – LSC 73,06

(U) → A; shift (A) left circularly until  $(A)_{35} \neq (A)_{34}$ ; number of shifts → A+1.

The contents of location U is transferred to a nonaddressable 36-bit register in the arithmetic section and then shifted left circularly the minimum number of bit positions which will make bit 35 unequal to bit 34. The resultant scaled number is transferred to Aa and the shift count to Aa+1.

1. If bit 35 of the value from location U is not equal to bit 34, the number is already scaled and no shift occurs:  $(U) \rightarrow Aa$ ;  $+ 0 \rightarrow A, A+1$ .
2. If the value from location U is  $\pm 0$ :  $(U) \rightarrow Aa$ , the shift count is 35, and  $43_8 \rightarrow Aa+1$ .

#### 5.8.8. Double Load Shift and Count - DLSC 73,07

$(U, U+1) \rightarrow A, A+1$ ; shift  $(A, A+1)$  left circularly until  $(A, A+1)_{71} \neq (A, A+1)_{70}$ ; number of shifts  $\rightarrow A+2$ .

The contents of U and U+1 are transferred to a nonaddressable 72-bit register in the arithmetic section and then shifted left circularly the minimum number of bit positions which will make bit 71 unequal to bit 70. The resultant scaled number is transferred to Aa and Aa+1 and the shift count to Aa+2.

1. If bit 71 of the value from U and U+1 are not equal to bit 70, the double length number is already scaled and no shift occurs:  $(U) \rightarrow Aa$ ;  $(U+1) \rightarrow Aa+1$ ;  $+0 \rightarrow Aa+2$ .
2. If the double-length value from locations U and U+1 are  $\pm 0$ :  $(U) \rightarrow Aa$ ;  $(U+1) \rightarrow Aa+1$ ; the shift count is 71;  $107_8 \rightarrow Aa+2$ .

#### 5.8.9. Left Single Shift Circular - LSSC 73,10

Shift (A) left circularly U places.

The contents of Aa is transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The value from Aa is shifted left circularly the number of bit positions specified by the shift count. The shifted value is stored in Aa.

1. The result stored is undefined for shift counts greater than 72.
2. If  $36 \leq n \leq 72$ , a shift count of n produces the same result as a shift count of n-36.

#### 5.8.10. Left Double Shift Circular - LDSC 73,11

Shift (A, A+1) left circularly U places.

The contents of Aa and Aa+1 are transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The 72-bit value from Aa and Aa+1 is shifted left circularly the number of bit positions specified by the shift count. The shifted value is stored in Aa and Aa+1.

1. The result stored is undefined for shift counts greater than 72.

#### 5.8.11. Left Single Shift Logical - LSSL 73,12

Shift (A) left U places, zero fill.

The contents of Aa is transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The value from Aa is left shifted the number of bit positions specified by the shift count. Bits shifted out of the leftmost bit positions are lost; the vacated rightmost bit positions are zero filled. The shifted value is stored in Aa.

1. The result stored is undefined for shift counts greater than 72.
2. If  $36 \leq U \leq 72$ , the result stored in Aa is +0.

### 5.8.12. Left Double Shift Logical - LDSL 73,13

Shift (A, A+1) left U places, zero fill.

The contents of Aa and Aa+1 are transferred to the arithmetic section. The shift count from bits 6 through 0 of U is transferred to the arithmetic section. The 72-bit value from Aa and Aa+1 is left shifted the number of bit positions specified by the shift count. Bits shifted out of the leftmost bit positions are lost; the vacated rightmost bit positions are zero filled. The shifted value is stored in Aa and Aa+1.

1. The result stored is undefined for shift counts greater than 72.

## 5.9. UNCONDITIONAL JUMP INSTRUCTION

A jump is a change in the sequence in which instructions are executed. It is accomplished by placing a new value in the P-register. Each unconditional jump instruction performs a unique operation in addition to the common operation of placing a new value in the P-register.

If the relative "jump to" address is less than  $200_g$ , the next instruction is taken from the storage location addressed by the value rather than from a control register.

The Jump Keys instruction can be used to specify either a conditional or an unconditional jump. The Halt Jump/Halt Keys And Jump instruction specifies an unconditional jump, but the halt portion is conditional. Both of these instructions are included in the section on conditional jump instructions (see 5.11).

### 5.9.1. Store Location and Jump - SLJ 72,01

Relative P+1  $\rightarrow$  U<sub>17-0</sub>: jump to U+1

The P-register is incremented. An 18-bit relative return address is stored in the rightmost 18 bits of the location specified by the operand address. The value of the operand address plus one is transferred to the P-register as the "jump to" address. The upper half of the operand is unchanged.

1. The contents of the a-field are ignored.
2. If  $U < 200_g$ , the 18-bit relative return address is stored in the rightmost 18 bits of the control register addressed by U and the leftmost 18 bit positions of that control register are unchanged.
3. The relative return address is stored in the low-order 18 bits of a word. If this 18-bit relative return address is larger than 16 bits, the two high order bits will be interpreted as h and i bits if the address is used in an instruction. The instruction may produce erroneous results.

### 5.9.2. Load Modifier and Jump - LMJ 74,13

Relative  $P+1 \rightarrow Xa_{17-0}$ ; jump to U

The P-register is incremented. An 18-bit relative return address is stored in the rightmost 18 bits of the index register specified by the a-field. The leftmost 18 bits of that index register are not affected. The value of the operand is transferred to the P-register as the "jump to" address.

1. If  $D6 = 0$  and the value in the a-field is zero, the relative return address is stored in index register zero (X0).
2. If index register incrementation is specified, the relative return address is stored in the index register specified by the a-field after the new value for  $X_m$  is stored in the index register specified by the x-field. As a consequence, if the value in the a-field is not zero and it is the same as the value in the x-field, it makes no difference whether the value in the h-field is zero or one.

### 5.9.3. Allow All Interrupts and Jump - AAIJ 74,07

Allow all interrupts and jump to U.

This instruction allows interrupts prevented by the occurrence of an interrupt or the execution of a Prevent All Interrupts And Jump instruction.

1. The contents of the a-field are ignored.
2. The Allow All Interrupts And Jump instruction does not affect the Dayclock interrupt when it is disabled by the Disable Dayclock instruction and enabled by the Enable Dayclock instruction.

## 5.10. BANK DESCRIPTOR SELECTION INSTRUCTIONS

Each program may be composed of or associated with a large number of program or data segments; of these, up to four may be active at any given time. Bank Descriptor selection instructions allow a program to select which segments are among the four that are currently active.

### 5.10.1. Load Bank and Jump - LBJ 07,17

The LBJ instruction loads the bank descriptor register selected in bit positions 34 and 33 of the index register specified by the a-field of the instruction word ( $X_a$ ) with a new bank descriptor, stores the old bank descriptor specifications and program address in  $X_a$  as return information, and then jumps to the location specified by the operand address. An Addressing Exception interrupt occurs if the bank descriptor register named in  $X_a$  is not available for use as defined by the designator register. The new bank descriptor is located by adding the bank descriptor index contained in bit positions 18 through 29 of  $X_a$  to the bank descriptor table pointer selected in bit position 35 of  $X_a$ ; if bit 35 is zero, the user pointer and table are selected, and if bit 35 is one, the Executive pointer and table are selected if D19 is one. An Addressing Exception interrupt occurs if the bank descriptor index value exceeds the length of the table, or if bit 35 of  $X_a$  is one and D19 is zero.

When the new bank descriptor is located, the associated use count field is increased by one under storage lock, and an Addressing Exception interrupt occurs if the R-flag of the bank descriptor is one, if there is a V-flag violation, or if the use-count field is increased from all ones to zero.

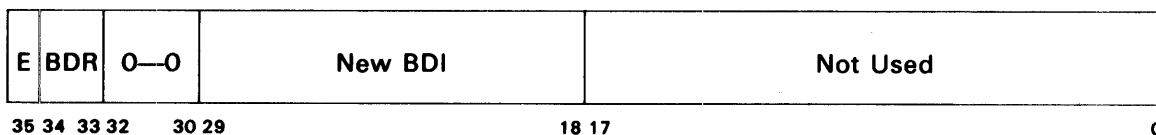


Before the new bank descriptor values are actually loaded, the old bank descriptor is located and the use-count field is decreased by one under storage lock. An Addressing Exception interrupt occurs if the C-flag of the old bank descriptor is one and the use count is decreased to zero, or if the use count is decreased from zero to all ones. The new bank descriptor is loaded in the bank descriptor register, the P-flag is transferred to designator register bit 2 (D2), and the W-flag of the new bank descriptor is placed in the appropriate write-protection bit of the designator register (D13 through D16).

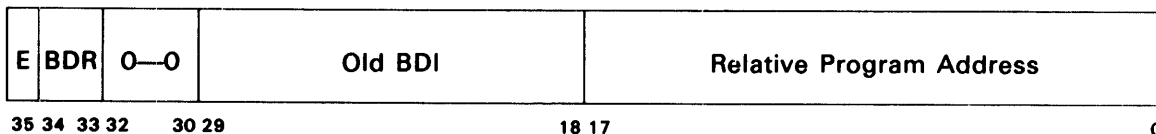
The specifications of the old bank descriptor is copied from the GRS processor-state storage area into the upper half of Xa, the relative program address is copied into the lower half of Xa, and the specifications of the new bank descriptor is then stored in GRS. The operand address is formed and a jump to that location is effected. If both an address exception and jump address guard mode limits violation occur during the execution of this instruction, the address exception will be taken.

The following are the formats of Xa before and after execution of the instruction:

Xa Before Execution



Xa After Execution



**5.10.2. Load I-Bank Base and Jump - LIJ 07,13**

The LIJ instruction is executed as a special case of the LBJ instruction. Bit positions 34-33 of Xa is ignored; if D12 is zero, BDR0 is selected, and if D12 is one, BDR1 is selected.

**5.10.3. Load D-Bank Base and Jump - LDJ 07,12**

The LDJ instruction is executed as a special case of the LBJ instruction. Bit positions 34-33 of Xa is ignored; if D12 is zero, BDR2 is selected, and if D12 is one, BDR3 is selected.

## 5.11. CONDITIONAL JUMP INSTRUCTIONS

Each of the conditional jump instructions performs a test for a specific condition (or set of conditions). If the condition is satisfied, the value U is transferred to the P-register and the instruction addressed by U is performed next. If the condition is not satisfied, the next instruction (NI) is performed.

### 5.11.1. Jump Greater and Decrement - JGD 70

Jump to U if (control register)  $> 0$ ; go to NI if (control register)<sub>ja</sub>  $\leq 0$ ; always (control register)<sub>ja</sub> - 1  $\rightarrow$  control register<sub>ja</sub>.

If the 36-bit signed number in the control register addressed by the rightmost 7 bits of the ja-field is greater than zero (bit 35 contains a 0 bit and the number does not consist of all 0 bits), the instruction at location U is executed next. If the number is less than, or equal to, zero (bit 35 contains a 1 bit or the number consists of all 0 bits), the next instruction is performed. In either case, the number is decreased by one and the difference is stored in the control register addressed by the ja-field.

1. A Guard Mode interrupt occurs (if guard mode is set) when the ja-field specifies a value in the range  $40_8$  through  $100_8$ , or  $120_8$  through  $177_8$  when  $D2 = 1$ . This is true regardless of the value of D6 (A-, X-, and R-register set selector).
2. The leftmost bit in the j-field is ignored.

### 5.11.2. Double-Precision Jump Zero - DJZ 71,16

Jump to U if  $(A, A+1) = \pm 0$ ; go to NI if  $(A, A+1) \neq \pm 0$ .

If the 72-bit operand contained in Aa and Aa+1 is  $\pm 0$ , the instruction at location U is performed next. If the operand is not  $\pm 0$ , the next instruction (NI) is performed.

### 5.11.3. Jump Positive and Shift - JPS 72,02

Jump to U if  $(A)_{35} = 0$ ; go to NI if  $(A)_{35} = 1$ ; always shift (A) left circularly one bit position.

If bit 35 of Aa contains a 0 bit, the instruction at location U is performed next. If bit 35 contains a 1 bit, the next instruction is performed. The contents of Aa is always shifted left circularly one bit position.

1. The bit shifted out of bit 35 of Aa is shifted to bit 0 of Aa.

### 5.11.4. Jump Negative and Shift - JNS 72,03

Jump to U if  $(A)_{35} = 1$ ; go to NI if  $(A)_{35} = 0$ ; always shift (A) left circularly one bit position.

If bit 35 of Aa is a 1 bit, the instruction at location U is performed next. If bit 35 is a 0 bit, the next instruction is performed. The contents of Aa is always shifted left circularly one bit position.

1. The bit shifted out of bit 35 of Aa is shifted to bit 0 of Aa.

**5.11.5. Jump Zero - JZ 74,00**

Jump to U if  $(A) = \pm 0$ ; go to NI if  $(A) \neq \pm 0$ .

If  $(Aa)$  is  $\pm 0$ , the instruction at location U is performed next. If  $Aa$  does not contain  $\pm 0$ , the next instruction is performed.

**5.11.6. Jump Nonzero - JNZ 74,01**

Jump to U if  $(A) \neq \pm 0$ ; go to NI if  $(A) = \pm 0$ .

If  $(Aa)$  is not  $\pm 0$ , the instruction at location U is performed next. If  $(Aa)$  is  $\pm 0$ , the next instruction is performed.

**5.11.7. Jump Positive - JP 74,02**

Jump to U if  $(A)_{35} = 0$ ; go to NI if  $(A)_{35} = 1$ .

If bit 35 of  $Aa$  is a 0 bit, the instruction at location U is performed next. If bit 35 is a 1 bit, the next instruction is performed.

**5.11.8. Jump Negative - JN 74,03**

Jump to U if  $(A)_{35} = 1$ ; go to NI if  $(A)_{35} = 0$ .

If bit 35 of  $Aa$  is a 1 bit, the instruction at location U is performed next. If bit 35 is a 0 bit, the next instruction is performed.

**5.11.9. Jump - Jump Keys - J,JK 74,04**

Jump to U if  $a = 0$  or if  $a =$  set SELECT JUMPS switch; go to NI if neither is true.

If the  $a$ -field contains all 0 bits, the instruction at location U is performed next. If the  $a$ -field contains a value in the range of 1 through 15 ( $1_8$  through  $17_8$ ) and the correspondingly numbered SELECT JUMPS switch/indicator is set, the instruction at location U is performed next; if the correspondingly numbered SELECT JUMPS switch/indicator is not set, the next instruction is performed.

1. The indicator for each of the 15 SELECT JUMPS switch/indicators is turned on by pressing that SELECT JUMPS switch/indicator. Each is turned off by pressing the associated clear switch. Either can be done while the processor is running.
2. Care should be exercised in using a value other than all 0 bits in the  $a$ -field if the program is to run concurrently with one or more other programs. Any other program may include a Jump Keys instruction with the same value in the  $a$ -field and specify that it is to be run with the corresponding SELECT JUMPS switch/indicator set.

**5.11.10. Halt Jump – Halt Keys and Jump – HJ,HKJ 74,05**

Stop if [ $a=0$  OR if ( $a$  AND set SELECT STOPS switches)  $\neq 0$ ] AND  $D2 = 0$ ; on restart or continuation jump to U.

If the a-field contains all 0 bits, the execution of program instruction halts. If the a-field contains a 1 bit in a bit position which corresponds to a lit SELECT STOPS switch/indicator, the program halts.

In either case, a manual restart causes the instruction at location U to be performed next.

If neither of the conditions described above is fulfilled, the instruction at location U is performed and the program does not halt.

1. Unless the processor is operating in privileged mode ( $D2=0$ ) when a halt condition is satisfied for a Halt Keys And Jump instruction, it does not actually halt. Instead, it proceeds with the jump.
2. The indicator for each of the four SELECT STOPS switch/indicators is turned on by pressing one of the SELECT STOPS switch/indicators. They are turned off by pressing the associated clear switch.
3. If the program address register is manually changed while the processor is halted, program execution will resume at the new address when the processor is restarted.

**5.11.11. Jump No Low Bit – JNB 74,10**

Jump to U if  $(A)_0 = 0$ ; go to NI if  $(A)_0 = 1$ .

If bit 0 of Aa is a 0 bit, the instruction at location U is performed next. If bit 0 is a 1 bit, the next instruction is performed.

1. If the Jump No Low Bit instruction is used to determine whether the value in Aa is an even or an odd integer, consideration must be given to the sign of the value.

**5.11.12. Jump Low Bit – JB 74,11**

Jump to U if  $(A)_0 = 1$ ; go to NI if  $(A)_0 = 0$ .

If bit 0 of Aa is a 1 bit, the instruction at location U is performed next. If bit 0 is a 0 bit, the next instruction is performed.

1. If a Jump Low Bit instruction is used to determine whether the value in Aa is an even or an odd integer, consideration must be given to the sign of the value.

**5.11.13. Jump Modifier Greater and Increment – JMGI 74,12**

Jump to U if  $(Xa)_{17-0} > 0$ ; go to NI if  $(Xa)_{17-0} \leq 0$ ; always  $(Xa)_{17-0} + (Xa)_{35-18} \rightarrow Xa_{17-0}$ .

If the signed number in bits 17 through 0 of the X-register specified by the a-field is greater than zero (bit 17 is a 0 bit and the number does not consist of all 0 bits), the instruction at location U is performed next. If the number is less than or equal to zero (bit 17 is a 1 bit or the number consists of all 0 bits), the next instruction is performed. In either case, the signed number in bits 35 through 18 of the X-register is added to the signed number in bits 17 through 0 and the sum is stored in bits 17 through 0 of the X-register.

1. The number in  $Xa_{17-0}$  before the addition is tested rather than the number resulting from the addition.
2. If  $a = x$  and  $h = 1$ , the specified index register is effectively modified only once for each execution of the instruction.

#### 5.11.14. Jump Overflow - JO 74,14; a = 0

Jump to U if  $D1 = 1$ ; go to NI if  $D1 = 0$ .

Where a-field is an extension of f- and j-field.

If the overflow indicator ( $D1$ ) is one, the instruction at location U is performed next. If  $D1$  is zero, the next instruction is performed.

1. Performing the Jump Overflow instruction does not change  $D1$ .

#### 5.11.15. Jump Floating Underflow - JFU 74,14; a = 1

Jump to U if  $D21 = 1$ , clear  $D21$ ; go to NI if  $D21 = 0$ .

If the characteristic underflow indicator ( $D21$ ) is one, the instruction at location U is performed next and  $D21$  is cleared by the instruction. If  $D21$  is zero, the next instruction is performed.

#### 5.11.16. Jump Floating Overflow - JFO 74,14; a = 2

Jump to U if  $D22 = 1$ , clear  $D22$ ; go to NI if  $D22 = 0$ .

If the characteristic overflow indicator ( $D22$ ) is one, the instruction at location U is performed next and  $D22$  is cleared by the instruction. If  $D22$  is zero, the next instruction is performed.

#### 5.11.17. Jump Divide Fault - JDF 74,14; a = 3

Jump to U if  $D23 = 1$ , clear  $D23$ ; go to NI if  $D23 = 0$ .

If the divide fault indicator ( $D23$ ) is one, the instruction at location U is performed next and  $D23$  is cleared by the instruction. If  $D23$  is zero, the next instruction is performed.

#### 5.11.18. Jump No Overflow - JNO 74,15; a = 0

Jump to U if  $D1 = 0$ ; go to NI if  $D1 = 1$ .

If the overflow indicator ( $D1$ ) is zero, the instruction at location U is performed. If  $D1$  is one, the next instruction is performed.

1. Executing the Jump No Overflow instruction does not change  $D1$ .

**5.11.19. Jump No Floating Underflow - JNFU 74,15; a = 1**

Jump to U if D21 = 0; go to NI if D21 = 1; clear D21.

If the characteristic underflow indicator (D21) is zero, the instruction at location U is performed next. If D21 is one, the next instruction is performed. D21 is cleared by the instruction.

**5.11.20. Jump No Floating Overflow - JNFO 74,15; a = 2**

Jump to U if D22 = 0; go to NI if D22 = 1; clear D22.

If the characteristic overflow indicator (D22) is zero, the instruction at location U is performed next. If D22 is one, the next instruction (NI) is performed. D22 is cleared by the instruction.

**5.11.21. Jump No Divide Fault - JNDF 74,15; a = 3**

Jump to U if D23 = 0; go to NI if D23 = 1; clear D23.

If the divide fault indicator (D23) is zero, the instruction at location U is performed next. If D23 is one, the next instruction is performed next. D23 is cleared by the instruction.

**5.11.22. Jump Carry - JC 74,16**

Jump to U if DO = 1; go to NI if DO = 0.

If the carry indicator (DO) is one, the instruction at location U is performed next. If DO is zero, the next instruction is performed.

1. The contents of the a-field are ignored.
2. Performing the Jump Carry instruction does not change DO.

**5.11.23. Jump No Carry - JNC 74,17**

Jump to U if DO = 0; go to NI if DO = 1.

If the carry indicator (DO) is zero, the instruction at location U is performed next. If DO is one, the next instruction is performed.

1. The contents of the a-field are ignored.
2. Performing the Jump No Carry instruction does not change DO.

**5.12. LOGICAL INSTRUCTIONS**

The three logical operations are the Logical Inclusive OR (referred to as the Logical OR and symbolized by **OR**), the Logical Exclusive OR (symbolized by **XOR**); and the Logical AND (symbolized by **AND**). Each of these instructions uses two input operands. One input operand is obtained from location U and the other from an A-register. Table 5-1 lists the four possible combinations of the two bits from any bit position of the two input operands and the result produced for that bit position for each of the three basic operations.

Table 5-1. Truth Table for Logical OR, XOR, and AND

Input Bits		Output (Result) Bit		
First Operand	Second Operand	OR	XOR	AND
0	0	0	0	0
0	1	1	1	0
1	0	1	1	0
1	1	1	0	1

The Masked Load Upper instruction performs a compound logical operation; the contents of selected bit positions of one operand are merged with the contents of the remaining bit positions of a second operand.

### 5.12.1. Logical OR – OR 40

(A)  $\boxed{\text{OR}}$  (U)  $\rightarrow$  A+1

The contents of Aa is transferred to the arithmetic section. The contents of U is transferred to the arithmetic section under j-field control. A 36-bit result is formed in the arithmetic section, as follows:

- The result contains a 1 in each bit position for which the corresponding bit position of either (or both) of the input operands contains a 1.
- The result contains a 0 in each bit position for which the corresponding bit position of both input operands contains a 0.

The result is stored in Aa+1.

### 5.12.2. Logical Exclusive OR – XOR 41

(A)  $\boxed{\text{XOR}}$  (U)  $\rightarrow$  A+1

The contents of Aa is transferred to the arithmetic section. The contents of U is transferred to the arithmetic section under j-field control. A 36-bit result is formed in the arithmetic section, as follows:

- The result contains a 1 in each bit position for which the corresponding bit position of either (but not both) of the input operands contains a 1.
- The result contains a 0 in each bit position for which the contents of the corresponding bit position of the input operands are both 0 or both 1.

The result is stored in Aa+1.

### 5.12.3. Logical AND – AND 42

(A)  $\boxed{\text{AND}}$  (U)  $\rightarrow$  A+1

The contents of Aa is transferred to the arithmetic section. The contents of U is transferred to the arithmetic section under j-field control. A 36-bit result is formed in the arithmetic section, as follows:

- The result contains a 1 in each bit position for which the corresponding bit position of both input operands contains a 1.
- The result contains a 0 in each bit position for which the corresponding bit position of either (or both) of the input operands contains a 0.

The result is stored in  $Aa+1$ .

#### 5.12.4. Masked Load Upper - MLU 43

[ (U)  $\overline{\text{AND}}$  (R2) ]  $\overline{\text{OR}}$  [ (A) AND NOT (R2) ]  $\rightarrow A+1$

The contents of  $Aa$  and  $R2$  are transferred to the arithmetic section. The contents of  $U$  is transferred to the arithmetic section under  $j$ -field control. A 36-bit result is formed in the arithmetic section, as follows:

- The result contains a 1 in each bit position for which the corresponding bit position of the operand from  $U$  and the operand from  $R2$  both contain 1 bits.
- The result contains a 1 in each bit position for which the corresponding bit position of the operand from  $Aa$  and the ones complement of the operand from  $R2$  both contain 1 bits.
- The result contains 0 bits in the remaining bit positions.

The result is stored in  $Aa+1$ .

1. The desired value must be loaded in  $R2$  (mask register) by an instruction preceding the Masked Load Upper instruction.

### 5.13. MISCELLANEOUS INSTRUCTIONS

Each of the eight following instructions is classed as miscellaneous.

#### 5.13.1. Load DR Designators - LPD 07,14

$U_{7-5,3-0} \rightarrow$  Designator register:

Bit 0  $\rightarrow$  D4    Bit 3  $\rightarrow$  D10

Bit 1  $\rightarrow$  D5    Bit 5  $\rightarrow$  D17

Bit 2  $\rightarrow$  D8    Bit 6  $\rightarrow$  D20

Bits 0, 1, 2, 3, 5, 6, and 7 of  $U$  are transferred to the designator register bits; D4, D5, D8, D10, D17, and D20, respectively. These are the only designator bits which can be changed by a user program.

#### 5.13.2. Store DR Designators - SPD 07,15

Designator register bits  $\rightarrow U_{7-0}$ ; zeros  $\rightarrow U_{17-8}$

D4  $\rightarrow$  Bit 0    D12  $\rightarrow$  Bit 4



D5 → Bit 1    D17 → Bit 5

D8 → Bit 2    D20 → Bit 6

D10 → Bit 3

D20, D17, D12, D10, D8, D5, and D4 of the designator register are transferred to bit positions 7-0 of U, respectively. The upper half of the operation location is unaffected.

### 5.13.3. Execute - EX 72,10

Execute the instruction at U.

The P-register is incremented provided the instruction was addressed by the contents of the P-register. The instruction at location U is transferred to the control section to replace the Execute instruction as the next instruction to be performed.

1. The contents of the a-field are ignored.
2. The remote instruction, specified by U, is always obtained from a storage location.
3. Execute instructions may be cascaded; that is, the instruction in the remote location may be an Execute instruction.
4. The P-register is incremented only once, when the original Execute instruction is obtained for execution.
5. Generally, an interrupt cannot occur between the time an Execute instruction is started and the instruction (or instructions) it leads to has been completed except when an Execute instruction leads to a repeated instruction (see 5.3.8 and 5.6). An interrupt cannot occur between the start of the Execute instruction and the completion of the initial stage of the repeated instruction. The interrupt, however, can cause initiation of a termination stage immediately following completion of the initial stage or any time thereafter in order to permit the interrupt to occur.
6. If an Execute instruction leads to a repeated instruction, index register incrementation should not be specified for the Execute instructions or for any indirect addressing sequence involved (see 5.3.8. note 6, and 5.6).

### 5.13.4. Executive Request - ER 72,11

Generate Executive Request interrupt

An Executive Request interrupt is generated.

1. A Guard Mode/Storage Limits interrupt will occur if indirect addressing is specified ( $i = 1$ ,  $D7 = 0$ ) and the operand address causes a storage limits violation.
2. The contents of the a-field are ignored.

**5.13.5. Test and Set - TS 73,17; a = 0**

If  $(U)_{30} = 1$ , Generate Test and Set interrupt; if  $(U)_{30} = 0$ , go to NI;

always  $01_8 \rightarrow U_{35-30}$ ;  $(U)_{29-0}$  unchanged.

A storage cycle is initiated to read and then write the operand specified by the operand address. If bit 30 of the operand is one, a Test and Set interrupt occurs. If bit 30 of the operand is zero, the next instruction is performed. The write portion of the storage cycle includes writing ones in bits 35 through 30 of the storage operand. Bits 29 through 0 at location U are neither examined nor altered.

1. If  $U < 200_8$ , always interrupt.

**5.13.6. Test and Set and Skip - TSS 73,17; a = 1**

If  $(U)_{30} = 0$ , skip NI; if  $(U)_{30} = 1$ , go to NI; always  $01_8 \rightarrow U_{35-30}$ ;  $(U)_{29-0}$  unchanged.

A storage cycle is initiated to read and then write the operand specified by the operand address. If bit 30 of the operand is zero, the next instruction is skipped. If bit 30 of the operand is one, the next instruction is performed. The write portion of the storage cycle includes writing ones in bits 35 through 30 of storage location U. Bits 29 through 0 at location U are neither examined nor altered.

1. If  $U < 200_8$ , always execute NI.

**5.13.7. Test and Clear and Skip - TCS 73,17; a = 2**

If  $(U)_{30} = 0$ , perform NI; if  $(U)_{30} = 1$ , skip NI; always clear  $(U)_{35-30}$ ;  $(U)_{29-0}$  unchanged.

A storage cycle is initiated to read and then write the operand specified by the operand address. If bit 30 of the operand is zero, the next instruction is performed. If bit 30 of the operand is one, the next instruction is skipped. The write portion of the storage cycle includes writing zeros in bits 35 through 30 of storage location U. Bits 29 through 0 at location U are neither examined nor altered.

1. If  $(U) < 200_8$ , always execute NI.

**5.13.8. No Operation - NOP 74,06**

Proceed to next instruction.

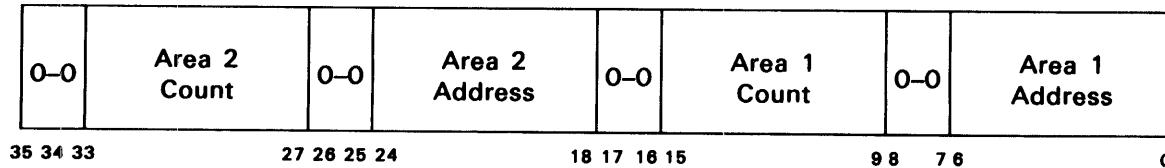
This instruction ensures that there is an interval between the end of the instruction that precedes it and the start of the one that follows it.

1. The contents of the a-field are ignored.
2. The only effects that the values in the x-, h-, i-, and u-fields can have on the operation is the index register incrementation obtained when  $x \neq 0$  and  $h = 1$ , and the indirect addressing delay introduced when  $i = 1$  and  $D7 = 0$ .

**5.13.9. Store Register Set - SRS 72,16**

Aa contains an address and count for each of two GRS areas. These areas are stored consecutively, starting at the location specified by the operand address of the instruction. If either or both count values are zero, no transfer occurs to the respective area(s).

The following is the format of Aa for this instruction:

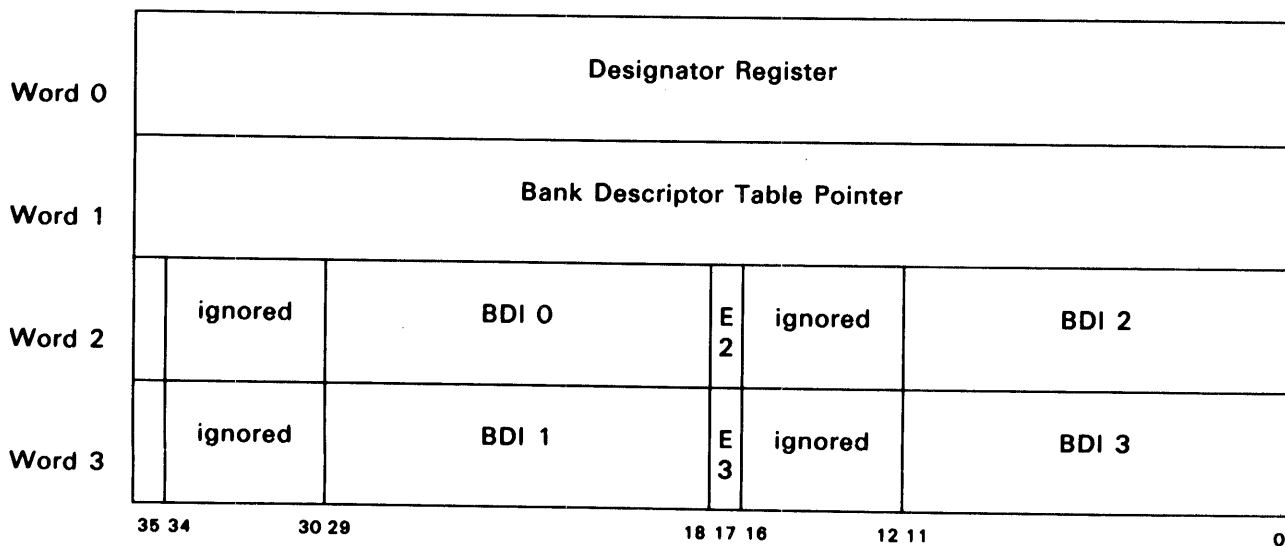


**5.13.10. Load Register Set - LRS 72,17**

The format of Aa and the operation of the instruction are like that of SRS, except that information is transferred from the location specified by the operand address to the area specified by Aa.

**5.13.11. Test Relative Address - TRA 72,15**

This instruction provides a means to determine whether a specific relative address is within a given relative addressing range. The operand address is the first word of a four-word packet defining an addressing environment to be used in testing the relative address. The packet contains a designator register, bank descriptor table pointer, four bank descriptor indexes, and E bits, in the following format:



The relative address to be tested is contained in  $Xa_{17-00}$ . This relative address is translated into an absolute address within the addressing environment specified by the above packet. Relative addresses less than  $200_8$  are treated as a storage address, not GRS addresses. The four E bits within the packet determine whether the BDT pointer in the packet ( $E=0$ ) or the EXEC BDT pointer ( $E=1$ ) contained in GRS  $40_8$  is to be used to reference the respective bank descriptor. The TRA instruction ignores D19, does not check table length violations, does not cause MSR basing, and will not produce a Guard Mode interrupt as a result of a relative address out of limits.

Prior to fetching of packet Word 0, the designator register is stored in GRS  $44_8$ . A check for EXEC GRS area storage is not made on this store. This store has the effect of making the current designator state, specifically D12, available for use by a user who is normally expected to execute the TRA instruction with an operand address of  $44_8$ . At the same time, this allows use of some pre-set designator state to be used if the operand address is other than 44.

The results of this instruction are stored in Xa and indicated by skip or no skip. If the relative address tested is within limits, the number of the bank descriptor register within whose limits the relative address exists, is stored in  $Xa_{34-33}$ , and the absolute address produced is stored in  $Xa_{23-00}$ . If the relative address does not fall within any limits, Xa is cleared to zero and the next instruction is executed. If the relative address tested is within limits, the write protect bit of the Bank Descriptor within whose limits the relative address exists is tested. If it is zero, the next instruction is skipped; if it is one, the next instruction is executed.

#### 5.13.12. Increase Instructions - XX 05; a = 10-17

The operand specified by the operand address is transferred under j-field control to the arithmetic section, increased by a value specified by the a-field control to the arithmetic section, increased by a value specified by the a-field, and stored under j-field control in the location specified by the operand address; the operation is performed under storage lock (test and set). If the initial operand or the result is zero, the next instruction is executed; otherwise, the next instruction is skipped. The following values may be selected by the a-field:

mnemonic	a-field	increase value
INC	10	+1 plus one
DEC	11	-1 minus one
INC2	12	+2 plus two
DEC2	13	-2 minus two
ENZ	14 - 17	0 zero (-0 is changed to +0 for sign-extended operands)

The increase and zero test operations depend on the j-field values to some degree. Certain j-field values extend or interpret the sign of the operand (W, XH1-XH2, T1-T3); for these values, the increase is a ones complement, sign-extended operation, and either positive zero or negative zero satisfies the zero test. The remaining j-field values do not consider the sign of the operand (H1-H2, Q1-Q4, S1-S6); for these values, the increase is a twos complement, field-size operation, and only positive zero satisfies the zero test.



Field	Function
I	J-register modifier bit; used with the h-bit of the instruction to control J (I=1) or X(I=0) register modification.
M	Mode 6/9-bit modulus: 0 = 9-bit mode (ASCII) 1 = 6-bit mode (Fieldata)
W	Width 6-12 or 9-18 bits: 0 = 6/9 bits 1 = 12/18 bits
E	a) for 33,03 E = Translate: 0 = translation 1 = no translation  b) for all other byte instructions E must be zero.  c) for character addressing (non-byte instructions): 0 = no sign extension 1 = sign extension
Iw	Increment in words
Ib	Increment in bytes
Ow	Offset in words
Ob	Offset in bytes

The direction in which each instruction progresses through its operand byte strings is specified per instruction in the J-register. The increment word (Iw) and increment byte (Ib) fields of the J-register are used during instruction execution to update the effective byte address. The effective value of Iw and Ib may be either  $\pm 1$ , the actual value loaded into the register by the program depends on the byte length being used. The value of Iw must be  $\neq 0$  and have the same sign as Ib. Therefore Iw is effectively Iwb sign extended (Iwb).

Table 5-2 gives the values of Iwb for +1 and -1 effective increments for 6, 9, 12 and 18-bit bytes.

Some of the extended-sequence byte-manipulation instructions are designed to permit their interruption during their execution. However, interrupts are accepted only following the store or compare phase of the instruction. As the instruction completes each of these phases, a check is made to see if an interrupt is waiting to be processed. If an interrupt request is current, it is acknowledged and processed immediately. When the instruction is again activated, the interrupt control bits are decoded, and control is returned to the appropriate phase of execution. There are three bits (29-27) in SR1 that are available for interrupt control, thus providing up to 7 types of interrupt classification within an instruction.

Table 5-2. J-Register Increment Field Values

For a Byte Length of	And an Effective Increment of	The Value of I Must be
6 bits	+ 1	+ 1
6	- 1	- 1
9	+ 1	+ 2
9	- 1	- 2
12	+ 1	+ 2
12	- 1	- 2
18	+ 1	+ 4
18	- 1	- 4

There are seven restrictions on byte addressing that should be noted:

1. The result of an instruction performed on overlapping byte strings is undefined.
2. A byte string may not wrap around its J-register offset field; i.e., 0w cannot be incremented through its maximum value of 77777<sub>8</sub> or decremented through its minimum value of 00000<sub>8</sub>.
3. Normal address limits violation detection and interrupt will be in effect.
4. All instructions utilizing the J-registers must have their operands located in storage.
5. The h-field of all byte instructions must be set to 1. This is set automatically by the assembler when a byte mnemonic is encountered.
6. The I-field of all J-registers used must be set to 1.
7. The E-field of all J-registers used must be set to 0 for all byte instructions except 33,03.

The 33,03-04; 33, 10-11; 33, 14-17; and 37, 06-07 instruction will store a 7-bit status word in SR3<sub>17-9</sub> either upon successful completion of the instruction or upon detection of an error condition which prevents completion of the instruction. A definition of the 7 status bits is contained in Table 5-3.

Successful completion of an instruction will result in the storing of an all-zero word except for the cases of a decimal-add overflow (27, 06-07) or a missing mantissa field (33, 14-15).

When dealing with 9-bit bytes, the ASCII format shall be accepted and only ASCII is generated when used for operations involving signed numeric-byte strings. An ASCII byte is the eight lowest-order bits in a quarter word. The byte is divided into a 4-bit zone and a 4-bit digit; the zone is the most significant part of the byte. The sign convention adopted for a byte string is called "trailing-included sign format," i.e., the sign of the byte string is contained in the zone (Z) portion of the least significant byte.

There are three exceptions to the "trailing-included sign convention". The Byte-to-Single Floating Conversion (fj = 33, 15) instructions use a separate non-included sign byte with the byte string. This byte is simply a "+" or "-" character.

Table 5-4 gives the binary coding for the plus and minus signs to be used in ASCII and Fielddata coding. The hardware checks for a minus sign in arithmetic operations. If the sign of the arithmetic operations is not minus, then the result is assumed to be plus. The types of signs accepted and generated by each of the byte-manipulation instructions are listed in the table.

Table 5-3. Byte Status Word

Status Bit	Type of Error		Instruction and Condition Detected
Bit 0 Set	Format Error	33-10,11	Byte not digit or blank (checked on all but last byte) or least significant 4 bits of last byte greater than 9.
		37-06,07	Byte not digit (checked on all but first byte) or least significant 4 bits or first bytes greater than 9.
		33-14,15	<ul style="list-style-type: none"> <li>a. Two signs in string not separated by at least one non-blank character.</li> <li>b. Two decimal points in mantissa</li> <li>c. Significant character not found.</li> <li>d. Illegal character in string.</li> <li>e. Illegal character in exponent.</li> <li>f. Decimal point last character and no digit in string.</li> </ul>
Bit 1 Set	Underflow	33-15	Magnitude of input too small to represent in double-precision floating-point number.
		33-14,15	Exponent negative and power of ten too small to represent double-precision floating-point format.
Bit 2 Set	Overflow	33-10	Magnitude of input too large to represent in 35 binary bits.
		33-11	Magnitude of input too large to represent in 71 binary bits.
		37-06,07	Decimal-add overflow.
		33-14	Magnitude of input too large to represent in single-precision floating-point.
		33-15	Magnitude of input too large to represent in double-precision floating-point.
		33-14,15	Mantissa interpreted as integer too large to represent in 60 binary bits.
Bit 3 Set	Decimal Point Error	33-14,15	<ul style="list-style-type: none"> <li>a. Decimal point count greater than 31.</li> <li>b. Two decimal points in mantissa.</li> <li>c. Decimal point last character and no digit in string.</li> <li>a. Bits 0, 3, or 6 set and significant character not read yet.</li> </ul>
Bit 4 Set	No Significant Character Found	33-14,15	<ul style="list-style-type: none"> <li>b. Mantissa field does not contain at least one digit (note that a blank following a decimal point is considered a digit).</li> <li>c. String does not contain at least one nonblank and nonsign character.</li> <li>a. Bits 0, 1, 2, 3, or 6 set and exponent field detected.</li> </ul>
Bit 5 Set	Exponent Found or Byte Roundup	33-14,-15	
Bit 6 Set	Mode Error	33-16-17	
		33-10,11	Byte 10 (33,16) or 19 (33,17) is greater than four.
		33-14,15	6- or 9-bit mode not selected (W bit) on one of the following instructions.
Bit 7 Set	Byte Compare	33,03-04	Compare encountered during instruction.



Table 5-4. Byte String Sign Codes

Character Code Formats		Sign Conventions	
		+	-
1. ASCII	Included (Zone portion)	1010	1011
2. Fieldata	Included	11	10
3. ASCII	Separated (Entire byte)	00101011	00101101
4. Fieldata	Separate	100010	100001

#### 5.14.1. Byte Move - BM 33,00

Transfer LJ1 bytes from source string to receiving string. Truncate or fill.

This instruction transfers LJ1 bytes from a source string starting at address SJ0 to a receiving string starting at address SJ1. The byte string at address SJ0 contains LJ0 bytes; the byte string at address SJ1 contains LJ1 bytes. If LJ1 is less than LJ0, the move will be truncated when LJ1 bytes have been transferred. If LJ1 is greater than LJ0, then (LJ1-LJ0) fill bytes will be added in the trailing positions of the byte string located at address SJ1. The contents of SR2<sub>17-0</sub> are used as the fill byte.

When byte strings of different byte size are transferred, the receiving string determines how many bits from each source string byte will be accepted. For example, if SJ1 is in the nine-bit mode and LJ- is in six-bit mode, the three leading bits of the SJ1 byte are made zero. If SJ1 is in the six-bit mode and SJ0 is in the nine-bit mode, only the six least significant bits of the SJ0 byte are accepted, the rest being lost.

Both the values LJ1 and LJ0 are reduced by one following each byte move. This instruction is terminated when the value of LJ1 equals zero (LJ1 = 0).

1. This instruction is interruptible after each store operation.
2. The lwb fields in J0 and J1 must be loaded with effective values of  $\pm 1wb$ , depending on mode and width.
3. The desired fill byte must be loaded in SR2<sub>17-0</sub>.

#### 5.14.2. Byte Move With Translate - BMT 33,01

Translate and transfer LJ1 bytes from source string to receiving string. Truncate or fill.

This instruction translates and transfers LJ1 byte from byte string SJ0 to byte string SL1. The translation and transfer process uses byte string SJ2 as a translation table for byte string L0. That is, each byte of the string SJ0 is used as an index to a byte in string SJ2. The SJ2 byte thus addressed is transferred to the byte string SJ1. If the value of LJ1 is less than LJ0, the transfer terminates when LJ1 bytes have been processed. If the value LJ1 is greater than LJ0, then (LJ1-LJ0) translated fill bytes are placed in the trailing positions of SJ1. The contents of SR2<sub>17-0</sub> are used to index the fill byte.

When byte strings of different byte size are transferred, the receiving string determines how many bits from each byte of the source string will be accepted. If SJ1 is in the six-bit mode and SJ2 is in the nine-bit mode, only the six least significant bits of SJ2 byte are accepted, the rest being lost.

The translation table pointer register, J2, must be in the 9- or 18-bit mode. This restriction does not prevent FIELDATA translations, but it requires that the translation table bytes are either 9- or 18-bit entries. Both the values LJ1 and LJ0 are decreased by one following each byte translation. When the value of LJ1 is equal to zero ( $LJ1 = 0$ ), the instruction is terminated.

The fill byte referenced by  $SR2_{17-0}$  must be preloaded left shifted one bit if  $MW = 0$  in J0 (see Figure 5-1), indicating the source string is 9-bit bytes, and must be preloaded left shifted two bits if  $BL = 1$  in J0, indicating the source string is 18-bit bytes.

1. This instruction is interruptible after each byte store operation.
2. The lwb fields of J0 and J1 must be loaded with effective values of  $\pm$ , depending on mode and width (see Table A3).

### 5.14.3. Byte Translate and Compare – BTC 33,03

Optionally, translate and compare LJ0 bytes from SJ0 with LJ1 bytes from SJ1; terminate the instruction on not equal or when both LJ0 and LJ1 equal zero; when:

(A) > 0; string SJ0 > SJ1

(A) = 0; string SJ0 = SJ1

(A) < 0; string SJ0 < SJ1

This instruction optionally translates and compares LJ0 bytes of string SJ0 with the optionally translated LJ1 bytes of string SJ1. String SJ2, starting at address  $(u+(X+2)+J2_{ow})$ , is used as the translation table for strings SJ0 and SJ1 when the corresponding E bit is zero. A one in the corresponding E bit inhibits translation. Thus a translation can be made on either or both strings. If no translation is desired, the Byte Compare instruction (33,04) should be used. The comparison is made by subtracting the optionally translated SJ1 byte from the optionally translated SJ0 byte and storing the result in register Aa. If the contents of Aa is zero ( $Aa = 0$ ), then the next pair of bytes are translated or not, according to the content of (E) and compared. If the contents of Aa is not zero ( $Aa \neq 0$ ), or if both of the strings SJ0 and SJ1 have a value of  $LJ1 = 0$  and  $LJ0 = 0$ , then the instruction is terminated. The values of LJ1 and LJ0 are always decreased by one, and the J0- and J1-registers are increased or decreased by one, depending upon the direction addressed.

When the instruction termination occurs, the relative value of an SJ0 string in respect to the value of an SJ1 string may be determined as follows:

- If the contents of the Aa-register is positive ( $A > 0$ ), then the SJ0 string is greater than the SJ1 string (after optional translations).
  - If the contents of the Aa-register is zero ( $Aa = 0$ ), then the SJ0 string is equal to the SJ1 string (after optional translations).
  - If the contents of the Aa-register is negative ( $Aa < 0$ ), then the SJ0 string is less than the SJ1 string (after optional translations).
1. If either string SJ0 or string SJ1 is depleted before the other, trailing fill characters are added to the shorter string.
  2. The fill byte for string SJ0 is contained in  $SR1_{17-9}$  and the fill byte for string SJ1 is contained in  $SR2_{35-18}$ .

3. The fill bytes in SR2 must be preloaded left shifted one bit if  $MW = 0$  in J0 indicating the source string is 9-bit bytes and must be preloaded left shifted two bits if  $MW = 1$  in J0 indicating the source string is 18-bit bytes (see Figure 5-1).

#### 5.14.4. Byte Compare – BC 33,04

Compare LJ0 bytes from string SJ0 with LJ1 bytes from string SJ1; terminate instruction on not equal or when both LJ0 and LJ1 are zero.

The corresponding string SJ1 byte is subtracted from the string SJ0 byte, the result is stored in Aa, and a zero test is performed. The value of LJ1 and LJ0 are always decreased by one, and the J0- and J1-registers are updated. If the contents of the Aa is zero, the next pair of bytes are tested. If the value of Aa is nonzero, or both LJ1 and LJ0 are zero (i.e., the longer string has been depleted), the instruction is terminated.

When the instruction termination occurs, the relative value of an SJ0 string in respect to the value of an SJ1 string may be determined as follows:

- If the contents of the Aa-register is positive ( $Aa > 0$ ), then the SJ0 string is greater than the SJ1 string.
  - If the contents of the Aa-register is zero ( $Aa = 0$ ), then the SJ0 string is equal to the SJ1 string.
  - If the contents of the Aa-register is negative ( $Aa < 0$ ), then the SJ0 string is less than the SJ1 string.
1. If either string SJ0 or string SJ1 are depleted before the other, trailing fill characters are added to the shorter string. The fill byte for the string SJ0 is contained in SR2<sub>17-0</sub> and the fill byte for string SJ1 is contained in SR2<sub>35-18</sub>.
  2. This instruction is interruptible after each compare.
  3. The lwb-fields of J0 and J1 must be loaded with effective values of  $\pm 1wb$ , depending on mode and width (see Table 6-2).

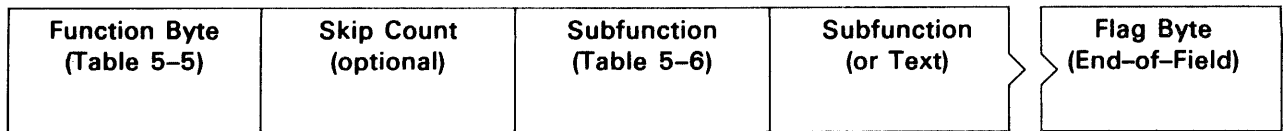
#### 5.14.5. Edit – EDIT 33,07

Edit byte string SJ0 and transfer to string SJ1 under the control of string SJ2.

A source byte string (string SJ0) specified by the u-field of the instruction, utilizing registers Xx and J0, are edited into a receiving byte string (string SJ1) specified by the u-field of the instruction, Xx + 1, and J1. Specific editing commands are coded within a control byte string (string SJ2) whose location is designated by the J2 register, the u-field of the instruction, and the register Xx + 2. The control stream commands are designed to duplicate all of the functions of the PICTURE clause of the COBOL compiler. Therefore, the main use of the Edit instruction is to make the appropriate editing changes to a numeric byte string for output to the printer. For example, blanking-out the leading zeros, adding a "\$" character or the appropriate sign code, inserting commas or a decimal point within the number, or appending a descriptor word such as "CR" or "DB".

The following information describes and summarizes the basic operational steps of the Edit instruction.

A typical field in the control stream, (string SJ2) will contain the following elements:



The function byte specifies control information for the whole field following it (see Table 5-5). One function of this byte is to specify whether there is a skip count or not. If there is a skip count it is given in the next byte. The rest of the field contains a series of subfunction bytes and text bytes. The subfunction bytes are those described in Table 5-6 and specify operations to be performed as the source string is edited into the receiving string. The text bytes are bytes similar to the source string bytes which may be edited into the receiving string. The last subfunction byte in the field is the flag byte which establishes the end-of-file action. The flag byte may be followed by another field starting with a function byte, or a second flag byte indicating termination of the Edit instruction.

Operation of the Edit instruction is based on performing a sequence of field "microprograms" defined by the control string. A field scan is established when the instruction is initiated or when the initiation of a new field occurs and results in certain "function initiation" actions based on the contents of the function byte. The first control stream byte must be the function byte. It will be stored in staging register SR1<sub>26-18</sub>. If a skip count is required, as indicated by the function byte, the second control stream byte contains the skip count and is transferred from the 0 control stream to staging register SR1<sub>17-9</sub>. Next, the J1-register is saved in J3. This saves the position of the first byte of the receiving string for use if the "blank-if-zero" command (bit 0 of function byte is set) is required when the end of the field is encountered. Finally, the skip count (SR1<sub>17-9</sub>) is used to skip the indicated number of bytes in the receiving string. This is done by updating J1 position 0wb as many times as the value in SR1<sub>17-9</sub> (skip count). At this point the edit is established.

When the function initiation actions are completed for this field, the first subfunction byte is transferred from the control stream to SR1<sub>8-0</sub> for interrogation. The subfunction byte is transferred from the control stream to SR1<sub>8-0</sub> for interrogation. The subfunction and text bytes are sequentially interrogated until a "flag" (end-of-field) subfunction byte is encountered. At this point the end-of-field action is completed and another function is initiated. This process continues until two "flag" bytes are encountered together indicating termination of the instruction. A detailed description of the function byte, subfunction, and text bytes follows.

### 5.14.5.1. Function Byte

The interpretation of the function byte is given in Table 5-5. A more detailed description of each bit position follows:

Table 5-5. Function Byte Interpretation

Function Byte		
Bit	0	1
5	No Skip Count	Skip Count Follows
4	Fixed Sign	Floating Sign
3	Fixed Symbol	Floating Symbol
2	Sign = Minus or Fill	Sign = Minus or Plus
1	Edit - No Sign Action	Sign Action on Edit

Table 5-5. Function Byte Interpretation (continued)

Function Byte		
0	Normal Edit	Blank if Zero

- **Function Bit 5** – A skip mechanism is included that allows the programmer the option of ignoring a series of bytes in the receiving string. The skip count is placed in position SR1<sub>17-9</sub> during function initiation and specifies the number of bytes to be skipped in the receiving string before the first subfunction byte is interrogated. The maximum value allowed is 63<sub>10</sub> for either the 6 or 9 bit mode.
- **Function Bit 4** – If fixed sign is indicated, an appropriate sign byte (as specified by function bit 2) is placed in the receiving string position specified by SR2<sub>35-18</sub>. SR2<sub>35-18</sub> must be loaded by the subfunction "sign-position indicator" discussed in 5.14.5.2. If floating sign is indicated, an appropriate sign byte is placed in the receiving string position specified by SR2<sub>17-0</sub>. SR2<sub>17-0</sub> is loaded by the subfunctions "digit select" or "significance start indicator" as discussed in 5.14.5.2. In either case, a fill byte is transferred to the receiving string where the sign will be. The sign bytes are specified by the programmer in SR3 and are transferred to the receiving string when a "flag" subfunction byte is interrogated (end-of-field action). This bit has no meaning unless function bit 2 (sign action on edit) is set to 1.
- **Function Bit 3** – If this bit is a 1 bit, the symbol specified by the programmer in SR3<sub>8-0</sub> is transferred to the receiving string at the position specified by SR2<sub>17-0</sub>. Position SR2<sub>17-0</sub> is loaded the same way as for "floating sign" above. If both function bits 3 and 4 are set to one, the floating sign will be inserted and the floating symbol ignored. As with the sign codes the symbol is actually inserted during end-of-field action. If function bit 3 is a zero there is no symbol inserted during end-of-field action. A symbol may still be inserted into the receiving string during subfunction interrogation with a "symbol-position indicator" subfunction (described in 5.14.5.2).
- **Function Bit 2** – If function bit 2 is a 0, the sign code inserted into the receiving string is either a minus or a fill as appropriate. If function bit 2 is a 1 bit, the sign code is either a minus or a plus. The plus, minus, and fill bytes are specified by the programmer in SR3<sub>17-9</sub>, SR3<sub>26-18</sub>, or SR3<sub>35-27</sub>, respectively. This bit has no meaning unless function bit 1 (sign action on edit) is set to one.
- **Function Bit 1** – If function bit 1 is a 1, the sign action indicated by bits 4,2 and the sign of the source string is taken (i.e., a plus, minus, or fill byte is inserted into the receiving string). If bit 1 is zero, no plus or minus bytes are inserted into the receiving string. The only effect bit 4 (floating sign) would have is to insert a fill byte in the position where the floating sign byte should be.
- **Function Bit 0** – The programmer has the option of leaving an all-zero receiving field or replacing it with fill bytes. The field is considered to be all-zero until a nonzero byte has been transferred from the source string by a "digit select" subfunction. Position SR1<sub>31</sub> is set to one when the first nonzero digit is transferred. SR1<sub>31</sub> is not set to one by any subfunction other than "digit select". The entire receiving string field is replaced with fill bytes during end-of-field action (see "flag" subfunction in 5.14.5.2) if function bit 0 and SR1<sub>31</sub> are both set to 1. The start and end of the field are indicated by J3 (loaded during function initiation) and J1, respectively.

### 5.14.5.2. Subfunction Byte

The interpretation of the subfunction byte is given in Table 5-6 and discussed in the following paragraphs.

Table 5-6. Subfunction Byte Interpretation

Byte	Function	Fielddata Symbol
000 000 011	Pass Byte	#
000 101 111	Significance Start Indicator	\
000 100 110	Digit Select	&
000 111 110	Symbol Position Indicator	□
000 111 010	Sign Position Indicator	' (apostrophe)
000 101 001	Trailing Text Start Indicator	(
000 111 111	Flag (End-of-Field)	t b

- **Pass Byte** – If the control byte is a "pass byte", the byte currently pointed at by the source field pointer is transferred to the receiving string intact.
- **Significance Start Indicator** – If the control byte is a "significance start byte," the "significance trigger" SR1<sub>34</sub> is set to one. Also, if either floating sign (function bit 4 set) or a floating symbol (function bit 3 set) will be require, then the receiving field pointer, J1-register, is stored in SR2<sub>17-0</sub>, and a fill byte SR3<sub>35-27</sub> is inserted in the receiving field. This fill byte is replaced by the appropriate sign byte in end-of-field processing as indicated by the function byte bit settings. If the "significant trigger" has already been set to one, this byte is ignored.
- **Digit Select** – If the control byte is a "digit select byte," the byte currently pointed at by the source field pointer and the significance trigger are examine, according to the following criteria:
  1. If the significance trigger is off, and the source byte has a zero digit portion, the receiving field will have a fill character (SR3<sub>35-27</sub>) inserted into it.
  2. If the significance trigger is off and the source byte is not a zero,
    - a. the significance trigger is set on, and
    - b. if either floating sign or floating symbol will be required, the receiving field pointer (J1) is stored in SR2<sub>17-0</sub>, and a fill byte, (SR3<sub>35-27</sub>) is inserted in the receiving string. This fill byte is replaced by the appropriate sign byte in end-of-field processing as indicated by the function byte bit settings. The receiving field pointer (J1) is incremented, and
    - c. the source byte is transferred to the receiving string.
  3. If the significance trigger is "on," then the source byte is transferred to the receiving field.
  4. If this is the first nonzero digit to be transferred from the source string, SR1<sub>31</sub> is set to one for use when interrogating the "blank-if-zero" function bit in end-of-field processing.
  5. The appropriate zone code, as prescribed by the receiving string pointer (J1), is always written into the receiving string.

An exception to the above "digit-select" transmission exists if the zone portion of the byte is negative sign (sign overpunch). In this case, the "N" bit is turned "on" (SR1<sub>32</sub>) and the negative sign bits are replaced by the appropriate zone code.

- **Symbol Position Indicator** – If the control byte is a "symbol position indicator", the symbol byte (SR3<sub>8-0</sub>) is stored in the receiving field. The setting of function bit 3 does not affect the operation of this subfunction byte.
- **Sign Position Indicator** – If the control byte is a "sign position indicator," the receiving field pointer, contained in the J1-register, is copied into the fixed-sign position pointer (SR2<sub>35-18</sub>) and the fill character (SR3<sub>35-27</sub>) is transmitted to the receiving field. This fill byte is replaced by the appropriate sign byte in end-of-field processing as indicated by the function byte bit settings. This subfunction byte must be used to indicate the position of the fixed sign if the function bit 4 is 0 (fixed sign).
- **Trailing Text Start Indicator** – If the control byte is a "trailing text start" byte, the trailing text trigger (SR1<sub>33</sub>) is set to one. If a negative sign has been detected in the source string scan (SR1<sub>32</sub> set to one), any text information encountered in the control string is now transferred to the receiving string. If SR1<sub>32</sub> equals zero, fill bytes (SR3<sub>35-27</sub>) are transferred to the receiving field rather than the text bytes.
- **Flag Byte (End-of-Field)** – If the control byte is a "flag" byte, then the end-of-field action will be established. At this point the appropriate sign insertion and "blank-if-zero" command actions are done as indicated by the function byte. The net control stream byte is either a function byte starting a new field or another flag byte terminating the Edit instruction.

If the control byte is none of the subfunction bytes of Table 5-6, it is assumed to be a text byte. If either SR1<sub>33</sub> and SR1<sub>32</sub> are set (see Table 5-7) or SR1<sub>34</sub> is set and SR1<sub>33</sub> is not set, the text will be transferred to the receiving field. In all other cases, the fill byte (SR3<sub>35-27</sub>) will be transferred to the receiving field.

A summary of staging register (SR1-SR3) and J-register (J0-J3) usage are given in Table 5-7.

1. The lwb-fields of J0, J1, and J2 must be loaded with effective values of +1wb, depending on values of +1wb, depending on mode and width (see Table 5-2).
2. SR3 must be loaded with the desired codes.
3. This instruction is interruptible.

#### 5.14.6. Byte to Binary Single Integer Convert – BI 33,10

Convert LJ0 byte in string SJ0 into a signed binary integer in register A.

This instruction converts byte string SJ0 composed of LJ0 bytes, coded in either ASCII or Fieldata, into a signed binary integer in the Aa-register. The J0-register initially points to the leftmost byte in the string and is set for left-to-right incrementation. The sign must be represented in the zone of the least significant byte.

Table 5-7. Summary of Staging Register and J-Register Fields

Field	Position	Function
CMP (Complement Mode)	SR1 <sub>35</sub> (BT0)	No complement if 0, complement if 1.
ST (Significance Trigger)	SR1 <sub>34</sub> (BT1)	set to 1 if the control byte is a "significant start: byte.
T (Trailing Text Trigger)	SR1 <sub>33</sub> (BT2)	Set to 1 when a trailing text start indicator has been detected.
N (Negative Bit)	SR1 <sub>32</sub> (BT3)	Set to 1 when a negative sign has been detected.
C (Control Bit)	SR1 <sub>31</sub> (BT4)	Set to 1 when the first nonzero digit is transferred from the source string by the "digit select" subfunction.
L (Skip Bit)	SR1 <sub>30</sub> (BT5)	Set to 1 if a skip is in progress.
I (Interrupt Bits)	SR1 <sub>29-27</sub> (BT6-8)	Controls return after instruction interrupt.
Function	SR1 <sub>26-18</sub> (BS2)	Contains active Edit field function.
Skip Count	SR1 <sub>17-9</sub> (BS3)	Contains skip count to bypass receiving field.
Subfunction	SR1 <sub>8-0</sub> (BS4)	Contains active Edit subfield function or text.
Fixed-Position Pointer	SR2 <sub>35-18</sub> (BH0)	Acts as index modifier for pointing to byte in receiving string which will receive fixed sign or symbol. Receives contents of J1-register position Owb.
Floating-Position Pointer	SR2 <sub>17-0</sub> (BH1)	Acts as index modifier for pointing to byte in receiving string which will receive floating sign or symbol. Receives contents of J1-register position Owb.
Fill Byte	SR3 <sub>35-27</sub> (BBO)	Byte used when fill is called for.
Negative-Sign Byte	SR3 <sub>26-18</sub> (BB1)	Byte used when negative sign insertion is specified.
Plus-Sign Byte	SR3 <sub>17-9</sub> (BB2)	Byte used for positive sign insertion.
Symbol Byte	SR3 <sub>8-0</sub> (BB3)	Byte used for symbol insertion.
Source Pointer	J0	Points at source byte for Edit action.
Receiving Pointer	J1	Points at byte to receive edited byte.
Control Pointer	J2	Acts as index modifier for pointing to control string (byte).
Start-of-Field Pointer	J3	Copy of contents of J1-register at start of field. Used to control blank-if-zero action.



A 7-bit status word is stored in the low-order bits of  $SR3_{17-9}$ . An all zero word indicates successful completion of the instruction. Bit 0 set indicates a Format error and is set if one of the input bytes is not a digit or a blank (checked on all but the last byte) or the least significant 4 bits of the last byte are greater than 9. Bit 1 set indicates an overflow condition and is set if the magnitude of the input string SJO is too large to be represented by 35 binary bits.

1. If the arithmetic section detects a register overflow, an interrupt (to  $MSR + 250_g$ ) is generated.
2. The *lwb*-field of JO must be loaded with effective value of +1 depending on mode and width (see Table 5-2).
3. J1, J2 and J3 are not used in this instruction.

#### 5.14.7. Byte to Binary Double Integer Convert – BDI 33,11

Convert LJO bytes in string SJO into a signed binary integer in registers A and A+1.

This instruction converts byte string SJO composed of LJO bytes, coded in either ASCII or Fieldata, into a signed binary integer in the Aa- and Aa+1-register. The JO-register initially points to the leftmost byte in the string and is set for left-to-right incrementation. The sign must be represented in the zone of the least significant byte.

A 7-bit status word is stored in the low-order bits of  $SR3_{17-9}$ . An all zero word indicates successful completion of the instruction. Bit 0 set indicates a Format error and is set if one of the input bytes is not a digit or a blank (checked on all but the last byte) or the least significant 4 bits of the last byte are greater than 9. Bit 1 set indicates an overflow condition and is set if the magnitude of the input string SJO is too large to be represented by 72 binary bits.

1. If the arithmetic section detects a register overflow, an interrupt (to  $MSR + 250_g$ ) is generated.
2. The *lwb*-field of JO must be loaded with effective value of +1, depending on mode and width (see Table 5-2).
3. J1, J2 and J3 are not used in this instruction.

#### 5.14.8. Binary Single Integer to Byte Convert – IB 33,12

Convert the binary integer in A to byte format and store in string SJO.

This instruction converts the binary integer contained in the Aa-register to a byte format and stores the results in string SJO. String SJO is LJO bytes long and the rightmost byte has the sign in the zone portion.

The converted number is right-justified and zero-filled in the string. If string SJO is not long enough to accommodate the converted number, the remaining bytes will be truncated. The JO-register must be set for negative incrementation and point to the rightmost byte.

1. The *lwb*-field of JO must be loaded with effective value of -1, depending on mode and width (see Table 5-2).
2. J1, J2 and J3 are not used in this instruction.

#### 5.14.9. Binary Double Integer to Byte Convert - DIB 33,13

Convert the binary integer in A and A+1 to byte format and store in string SJO.

This instruction converts the binary integer contained in the Aa- and A+1-registers to a byte format in string SJO. String SJO is LJO bytes long and the rightmost byte has the sign in its zone portion.

The converted number is right-justified and zero filled in the string. If string SJO is not long enough to accommodate the converted number, the remaining bytes will be truncated. The JO-registers must be set for negative incrementation and point to the rightmost byte of string SJO.

1. lwb-field of JO must be loaded with effective value of -1, depending on mode and width (see Table 5-2).
2. J1, J2 and J3 are not used in this instruction.

#### 5.14.10. Byte to Single Floating Convert - BF 33,14

Convert LJO bytes in string SJO into a single-length floating-point format in register A.

This instruction converts byte string SJO composed of LJO bytes, coded in either ASCII or Fieldata, into a single floating-point number in register Aa.

String SJO may have either a leading plus-sign character (+) or a leading minus-sign character (-) that indicates the sign of the mantissa. In the absence of either a plus or minus sign, the mantissa is assumed to be positive. The mantissa must be representable in 30 binary bits if interpreted as an integer and may or may not contain a decimal point character. If the mantissa does not contain a decimal point character, this character is assumed and its position is contained in SR3<sub>26-18</sub>. If the decimal point character is present, this condition overrides the effect of SR3<sub>26-18</sub>.

The exponent, if present, follows the least significant digit of the mantissa. An exponent is indicated by an E or D character followed by a minus sign and then the digits, if the exponent is negative. If the exponent is positive the E or D character is followed by the digits alone or by a plus sign followed by the digits. The E or D character may be optionally omitted. In this case, either a plus sign or a minus sign must precede the digits of the exponent. The exponent must be limited to two digits. If an exponent is not present, 10<sup>0</sup> will be assumed.

A 7-bit status word is stored in the low-order bits of SR3<sub>19-0</sub>. An all zero word indicates successful completion of the instruction. For possible error conditions and status word indications see Table 5-3.

1. Floating-point interrupts (characteristic underflow/overflow) may occur during the instruction.
2. lwb-field of JO must be loaded with effective value of +1, depending on mode and width (see Table 5-2).
3. J1, J2 and J3 are not used in this instruction.
4. SR3 positions 26-18 are the number of digits to the right of the decimal point.

#### 5.14.11. Byte to Double Floating Convert – BDF 33,15

Convert LJO bytes in string SJO into a double-length floating-point format in registers A and A+1.

This instruction converts byte string SJO composed of LJO bytes, coded in either ASCII or Fieldata, into a double-precision floating-point number in registers Aa and Aa+1.

The SJO string may have either a leading plus-sign character (+) or a leading minus-sign character (-) that indicates the sign of the mantissa. In the absence of either a plus or minus sign, the mantissa is assumed to be positive. The mantissa must be representable in 60 binary bits if interpreted as an integer and may or may not contain a decimal point character. If the mantissa does not contain a decimal point character, this character is assumed and its position is contained in SR3<sub>26-18</sub>. If the decimal point character is present, this condition overrides the effect of SR3<sub>26-18</sub>.

If the exponent is present, it follows the least significant digit of the mantissa. The exponent is formed according to the same rules that apply to the Byte to Single Floating-Point instruction (see 5.14.10), except that there may be up to three digits in the exponent.

A 7-bit status word is stored in the low-order bits of SR3<sub>19-7</sub>. An all zero word indicates successful completion of the instruction. For possible error conditions and status word indications see Table 5-3.

1. Floating-point interrupts (characteristic underflow/overflow) may occur during the instruction.
2. lwb-field of JO must be loaded with effective value of +1, depending on mode and width. (See Table 5-2.)
3. J1, J2 and J3 are not used in this instruction.
4. SR3 position 26-18 is the number of digits to the right of the decimal point.

#### 5.14.12. Single Floating to Byte Convert – FB 33,16

Convert the single-length floating-point number in A to byte format and store in string SJO.

This instruction converts a single-length floating-point number contained in the Aa-register to a byte string starting at address SJO. The format of the resulting string SJO contains two numbers. The first number is a nine-byte decimal fraction that has its sign in the zone part of the least significant byte. The second number is a two-byte exponent with its sign in the zone portion of the least significant byte.

1. the lwb-field of JO must be loaded with effective value of +1, depending on mode and width. (See Table 5-2.)
2. J1, J2 and J3 are not used in this instruction.

#### 5.14.13. Double Floating to Byte Convert – DFB 33,17

Convert the double-length floating-point number in A and A+1 to byte format and store in string SJO.

This instruction converts a double-length floating-point number contained in the Aa- and Aa+1-registers to a byte string starting at address SJO. The format of the resulting string is similar to that of the Single Floating to Byte Convert (see 5.14.12.) instruction except that the first number

is equivalent to an 18-byte string and the second number is equivalent to a three-byte string. The first number is the mantissa and the second number is the exponent. Each number has its sign in the zone portion of the least significant byte.

1. The lwb-field of J0 must be loaded with effective value of +1, depending on mode and width. (See Table 5-2.)
2. J1, J2, and J3 are not used in this instruction.

#### 5.14.14. Byte Add - BA 37,06

Add the LJ0 bytes in string SJ0 to the LJ1 bytes in string SJ1 and place the results in string SJ2.

This instruction adds byte string SJ0 (of length LJ0) to byte string SJ1 (of length LJ1) and stores the results in byte string SJ2 (of length LJ2). Only 6-bit Fielddata or 9-bit ASCII formats may be used. The sign of the SJ0 and SJ1 strings must be stored in the zone portion of the least significant byte. If the length of the resultant byte string is smaller than LJ2 digits, then the SJ2 string will be zero filled. The J-registers must point to the least significant digit and should be set for right-to-left incrementation.

1. This instruction is interruptible.
2. The lwb-fields of J0, J1, and J2 must be loaded with the effective value of -1, depending on mode and width. (See Table 5-2.)

#### 5.14.15. Byte Add Negative - BAN 37,07

Subtract the LJ0 bytes in string SJ0 from the LJ1 bytes in string SJ1 and place the results in string SJ2.

This instruction subtracts byte string SJ0 (of length LJ0) from byte string SJ1 (of length LJ1) and stores the results in byte string SJ2 (of length LJ2). Only 6-bit Fielddata or 9-bit ASCII format may be used. The sign of the SJ0 and SJ1 strings should be stored in the zone portion of the least significant byte. If the length of the resultant byte string is smaller than LJ2 digits, then string SJ2 will be zero filled. The J-registers must point to the least significant digit and should be set for right-to-left incrementation.

1. This instruction is interruptible.
2. The lwb-fields of J0, J1, and J2 must be loaded with effective value of -1, depending on mode and width. (See Table 5-2.)

### 5.15. EXECUTIVE INSTRUCTION REPERTOIRE

The instructions in this group are intended for use by the Executive system. When designator register bits 35 and 2 are zero, the Executive repertoire is selected. This allows execution of all Executive (privileged) instructions in addition to those of the user repertoire. The Executive repertoire includes instructions for control of the processor state, interrupts, input/output, and instrumentation.

The Executive control instructions defined for the processor are described in the following paragraphs. They are listed in Appendix B.

### 5.15.1. Prevent All Interrupts and Jump – PAIJ 72,13

The processor will not recognize certain interrupt requests received following the completion of the instruction nor will it react to interrupt requests received following the start of the execution of the instruction.

The following interrupts may be prevented by this instruction:

- All I/O interrupts, including those for normal status, tabled status, and machine check interrupts.

This instruction causes the internal dayclock register value of the processor to be replaced at the start of the next update cycle with the value in the dayclock location in fixed storage.

### 5.15.2. Enable/Disable Dayclock – EDC,DDC 73,14, 11-12

These instructions enable and disable, respectively, the internal dayclock of the processor. When a dayclock is enabled, if the dayclock is also selected the dayclock value is stored in the dayclock location in fixed storage during each update cycle, and a dayclock interrupt request may be generated by the dayclock.

### 5.15.3. Select Dayclock – SDC 73,14, 13

Each processor contains an internal dayclock. One dayclock in each application may be selected at any given time to store its value in the dayclock location in fixed storage. The operand address of the instruction specifies the processor number whose dayclock is selected for this function; note that the selected dayclock must also be enabled (via EDC).

### 5.15.4. Select Interrupt Locations – SIL 73,15, 00

Bits 22 through 16 of the operand are transferred to the module select register (MSR) specified by bit 23. If bit 23=1, transfer is to MSR in SIU upper half; if bit 23=0, transfer is to MSR in SIU lower half.

MSR is used as the base for all fixed address assignment references, and there is a separate MSR for the lower half of the addressing range (0-8M) and the upper half of the addressing range (8-16M); the load path selection of the system transition unit determines which is to be used for fixed references.

### 5.15.5. Load Breakpoint Register – LBX 73,15, 02

The operand specified by the operand address is transferred to the breakpoint register. This establishes the modes of operation for the breakpoint mechanism, and activates and establishes the modes of operation for the jump history stack.

### 5.15.6. Store Processor ID – SPID 73,15, 05

The binary serial number is stored in the first third of the operand, the two character fielddata revision level is stored in the second third of the operand, the processor features provided are stored in the fifth sixth of the operand (bit 8 byte-oriented instructions, bit 7 floating-point instructions) and the binary processor number is stored in the last sixth of the operand.

**5.15.7. Load Quantum Timer - LQT 73,15, 03**

The full-word operand specified by the operand address is placed in the quantum timer.

**5.15.8. Load Base - LB 73,15 10**

Bits 17 through 0 of the operand specified by the operand address are placed in the base-value field of the bank descriptor register specified by bits 34 and 22 of Xx. If the x-field of the instruction is zero, BDRO is implicitly specified.

**5.15.9. Load Limits - LL 73,15, 11**

Bits 35 through 24 and 23 through 15 of the operand specified by the operand address are placed in the upper and lower limits fields, respectively, of the bank descriptor register specified by bits 34-33 of Xx. If the X-field of the instruction is zero, BDRO is implicitly specified.

**5.15.10. Load Addressing Environment - LAE 73,15, 12**

The double-word operand specified by the operand address contains four bank descriptor specifications in the following format:

E 0	XX	ign- ored	BDI0	E 2	XX	ign- ored	BDI2
E 1	XX	ign- ored	BDI1	E 3	XX	ign- ored	BDI3
35 34 33 32 30 29				18 17 16 15 14 12 11			0

This operand is placed in GRS locations 046 and 047, and the limits and base values of the four bank descriptors specified by this operand are placed in the respective bank descriptor registers. The bank descriptor table length check is not performed on the bank descriptor index supplied by the instruction. Bank descriptor flags and use counts are neither interpreted nor altered by LAE.

**5.15.11. Store Quantum Time - SQT 73,15, 13**

The current value of the quantum timer is stored at the operand address, which may be in GRS or storage. Execution of this instruction has no effect on D29.

**5.15.12. Load Designator Register - LD 73,15, 14**

The full-word operand specified by the operand address is placed in the designator register. All designator register specifications are in effect at the completion of this instruction.

**5.15.13. Store Designator Register - SD 73,15, 15**

The contents of the designator register is stored at the location by the operand address.

**5.15.14. User Return - UR 73,15, 16**

This instruction provides an orderly mechanism for returning to a user program. The instruction effectively combines LD and jump except that the component operations are performed with the correct repertoire, addressing, and register set.

The double-word operand specified by the operand address contains the relative program address and designator register value that establish the user operating state.

The second word of the operand is placed in the designator register, and all specifications are put in effect. The lower 24 bits of the first word of the operand then becomes the relative program address. If the relative program address is subsequently found to be out of limits, the interrupt will capture the new P-value.

Bit positions 23 through 18 of the relative address and the A-flag (bit position 35 of the same word) should be zero, unless base register suppression ( $D35 = 0, D7 = i = 1$ ) is intended or was in effect when the address was stored as the result of an interrupt.

**5.15.15. Reset Auto-Recovery Timer - RAT 73,15, 06**

This instruction resets the timer in the auto-recovery section of the partitioning unit. This must be done within an interval specified by the auto-recovery design in order to prevent an automatic initial load from being initiated.

**5.15.16. Toggle Auto-Recovery Path - TAP 73, 5, 07**

The system allows for two auto-recovery paths (processor/IOU/SIU Half combinations). Each time an auto-recovery is attempted, the path selection is toggled. When a successful recovery does occur, this instruction allows the software to return the auto-recovery selection to the successful path.

**5.15.17. Store System Status - SSS 73,15, 17**

This instruction stores two words of system status at the location specified by the operand address. System status includes partitioning information relating to each processor IOU, SIU, and MSU.

**5.15.18. Diagnostics - 73,14, 14 - 17**

These instructions are provided to test a large portion of the arithmetic hardware and a smaller portion of the control section hardware. They generate specific operands, cause arithmetic to exercise its logic, and place results in GRS. The results may be then tested via TE/TNE instructions to verify operation. If an error is found it is recommended that diagnostic procedures using scan and/or test routines be used. Detail instructions are:

MDA 73, 14, 14 - Generates A and A+1 operands of 0707—07, a U operand of 070707070622, and a U+1 operand of 2525—25. The result,  $A=00372706711, A+1=256171354400$  is stored in GRS addresses 62 and 63.

MDB 73, 14, 15 - Generates an A operand of 0707—07, an A+1 operand of 070707070701, a U operand of 070707070600, and a U+1 operand of 0707—07. The result, A=771177117711, A+1=777777776677 is stored in GRS addresses 62 and 63.

73, 14, 16, and 17 - are undefined and will not operate.

### 5.15.19. Input/Output Instructions

The I/O instructions are described in detail in Section 6.

For each I/O instruction, the operand address specifies the IOU, channel, and device number, if applicable. For certain instructions, the index register specified by the a-field of the instruction (Xa) contains a parameter associated with the operation, generally an address. Each I/O instruction skips the next instruction if the operation was initiated properly, and a condition code of zero is stored in the upper sixth-word of register Xa.

If the next instruction is executed, the upper sixth-word of register Xa contains a code that describes one of four conditions: 000 = operation initiated, 020 = status is available, 040 = busy, and 060 = not operational; the remainder of Xa is not disturbed.

The Input/Output instructions are:

- Sense Release (SRL-75,00)
- SIO Fast Release (SIOF-75, 11): Xa contains the first CCW address.
- Test I/O (TIO-75, 02)
- Test Subchannel (TSC-75, 03)
- Halt Device (HDV-75, 04)
- Halt Channel (HCH-75, 05)
- Load Channel Register (LCR-75, 10): Xa contains the value to be loaded.
- Load Table Control Words (LTCW-75, 11): Xa contains the first CCW address.

#### NOTE:

*Because of the single IAW and CSW locations for a processor, I/O instructions that may alter these locations must be executed with interrupts locked out.*



## 6. Input/Output

### 6.1. INTRODUCTION

The input/output unit (IOU) provides a means of communications between the processor and its external media. Under processor control the IOU handles all transfer of data and status between peripherals and main storage. It minimizes processor involvement in input/output operations, yet provides a flexible method of controlling and interrogating input/output activity. This section describes the IOU's system philosophy, functional characteristics, various hardware and software options, and overall operation.

### 6.2. FUNCTIONAL CHARACTERISTICS

The IOU has three interfaces: a storage interface, a processor interface, and a control unit or peripheral interface. Each IOU consists of one control module and from two to eight channel modules. (See Figure 6-1.) The control module handles all the interfacing with the storage unit and either one or two processors. The control module to processor(s) interface initiates instructions and handles interrupts. The control module to storage interface transfers data, input/output control words, and status between the channel modules and the storage interface. The control module establishes a data handling and interrupt priority among the eight channel modules.

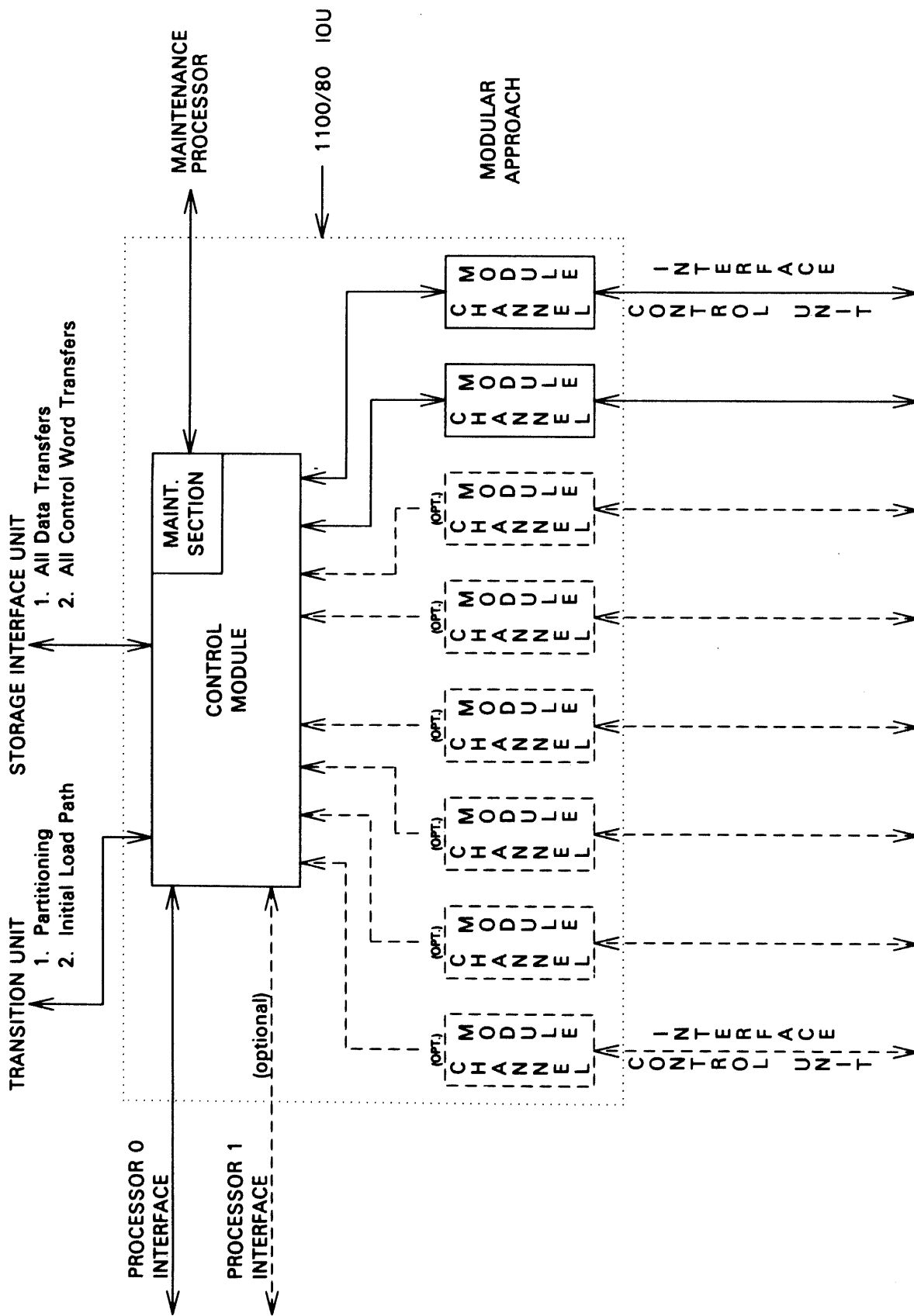


Figure 6-1. 1100/80 Input/Output Unit

### 6.2.1. Channels

A channel is defined as a channel module. The channel handles all interfacing with the control units. The channel executes input/output instructions, formats and transfers data, generates interrupts and status, and establishes priority among input/output instructions, data transfers, and interrupts. Each channel is designated by feature to be either a byte multiplexer channel, a block multiplexer channel, or an 1100 series compatible word channel.

A channel provides a standard interface for communicating with control units. A control unit provides the logical capability necessary to adapt the standard form of control provided by the channel to the characteristics of an input/output device. A control unit may be housed separately and connected to one or many devices, or it may be physically and logically integrated with an input/output device. Input/output devices provide external storage and a means of communications for a processing system. Magnetic tape units, printers, storage devices such as disks and drums, consoles, and card readers are examples of input/output devices.

Priority among devices is established by the control units. Priority among control units is determined by their physical connection to the channel. Each 1100/80 word channel has a maximum of four parallel input/output interfaces, each with an assigned priority. (See Figure 6-2.) Each byte or block multiplexer channel has one input/output interface, and many control units are physically connected to the interface, but only one control unit at a time is logically connected to the channel. (See Figure 6-2.) The channel polls the control units serially, and the highest priority control unit requiring service logically connects to the channel. A byte multiplexer connects to a control unit for the length of time to transfer one byte of data. A block multiplexer connects to a control unit for the length of time to transfer a block of data. No other device can communicate over the interface during the time a block is being transferred.

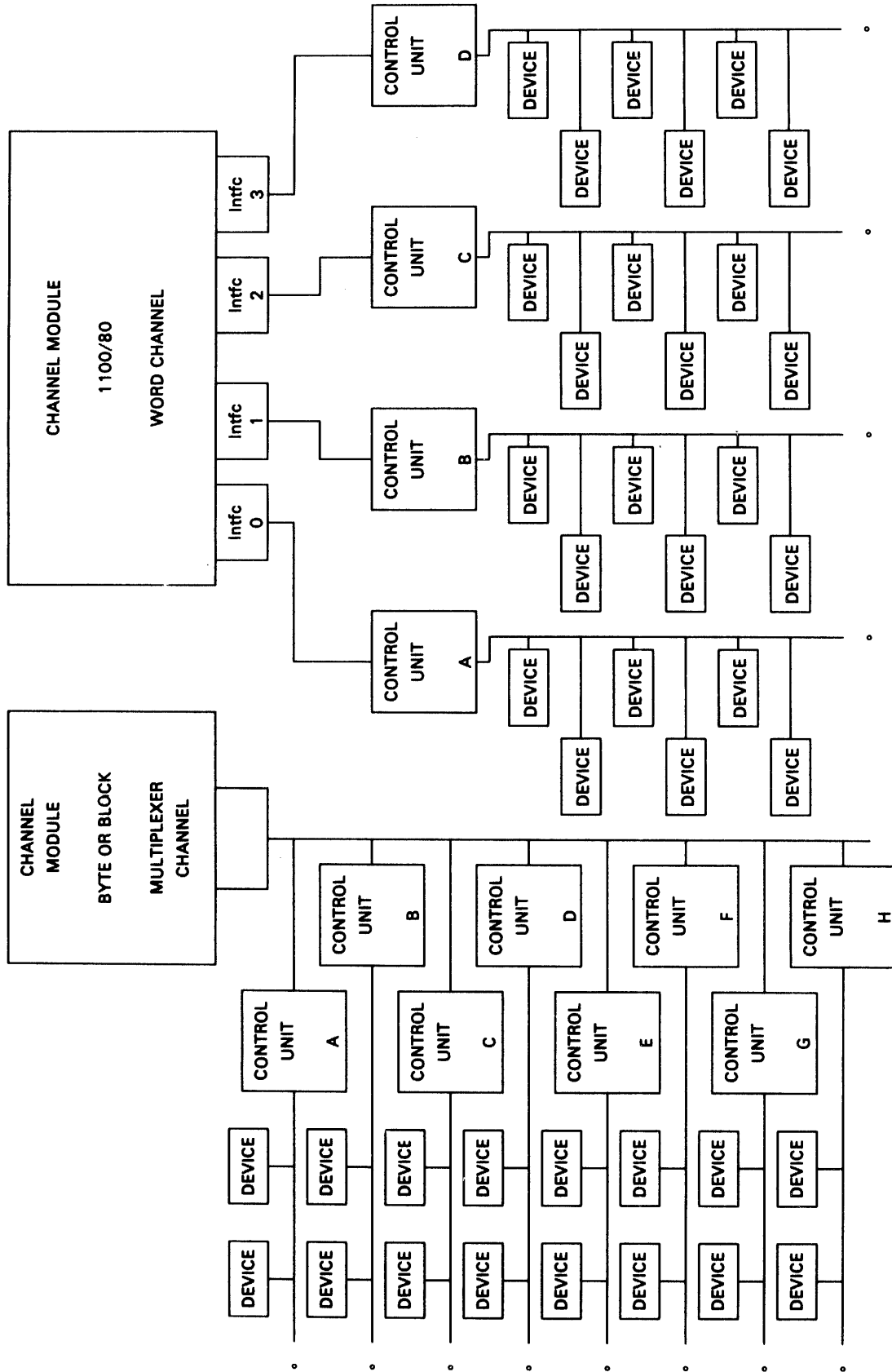


Figure 6-2. Byte or Block Multiplexer Channel Compared to 1100/80 Word Channel

### 6.2.2. Subchannels

A subchannel is defined as a set of control words that manages input/output operations. Each set of control words contains a data address, a data count, the mode of the subchannel, the storage address of the next control word, and special flags. Subchannels may be either shared or nonshared. A subchannel is referred to as shared if two or more devices use the same subchannel for input/output operations. On a shared subchannel only one device at a time can transfer data. A subchannel is referred to as nonshared if it is associated and can be used only with a single input/output device. On a 1100/80 word channel, ISI subchannels are shared and ESI subchannels are nonshared.

An IOU channel has the capability of maintaining eight resident subchannels. The basic word channel provides that all eight (four in word channel) resident subchannels are shared. In word channel modules with the subchannel expansion feature (F1654-00) and option 0 (C1655-00), there are four resident shared subchannels, four resident nonshared subchannels, and 124 nonresident nonshared subchannels. With the subchannel expansion feature and option 1 (C165501), there are eight resident nonshared subchannels and 120 nonresident nonshared subchannels. Nonresident, nonshared subchannels are kept in main storage. With the subchannel expansion feature and option 1, the eight most recently active nonshared subchannels are held in the channel. The remaining 120 subchannels are held in main storage. If the channel receives a request for a nonshared subchannel that is not resident in the channel, the least recently used resident nonshared subchannel is determined and then moved into main storage. The requested nonshared subchannel is then moved from main storage to the channel, and the request is handled. With the subchannel expansion feature and option 0, each channel has four shared subchannels and 128 nonshared subchannels. The four most recently active nonshared subchannels are kept resident in the channel, and the remaining 124 nonshared subchannels are held in main storage.

## 6.3. CONTROL OF INPUT/OUTPUT DEVICES

The processor controls I/O operations by means of eight I/O instructions: Sense Release (SRL), Start I/O Fast Release (SIOF), Test I/O (TIO), Test Subchannel (TSC), Halt Device (HDV), Halt Channel (HCH), Load Channel Register (LCR), and Load Table Control Words (LTCW). The instruction Load Channel Register addresses either the control module or a channel. The instructions Halt Channel and Load Table Control Words address only a channel; they do not address an I/O device. All other instructions address a channel and subchannel. On a byte or block multiplexer channel, the Sense Release, Test I/O, and Halt Device instructions may also address the device.

### 6.3.1. Input/Output Device Addressing

An I/O device and its associated channel module and control module are designated by a 13 bit I/O address. The I/O address has an 8 bit device address in bits 00-07, a channel address in bits 08-11, and an IOU number in bit 12. Because the maximum configuration allows for only eight channel modules, bit 11 of the channel address is ignored and bits 08-10 are used to select a channel module. Of the 8 bit device address, bit 07 specifies whether the selected subchannel is shared or nonshared. Device addresses with bit 7 equal to zero specify nonshared subchannels and device addresses with bit 7 equal to one specify shared subchannels. Each nonshared subchannel, regardless of channel type, is identified by a unique device address allowing a maximum of 128 nonshared subchannels per channel. On a word nonshared subchannel bit 06 of the device address specifies the ESI interface. If bit 06 equals zero, ESI interface 0 is selected and if bit 6 equals one, ESI interface 1 is selected.

For shared subchannels on a byte or block multiplexer channel, bits 04-06 of the device address select one of eight shared subchannels and its associated control unit. Bits 00-03 select one of a maximum of 16 devices. This allows a maximum of eight shared subchannels and 128 devices per byte or block multiplexer channel. There is a maximum of four shared subchannels and four

associated ISI interfaces on a word channel. Bits 05–06 of the device address select the subchannel and the ISI interface. Bit 04 must be zero and bits 00–03 are ignored. On word channels, the device address selects only a subchannel and an interface. The device is selected by an external function word.

Each channel can accommodate a different number of devices depending upon the type of channel (byte or block multiplexer or 1100 Series) and the option selected (all shared, subchannel expansion feature – option 0, or subchannel expansion feature – option 1) (See Table 6–1). Except for the rules described, the assignment of channel and device addresses is arbitrary.

### 6.3.2. States of the Input/Output System

The result of an I/O instruction is determined by the collective state of the channel, subchannel, and device selected by the I/O address. Depending on the type of channel and the I/O instruction being executed, different combinations of the states of the channel, subchannel, and device will be interrogated to determine the response to an I/O instruction. When the response to an I/O instruction is determined by the state of the channel, the subchannel and device are not interrogated. If the response to an I/O instruction is determined by the state of the subchannel, the device is not interrogated. On a word channel the device is never interrogated to determine an I/O instruction response.

The channel, subchannel, and device can each be in one of four states. (See Table 6–2.) There are 13 composite states that cover all the conditions detected by an I/O instruction. In the following paragraphs each composite state is identified by three letters. The first letter identifies the state of the channel, the second letter identifies the state of the subchannel, and the third letter identifies the state of the device. The three letters indicate the state of the channel, subchannel, and device selected by the I/O address of the I/O instruction. There are two exceptions:

1. For the LTCW instruction, the second letter indicates the state of the status table subchannel.
2. For the LCR instruction, the second letter indicates whether the channel has the feature installed to handle the LCR instruction.

Table 6-1. Device Addressing

Device Addresses (Hexadecimal)	Word Channel (1100 Series)			
	Byte Multiplexer Channel Eight Shared Subchannels	Block Multiplexer Channel 128 Nonshared Subchannels	Eight Shared Subchannels	Subchannel Expansion Feature Option 0 Subchannel Expansion Feature Option 1
00-3F <sub>16</sub>	Not used	Nonshared	Not Used	Nonshared ESI Interface A* Nonshared ESI Interface B* Not Used
40-7F <sub>16</sub>	Not Used	Nonshared	Not Used	Not Used
80-8F <sub>16</sub>	Shared 0**	Not Used	Shared 0 ISI Interface A	Not Used
90-9F <sub>16</sub>	Shared 1	Not Used	Not Used	Not Used
AO-AF <sub>16</sub>	Shared 2	Not Used	Shared 2 ISI Interface B	Not Used
BO-BF <sub>16</sub>	Shared 3	Not Used	Not Used	Not Used
CO-CF <sub>16</sub>	Shared 4	Not Used	Shared 4 ISI Interface C	Not Used
DO-DF <sub>16</sub>	Shared 5	Not Used	Not Used	Not Used
EO-EF <sub>16</sub>	Shared 6	Not Used	Shared 6 ISI Interface D	Not Used
FO-FF <sub>16</sub>	Shared 7	Not Used	Not Used	Not Used
Number of Subchannels	8 Shared 0 Nonshared	0 Shares 128 Nonshared	4 Shared 0 Nonshared	2 Shared 128 Nonshared 0 Shared 128 Nonshared

\* ESI Interface A has 64 device addresses 00-3F, and ESI Channel B has 64 device addresses 40-7F.

\*\* This number designates which of the eight channel hardware registers are associated with which device addresses. For nonshared subchannels, option 0 uses hardware registers 0, 1, 2, and 3, and option 1 uses hardware registers 0, 1, 2, 3, 4, 5, 6 and 7.

Table 6-2. Channel, Subchannel, and Device States

<u>Channel</u>	<u>Abbreviation</u>	<u>Definition</u>
Available	A	Ready to accept a nonpenetrating or penetrating instruction.
Interrupt Pending	I	Not defined.
Working	W	Operating in burst mode (block multiplexer channel only), and can accept and execute only penetrating instructions.
Not Operational	N	Not installed or offline.

<u>Subchannel</u>	<u>Abbreviation</u>	<u>Definition</u>
Available	A	Ready to accept new command.
Interrupt Pending	I	Holding status.
Working	W	Busy executing previous command.
Not Operational	N	Not installed.

<u>Device</u>	<u>Abbreviation</u>	<u>Definition</u>
Available	A	Ready to accept new command.
Interrupt Pending	I	Holding status.
Working	W	Busy executing previous command.
Not Operational	N	Not installed or not operational.

The symbol X in place of a letter indicates that the state of the corresponding component is not significant for the execution of an instruction. Unless specifically noted, the composite state applies to any type of channel.

**Channel Available (AXX):** (Byte and block multiplexer channels only.) The channel is available. The states of the subchannel and device are not significant. This condition is detected only by a Halt Channel (HCH) instruction.

**Subchannel Available (AAX):** The addressed channel and subchannel are operational, not busy executing a previous command, and not holding status. The state of the device is not significant. On a word channel a device is never interrogated to determine the response to an I/O instruction.



**Device Available (AAA):** (Byte and block multiplexer channels only) The addressed channel, subchannel, and device are operational, not busy executing a previous command, and not holding status.

**Interrupt Pending in Device (AAI) or Device Working (AAW):** (Byte and block multiplexer channels only) The addressed channel and subchannel are available. The addressed control unit or I/O device is executing a previously initiated operation or is holding status. The following situations are possible:

1. The control unit is executing an operation on the addressed device or on another device associated with the same control unit.
2. The device or control unit is executing an operation on another channel or subchannel.
3. The device or control unit is holding status for the addressed device or another device associated with the same control unit.

**Device Not Operational (AAN):** (Byte and block multiplexer channels only) The addressed channel and subchannel are available. The addressed I/O device is not operational. This occurs when the control unit for the addressed device is not installed or not on line.

**Interrupt Pending in Subchannel (AIX):** The addressed channel is available. The addressed subchannel is holding status from either a previously initiated operation or the present instruction attempting to be initiated. The subchannel is ready to store its status in a channel status word (CSW). The status can be for the addressed device or another device on the subchannel. The state of the addressed device is not significant unless a Test I/O instruction is issued to a byte or block multiplexer channel, and the addressed subchannel is holding status for the addressed device. In this case a Test I/O command is issued by the channel, and the channel status word always contains device status.

**Subchannel Working (AWX):** The addressed channel is available. The addressed subchannel is busy executing a previously executed operation. The state of the device is not significant.

**Subchannel Not Operational (ANX):** The addressed channel is available. The addressed subchannel is not operational. This occurs when the channel is not equipped to handle that subchannel because of the particular type of channel and features selected.

**Interrupt Pending in Channel (IXX):** This condition is never detected because channel status is reported by an independent interrupt mechanism. Channel status is not detectable or retrievable by way of I/O instruction. (See Machine Check interrupts.)

**Channel Working (WXX):** (Block multiplexer channel only) The addressed channel is operating in burst mode (transferring a block of data). The states of the subchannel and device are not significant. The TIO, TSC, and LCR instructions do not penetrate a channel in a working state and are not executed. A Halt Device (HDV) instruction penetrates a working channel only if the channel is working with the addressed device. The HCH instruction always penetrates a working channel and halts the device that has control of the channel interface at the time that the HCH instruction is received. The Start I/O Fast Release (SIOF) instruction always penetrates a working channel. The response to the SIOF instruction is determined by the state of the subchannel.

**Channel Not Operational (NXX):** An addressed channel is not operational when it is not installed in the system or is not on line. The states of the addressed subchannel and device are not significant.

**Hardware Fault (XXX):** If the IOU detects a hardware fault before the channel is selected, the instruction is terminated and a machine check interrupt is generated. The state of the I/O system is insignificant.

**Software Fault (XXX):** (Byte and Block multiplexer channel only) If an SRL instruction is issued to a block multiplexer channel that had not previously presented unit check device status, the instruction is not executed regardless of the state of the I/O system.

### 6.3.3. Condition Codes

The result of an I/O instruction is reported by a two bit condition code. The condition code is stored by the processor in bits 34-35 of the Xa register at the time the execution of the instruction is completed. The condition code is determined by the composite state of the I/O system selected by the I/O instruction. Tables 6-3, 6-4, and 6-5 show the relationship between the condition code for each instruction and the composite state of the I/O system. Special conditions affecting the condition code are also indicated.

### 6.3.4. Instruction Format and Channel Address Word

All I/O instructions have an  $f = 75_8$ . The  $j$  value specifies the particular I/O instruction to be initiated:

$j = 00_8 =$ Sense Release	(SRL)
$j = 01_8 =$ Start I/O Fast Release	(SIOF)
$j = 02_8 =$ Test I/O	(TIO)
$j = 03_8 =$ Test Subchannel	(TSC)
$j = 04_8 =$ Halt Device	(HDV)
$j = 05_8 =$ Halt Channel	(HCH)
$j = 10_8 =$ Load Channel Register	(LCR)
$j = 11_8 =$ Load Table Control Words	(LTCW)

Table 6-3. I/O System Composite State vs Condition Codes

Composite State	Condition Code	Byte Mux Channel	Block Mux Channel	Word Channel
AAA*	0, 1	HDV, TIO	HDV, SRL, TIO, SRL	(1)
AAI	1	HDV, TIO	HDV, TIO	(1)
AAW	1	HDV, TIO	HDV, TIO	(1)
AAN	3	HDV, TIO	HDV, TIO	(1)
AAX	0, 0, 1	LCR, LTCW, TSC	LCR, TSC, SIOF	HDV, LCR, LTCW, TIO, TSC, SIOF
AIX	1, 2 (a)	HDV, LTCW, SIOF, TIO, TSC	HDV, SIOF, TIO, TSC	HDV, LTCW, SIOF, TIO, TSC
AWX	0, 1, 2	HDV, LTCW, HDV, SIOF, TIO, TSC	HDV, HDV, SIOF, TIO, TSC	HDV, LTCW, SIOF, TIO, TSC
ANX	3	HDV, LCR, LTCW, SIOF, TIO, TSC	DHV, LCR, SIOF, TIO, TSC	HDV, LCR, LTCW, SIOF, TIO, TSC
AXX	0	HCH	HCH	(2)
IXX	-	(2)	(2)	(2)
WXX	0, 2	(3)	HCH, HCV, TIO, TSC, LCR	(4)
WAX	0	(3)	SIOF	(3)
WIX	1, 2	(3)	SIOF, SIOF, HEV, TIO, TSC	(3)
WWX	0, 2	(3)	HDV, HDV, SIOF, TIO, TSC	(3)
WNX	3	(3)	HDV, LCR, SIOF, TIO, TSC	(3)
NXX	3	HCH, HDV, LCR, LTCW, SIOF, TIO, TSC	HCH, HDV, LCR, LTCW, SIOF, TIO, TSC	HCH, HDV, LCR, LTCW, SRL, SIOF, TIO, TSC
XXX	2, 3	HCH, HDV, LCR, LTCW, SRL, SIOF, TIO, TSC, SRL	HCH, HDV, LCR, LTCW, SRL, SIOF, TIO, TSC, SRL	HCH, HDV, LCR, LTCW, SRL, SIOF, TIO, TSC

A = Available      I = Interrupt Pending      W = Working      N = Not Available

X = Any of the above states

\* The three letters, from left to right, indicate the state of the channel, subchannel, and device selected by the I/O address with two exceptions: a) For the LCR instruction, the second letter indicates whether the channel has the feature installed to handle the LCR instruction, b) For the LTCW instruction, the second letter indicates the state of the status table subchannel.

- a. Special conditions in the channel determine whether a condition code of 1 or 2 will be presented. These special conditions are covered in Table 6-4.
  1. A word channel never interrogates a device to determine a response to an I/O instruction.
  2. This condition is not detectable by any I/O instruction.
  3. Byte channels and word channels do not transfer blocks of data, only bytes or words, and thus can never be in the working state.
  4. The word channel is never in the working state.

Table 6-4. I/O Instruction Condition Codes for Byte or Block Channel

State #	Channel	Conditions	SRL (a)	SIOF (b)	TIO	TSC	HDV	HCH	LCR	LTCW
AAA	Block	-	0/1 (c)	-	0	-	0	-	-	-
AAI	Byte, Block	-	-	-	1	-	1	-	-	-
AAW	Byte, Block	-	-	-	1	-	1	-	-	-
AAN	Byte, Block	-	-	-	3	-	3	-	-	-
AAX	Byte, Block	-	0/1(e)	-	-	0	0	-	0	0
AIX	Byte, Block	(1) and (3)	1	1	1	1	1	-	0	1
AIX	Byte, Block	(1) and (4)	2	2	2	2	2	-	0	2
AIX	Byte, Block	(2)	2	2	2	1	2	-	-	-
AWX	Byte, Block	-	2	2	2	2	1	-	-	0
ANX	Byte, Block	-	3	3	3	3	3	-	3	3
AXX	Byte, Block	-	-	-	-	-	-	0	-	-
IXX	Byte, Block	-	-	-	-	-	-	0	10	-
WXX	Block	-	-	-	-	-	-	0	-	-
WAX	Block	-	0/1 (e)	-	2	2	2	0	-	-
WIX	Block	(1) and (3)	-	1	-	-	2	-	2	-
WIX	Block	(2) or (4)	-	2	-	-	2	-	-	-
WWX	Block	-	-	-	-	-	2	-	-	-
WNX	Block	-	-	-	-	-	1/2 (d)	-	-	-
NXX	Byte, Block	-	-	3	3	3	3	-	3	3
XXX	Byte, Block	(5)	2	3	2	2	3	3	3	3
XXX	Block	(6)	3	2	2	2	2	2	2	2
XXX	Byte, Block	(7)	-	-	-	-	-	-	-	-

States: A = Available I = Interrupt Pending W = Working N = Not Available X = Any of above states

Footnotes for table are as follows:

- \* The three letters, from left to right, indicate the state of the channel, subchannel, and device selected by the I/O address with two exceptions: a) for the LCR instruction, the second letter indicates whether the channel has the feature installed to handle the LCR instruction, b) for the LTCW instruction, the second letter indicates the state of the status table subchannel.

**Footnotes for Table 6-4.**

- a. The Sense Release instruction will respond with a condition code of 0 or 1 only if Unit Check device status has been presented via interrupt or previous instruction. Only one Sense Release instruction per Unit Check device status byte will be accepted. In all other cases a condition code of 3 will be returned.
- b. If the SIOF queue is full, or the channel is in contingent connection mode, the SIOF instruction will unconditionally receive a condition code of 2.
- c. For an immediate command, the condition code may equal 1. For any other command, the condition code equals 0.
- d. If the channel is working (operating in burst mode) with the addressed device, the operation is terminated and the condition code equals 1. If the channel is working, but not with the addressed device, the condition code equals 2.
- e. If a hardware or software error is detected when retrieving the second word of the CAW, a CSW is stored and the instruction receives a condition code of 1. If no hardware or software error is detected, the instruction receives a condition code of 0.
  - (1) Subchannel is holding status for addressed device.
  - (2) Subchannel is holding status for a device other than the addressed device.
  - (3) Interrupt cancellation was not attempted or was attempted and completed successfully.
  - (4) Interrupt cancellation was attempted and was unsuccessful.
  - (5) Hardware fault was detected when reading first word of the CAW and Machine Check interrupt was generated.
  - (6) Unit Check status had not been presented via interrupt or instruction.
  - (7) The channel is in contingent connection mode.

Table 6-5. I/O Instruction Condition Codes for Word Channels

State#	Channel	Conditions	SRL	SIOF	TIO	TSC	HDV	HCH	LCR	LTCW
AAx	Word	-	-	0	0	0	0	-	0	0
AIX	Word	(1)	-	1	1	1	1	-	-	1
AIX	Word	(2)	-	2	2	2	2	-	-	2
AWX	Word	-	-	2	2	2	0	-	9	0
ANX	Word	-	-	3	3	3	3	-	3	3
AXX	Word	-	-	-	-	-	-	-	-	-
IXX	Word	-	-	-	-	-	-	-	-	-
WXX	Word	-	-	-	-	-	-	-	-	-
NXX	Word	-	-	3	3	3	3	-	3	3
XXX	Word	(3)	2	2	2	2	2	2	2	2
XXX	Word	(4)	3	-	-	-	-	3	-	-

States

A = Available

I = Interrupt Pending

W = Working

N = Not Available

X = Any of the above statements

\* The three letters, from left to right, indicate the state of the channel, subchannel, and device respectively.

(1) Interrupt cancellation was not attempted or was attempted and completed successfully.

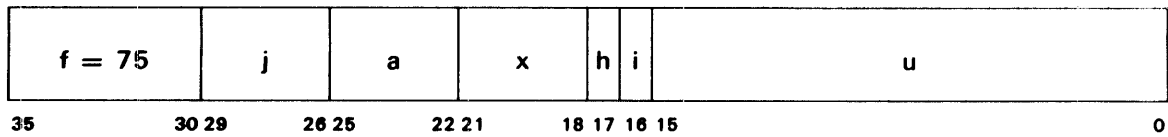
(2) Interrupt cancellation was attempted unsuccessfully.

(3) Hardware fault was detected when reading first word of the CAW and Machine Check interrupt was generated.

(4) The Sense Release and Halt Channel instructions are not accepted on a word channel.

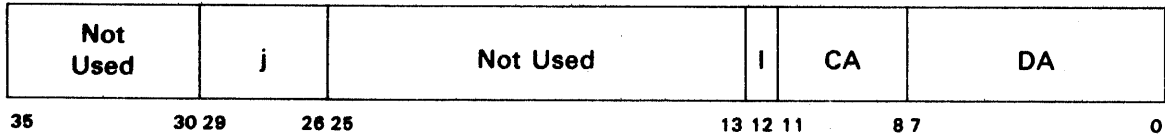
Bits 00-12 of  $u + X_m$  specify the I/O address (the IOU, channel, and device numbers). Bits 00-23 of  $X_a$  consist of either the starting address of the CCW or STCW list or the register input data for the LCR instruction. Upon detection of an I/O instruction, the processor builds a channel address word (CAW) in a fixed location of low-order storage. The CAW is for hardware use only and consists of two 36-bit words, CAW 0 and CAW 1. The  $j$ -value is stored in CAW 0 bits 26-29. Bits 00-12 of  $u + X_m$  (the I/O address) are stored in CAW 0 bits 00-12. Bits 00-23 of  $X_a$  (the first CCW or STCW address or register input data) are stored in CAW 1 bits 00-23. The IOU refers to the CAW only during the execution of an I/O instruction. The pertinent information thereafter is stored in the channel.

I/O Instruction Format



- $j$  Specifies I/O instruction
- $a$  Address of register holding first CCW address
- $x$   $u + X_m$  bits 0-12 equal I/O address

CAW



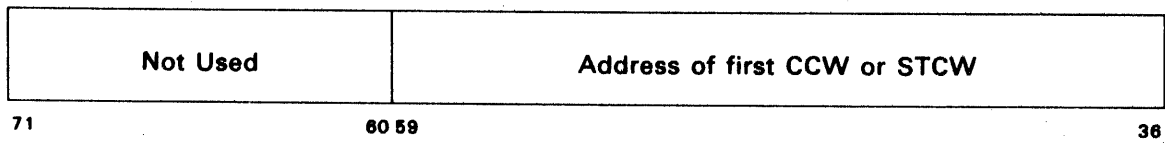
j I/O instruction

I IOU number

CA Channel Address

DA Device Address

CAW 1





### 6.3.5. Instruction Operation

Upon detection of an I/O instruction, the processor builds the CAW and then initiates the IOU via the processor/IOU interface. The IOU establishes instruction priority between the two processors (Processor 0 has priority over Processor 1) and reads CAW 0. If the IOU detects a hardware fault in reading CAW 0, the instruction is immediately terminated, a Machine Check interrupt is generated, and a condition code of 2 is presented to the processor.

If no hardware fault is detected, the IOU then executes the instruction. Depending on the I/O instruction, the IOU uses the state of the channel, the states of the channel and subchannel, or the states of the channel, subchannel, and device to determine the condition code. When the IOU has completed the I/O instruction, the processor clears bits 30–33 of Xa and stores the condition code in bits 34–35 of Xa. Bits 00–29 of Xa are left unchanged. If the condition code equals 0, the processor skips the next instruction in the program. If the condition code equals 1, 2, or 3, the processor executes the next instruction.

## 6.4. I/O INSTRUCTIONS

**Programming Note:** An I/O instruction may cause a Channel Status Word (CSW) to be stored. To prevent the contents of the CSW stored by the instruction from being destroyed by an immediately following I/O interrupt, interrupts must be locked out before issuing the I/O instruction and must remain locked out until the information in the CSW provided by the instruction has been acted upon or stored elsewhere for later use.

Use Tables 6–4 and 6–5 to determine what the condition code response to an I/O instruction means. The I/O instructions can be classified as penetrating or nonpenetrating. A penetrating I/O instruction is always executed even if the channel is in a working state. (Note that only a block multiplexer channel can be in a working state.) A nonpenetrating I/O instruction always receives a busy response (condition code = 2) when attempted on a channel in a working state. The Start I/O Fast Release and Halt Channel instructions are penetrating instructions and are always executed even on working channels. A Halt Device instruction is executed only if: a) the channel is working with the addressed device, or b) the channel is in the available state.

For any I/O instruction a condition code of zero indicates that the instruction was completed successfully. A condition code of one always indicates that a valid CSW has been written into low-order storage. A subchannel is always returned to the available state after being relieved of status by an instruction or an interrupt. A condition code of two indicates that the I/O instruction was not executed. A condition code of 3 indicates that the instruction was not executed because either the channel, subchannel, or device was not operational.

After each instruction, the condition codes and the conditions causing each condition code are listed.

### 6.4.1. Sense Release – SRL 75,00

A Sense Release instruction initiates the execution of a CCW list on the IOU, channel, subchannel, and device specified by the I/O address. A Sense Release instruction is accepted only on a block or byte multiplexer channel and only if the addressed channel has previously presented Unit Check device status via interrupt or previous instruction. In all other cases the instruction is not accepted and a condition code of 3 is returned.

When the block multiplexer channel receives Unit Check status from a device, all further channel operations are temporarily halted. The handling of any further device requests and the execution of queued SIOFs are halted by the block multiplexer channel until an SRL instruction is received. However, all I/O instructions are still executed.

When the byte multiplexer channel receives Unit Check device status, the execution of queued SIOFs is halted by the channel until an SRL instruction is executed by the channel. The byte multiplexer channel continues to handle all device requests for data transfers and status transfers except for another Unit Check device status indication. Device status containing Unit Check status is stacked in the control unit until an SRL instruction that relieves the first Unit Check status indication is executed by the channel.

The channel is halted when Unit Check device status is received to eliminate the possibility of destruction of sense data in the control units that save sense data per control unit rather than per device. The execution of an SRL instruction unconditionally unlocks the interface and the SIOF stack.

#### **6.4.1.1. Byte or Block Multiplexer Channel Operation**

##### **Condition Code = 0**

1. The channel had previously presented Unit Check status via interrupt or previous instruction. The first CCW address and first CCW were read, and the operation specified by the CCW was initiated successfully. The subchannel and device are now in a working state, and channel operations are no longer halted. Device requests are being handled, and queued SIOFs are being executed.

##### **Condition Code = 1**

1. The channel had previously presented Unit Check status via interrupt or previous instruction. The first CCW address and first CCW were read, and the operation specified by the CCW was initiated successfully. An immediate command was specified in the first CCW with either no command chaining specified or chaining suppressed because of unusual conditions detected during the operation (check the device and subchannel status fields of the CSW).
2. The channel had previously presented Unit Check status via interrupt or previous instruction. A software or hardware error was detected when attempting to execute the instruction (check the device and subchannel status fields of the CSW).

##### **Condition Code = 2**

1. A hardware fault was detected when fetching the first word of the CAW.
2. The channel or subchannel was busy.

##### **Condition Code = 3**

1. The channel, subchannel, or device was not operational.
2. Unit Check device status had not been presented via interrupt or previous instruction. The instruction was rejected and not executed.

#### **6.4.1.2. Word Channel Operation**

##### **Condition Code = 0**

Impossible

**Condition Code = 1**

Impossible

**Condition Code = 2**

1. A hardware fault was detected when reading the first word of the CAW.

**Condition Code = 3**

1. A Sense Release instruction was issued to a word channel.

**6.4.2. Start I/O Fast Release – SIOF 75,01**

If the addressed subchannel of an SIOF instruction is available, the second word of the CAW that contains the address of the first CCW is retrieved and stored in the subchannel. If the channel is not in contingent connection mode and no hardware or software errors are detected, the device address is placed in an SIOF queue, and a condition code of zero is presented to the processor. If a software or hardware error is detected during retrieval of the address of the first CCW, subchannel status is generated, a CSW is stored, and a condition code of 1 is presented to the processor.

Whenever the channel becomes available, the device addresses of successfully executed SIOFs (condition code of zero) are fetched from the queue on a first in/first out basis. When a device address is fetched from the queue, the first CCW specified for that device address is retrieved from storage and the operation specified by that CCW is initiated at the device if the device is available. If the device is not available, the operation is not initiated, and the software is notified via interrupt.

The subchannel is set to an active state at the time that the device address associated with that subchannel is placed in the SIOF queue. The subchannel remains in an active state until either termination status is detected or the operation is terminated by an I/O instruction. Only one SIOF instruction per subchannel and 64 SIOF instructions per channel can be held in the queue at one time.

**6.4.2.1. Byte or Block Multiplexer Channel Operation****Condition Code = 0**

1. The channel was operational and the subchannel was available. The first CCW address was read from the CAW and stored in the subchannel successfully, and the device address was loaded in the SIOF queue.

**Condition Code = 1**

1. The subchannel was holding status from a previous operation on the addressed device (check the device and subchannel status fields of the CSW).
2. A software or hardware error was detected when attempting to fetch the address of the first CCW from the CAW (check the subchannel status field of the CSW).

**Condition Code = 2**

1. The subchannel was holding status for the addressed device, but was busy presenting the status by way of interrupt.
2. The subchannel was busy holding status for or executing a previously initiated operation on a device other than the addressed device.
3. The subchannel was busy executing a previously initiated operation with the addressed device.
4. The SIOF queue was full. The channel had 64 pending SIOFs to be initiated at the device level.
5. A hardware fault was detected when fetching the first word of the CAW.

Condition Code = 3

1. The addressed channel or subchannel was not operational.

#### 6.4.2.2. Word Channel Operation

Condition Code = 0

1. The channel was operational, and the subchannel was available. The first CCW address was read from the CAW and stored in the subchannel successfully, and the subchannel address was loaded in the SIOF queue.

Condition Code = 1

1. The subchannel was holding status from a previous operation (check the device and subchannel status fields of the CSW).
2. A software or hardware error was detected when attempting to fetch the address of the first CCW from the CAW (check the subchannel status field of the CSW).

Condition Code = 2

1. The subchannel was holding status, but was busy presenting the status by way of interrupt.
2. The subchannel was busy executing a previously initiated operation.
3. The SIOF queue was full. The channel had 64 pending SIOFs to be initiated at the device level.
4. A hardware fault was detected when fetching the first word of the CAW.

Condition Code = 3

1. The channel or subchannel was not operational.

#### 6.4.3. Test I/O - TIO 75,02

The TIO instruction interrogates the channel, subchannel, and device specified by the I/O address for status conditions. The subchannel and device are not interrogated if the channel is in the working state. The device is not interrogated if the subchannel is in the working state. The TIO instruction interrogates the device only on a byte or block multiplexer channel. On a word channel the TIO instruction is performed as a Test Subchannel instruction.

### 6.4.3.1. Byte or Block Multiplexer Channel Operation

#### Condition Code = 0

1. The channel, subchannel, and device are available.

#### Condition Code = 1

1. The control unit was busy executing or holding status for a previously initiated operation on either the addressed device or another device (check the device status field of the CSW).
2. The subchannel was holding status from a previous operation on the addressed device (check the device and subchannel status fields of the CSW).

#### Condition Code = 2

1. The subchannel was holding status for the addressed device, but was busy presenting the status by way of interrupt.
2. The subchannel was busy holding status for or executing a previously initiated operation on a device other than the addressed device.
3. The subchannel was busy executing a previously initiated operation on the addressed device.
4. On a block multiplexer channel, the channel was busy operating in burst mode.
5. A hardware fault was detected when fetching the first word of the CAW.

#### Condition Code = 3

1. The channel, subchannel, or device was not operational.

### 6.4.3.2. Word Channel Operation

See the Test Subchannel instruction. On a word channel a Test I/O instruction is executed as a Test Subchannel instruction.

### 6.4.4. Test Subchannel – TSC 75,03

The Test Subchannel instruction interrogates the channel and subchannel specified by the I/O address. The addressed device is not affected. The subchannel is not interrogated if the channel is in a working state.

#### 6.4.4.1. Byte or Block Multiplexer Channel

##### Condition Code = 0

1. The channel and subchannel are available.

**Condition Code = 1**

1. The subchannel was holding status from a previously initiated operation (Check the device and subchannel status fields of the CSW).

**Condition Code = 2**

1. The subchannel was holding status, but was busy presenting the status by way of interrupt.
2. The subchannel was busy executing a previously initiated operation.
3. On a block multiplexer channel only, the channel was busy operating in burst mode.
4. A hardware fault was detected in fetching the first word of the CAW.

**Condition Code = 3**

1. The channel or subchannel was not operational.

**6.4.4.2. Word Channel Operation****Condition Code = 0**

1. The channel and subchannel are available.

**Condition Code = 1**

1. The subchannel was holding status from a previously initiated operation (Check the device and subchannel status fields of the CSW).

**Condition Code = 2**

1. The subchannel was holding status, but was busy presenting the status by way of interrupt.
2. The subchannel was busy executing a previously initiated operation.
3. On a block multiplexer channel only, the channel was busy operating in burst mode.
4. A hardware fault was detected in fetching the first word of the CAW.

**Condition Code = 3**

1. The channel or subchannel was not operational.

**6.4.5. Halt Device – HDV 75,04**

The Halt Device instruction terminates the current operation on the channel and subchannel specified by the I/O address. The operation on the specified subchannel is terminated immediately and the device is notified of the termination when the device references the channel.

**Programming Note:** The Halt Device instruction is intended for use only as a recovery mechanism for software or hardware faults because the capability of an HDV instruction is limited for two reasons:

1. A Halt Device instruction does not penetrate a working block multiplexer channel and is rejected unless the channel is working with the addressed device.
2. The resultant state of the device on a byte or block multiplexer channel after receiving a Halt Device instruction is unpredictable. The device may or may not still be active and may or may not eventually present status via interrupt or instruction.

#### 6.4.5.1. Byte or Block Multiplexer Channel Operation

##### Condition Code = 0

1. The channel, subchannel, and device were in the available state.
2. The channel was available. The operation that was being executed by the subchannel with either the addressed device or another device on that subchannel has been terminated. The device has been signaled to terminate the operation. At a later time the device, after having completed termination of the operation, may present status by way of interrupt or instruction. The subchannel has been returned to the available state.

##### Condition Code = 1

1. The channel and subchannel were available. The control unit was busy executing a previously initiated operation on either the addressed device or another device (check the device status field of the CSW).
2. The subchannel was holding status from a previous operation on the addressed device (check the device and subchannel status fields of the CSW).
3. The channel and subchannel were busy operating in burst mode with the addressed device. The operation has been terminated and the device has been signaled to terminate the operation. At a later time the device, after having completed termination of the operation, may present status by way of interrupt or instruction. The subchannel has been returned to the available state.

##### Condition Code = 2

1. The subchannel was holding status for the addressed device, but was busy presenting the status by way of interrupt.
2. The subchannel was holding status for a device other than the addressed device.
3. On a block multiplexer channel, the channel was operating in burst mode with a device other than the addressed device.
4. A hardware fault was detected when fetching the first word of the CAW.

##### Condition Code = 3

1. The channel, subchannel, or device was not operational.

#### 6.4.5.2. Word Channel Operation

##### Condition Code = 0

1. The addressed channel was operational and the addressed subchannel was in the available state.
2. The addressed channel was operational. The operation that was being executed by the addressed subchannel has been terminated. The subchannel has been returned to the available state.

##### Condition Code = 1

1. The subchannel was holding status from a previously initiated operation (check the device and subchannel status fields of the CSW).

##### Condition Code = 2

1. The subchannel was holding status, but was busy presenting the status by way of interrupt.
2. A hardware fault was detected when fetching the first word of the CAW.

##### Condition Code = 3

1. The channel or subchannel was not operational.

#### 6.4.6. Halt Channel – HCH 75,05

The Halt Channel instruction terminates the current operation on the channel specified by the I/O address. The Halt Channel instruction is intended to be used to recover a byte or block multiplexer channel that has become inoperative during peripheral interface sequencing because of a control unit interface error. The Halt Channel instruction is not accepted on a word channel.

##### 6.4.6.1. Byte or Block Multiplexer Channel Operation

##### Condition Code = 0

1. The channel was available.
2. The channel was busy operating in burst mode or was in a hung condition because an interface error has resulted in the suspension of normal sequencing. The operation on the subchannel that had control of the channel interface has been terminated, and the device has been signaled to terminate the operation. At a later time when the device has completed terminating the operation, it may present status by way of interrupt. The subchannel has been returned to the available state.

##### Condition Code = 1

Impossible



Condition Code = 2

1. A hardware fault was detected when fetching the first word of the CAW.

Condition Code = 3

1. The channel was not operational.

**6.4.6.2. Word Channel Operation**

Condition Code = 0

Impossible

Condition Code = 1

Impossible

Condition Code = 2

1. A hardware fault was detected when fetching the first word of the CAW.

Condition Code = 3

1. The channel was a word channel.

**6.4.7. Load Channel Register – LCR 75,10**

The Load Channel Register instruction is used to load the interrupt mask register in the IOU control module or the channel base register in the channel specified by the I/O address. The operation to be performed is specified by bit 00 of the I/O address field. If bit 00 of the CAW is a zero, the channel base register of the addressed channel is loaded with the contents of bits 36 through 50 of the CAW. If bit 00 of the CAW is a one, the interrupt mask register is loaded with the contents of bits 36–71 of the CAW. The use of the channel base register is described in 6.21 and the use of the interrupt mask register is described in 6.22.

**6.4.7.1. Byte and Block Multiplexer Channel**

Condition Code = 0

1. The interrupt mask register or channel base register was loaded successfully.

Condition Code = 1

Impossible

Condition Code = 2

1. A hardware fault was detected when fetching the first word of the CAW.
2. The channel was operating in burst mode (loading the channel base register on a block multiplexer channel only).

**Condition Code = 3**

1. The channel was not operational or did not have the subchannel expansion feature installed (loading the channel base register only).

**6.4.7.2. Word Channel Operation****Condition Code = 0**

1. The interrupt mask register or channel base register was loaded successfully.

**Condition Code = 1**

Impossible

**Condition Code = 2**

1. A hardware fault was detected when fetching the first word of the CAW.
2. The channel was operating in burst mode (loading the channel base register on a block multiplexer channel only).

**Condition Code = 3**

1. The channel was not operational or did not have the subchannel expansion feature installed (loading the channel base register only).

**6.4.8. Load Table Control Words – LTCW 75,11**

The Load Table Control Words instruction loads the status table subchannel in the IOU and channel specified by the I/O address. The status table subchannel controls the tabling of communication interrupt status as described in Section 16. The LTCW instruction initiates the execution of a CCW list by the status table subchannel on the IOU and channel selected by the I/O address. Bits 36–59 of the CAW contain the address of the first Status Table Control Word (STCW) of the STCW list. A STCW contains the data count and the starting address for the status table. Even if the status table subchannel is active (has a valid data count), a new LTCW instruction is accepted, a new STCW list is initiated, and the status table subchannel is loaded with the first STCW of the new list.

**6.4.8.1. Byte and Block Multiplexer Channel****Condition Code = 0**

1. The addressed channel was operational and the status table subchannel was available or active. The first STCW has been fetched and the status table subchannel has been initiated successfully.

**Condition Code = 1**

1. The status table subchannel was holding status because of a software or hardware error on a previous operation (check the subchannel status field of the CSW).
2. A software or hardware error was detected when attempting to fetch either the address of the first STCW from the CAW or the first STCW from storage (check the subchannel status field of the CSW).

**Condition Code = 2**

1. The status table subchannel was holding status from a previous operation, but was busy presenting the status by way of interrupt.
2. A hardware fault was detected when fetching the first word of the CAW.

**Condition Code = 3**

1. The channel was not operational or did not have the feature installed allowing it to handle communication interrupt status or was a block multiplexer channel.

**6.4.8.2. Word Channel Operation****Condition Code = 0**

1. The addressed channel was operational and the status table subchannel was available or active. The first STCW has been fetched and the status table subchannel has been initiated successfully.

**Condition Code = 1**

1. The status table subchannel was holding status because of a software or hardware error on a previous operation (check the subchannel status field of the CSW).
2. A software or hardware error was detected when attempting to fetch either the address of the first STCW from the CAW or the first STCW from storage (check the subchannel status field of the CSW).

**Condition Code = 2**

1. The status table subchannel was holding status from a previous operation, but was busy presenting the status by way of interrupt.
2. A hardware fault was detected when fetching the first word of the CAW.

**Condition Code = 3**

1. The channel was not operational or did not have the feature installed allowing it to handle communication interrupt status or was a block multiplexer channel.

**6.5. EXECUTION OF I/O OPERATIONS**

A channel can execute seven commands: write, read, read backward (only on a byte or block multiplexer channel), sense (only on a byte or block multiplexer channel), control, transfer in channel (TIC), and store subchannel status. Each command except transfer in channel and store subchannel status initiates a corresponding I/O operation. The initiation and execution of a command issued to a subchannel and device is termed an "I/O operation".

A series of I/O operations on the same device (byte or block multiplexer channels) or on the same subchannel (word channel) is executed under control of a set of channel command words (CCW's). The execution of a set of channel command words (CCW list) is initiated by a Start I/O Fast Release instruction. For a Start I/O Fast Release instruction, the address of the first CCW is stored in the channel, and a condition code is presented to the processor. At an idle time in the channel

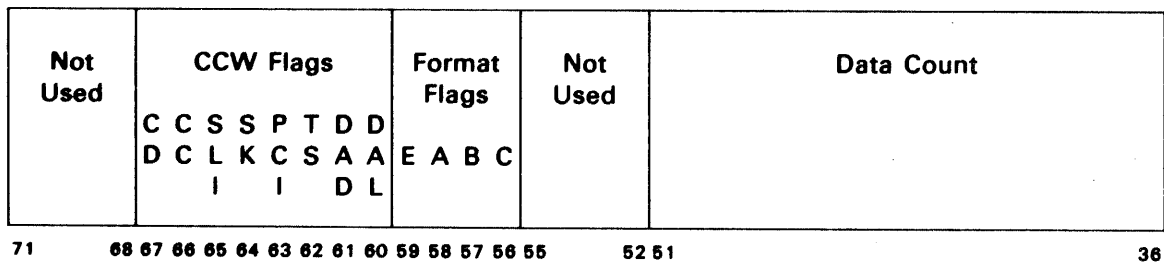
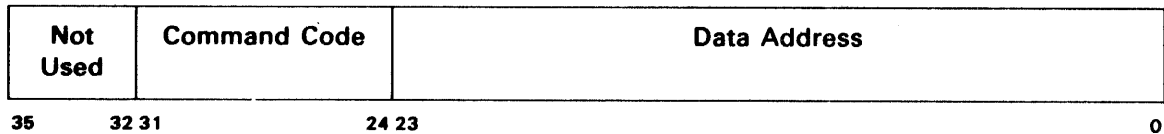
sequencing, the first CCW is fetched and the specified I/O operation is initiated. The CCWs can be located anywhere in main storage. Fetching of a CCW by the channel does not affect the contents of the location in main storage. Each additional CCW in the CCW list is obtained when the operation has progressed to the point where the additional CCW is needed.

### 6.5.1. Channel Command Word

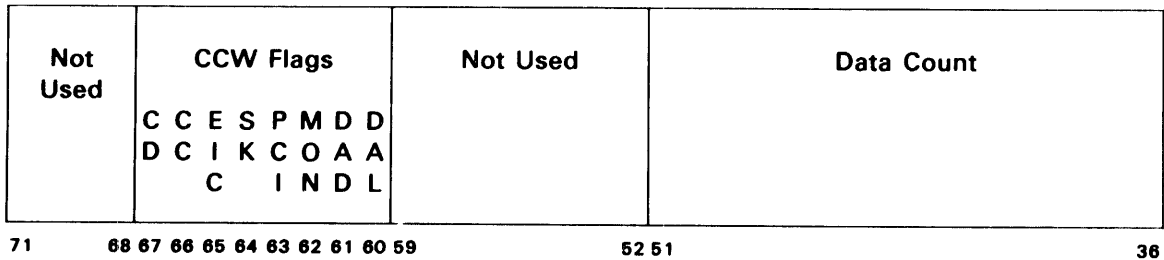
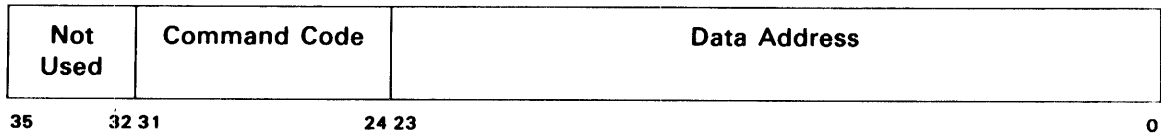
A channel command word specifies the command to be executed, and for commands initiating I/O operations it designates the storage area associated with the operation, the amount of data to be transferred, the formatting of data that is to be done, and the action to be taken when the operation is completed.

The CCW has the following format:

#### Byte or Block Multiplexer Channel CCW



Word Channel CCW



The fields in the CCW are allocated for the following purposes:

**Data Address:** Bits 00–23 contain the storage address of the first data word to be transferred unless the command code is Transfer in Channel or Store Subchannel Status. For the Transfer in Channel command, the field contains the new CCW address, and for the Store Subchannel Status command the field contains the address where status is to be stored.

**Command Code:** Bits 24–31 specify the operation to be performed by the subchannel and device.

**Data Count:** Bits 36–51 specify the number of bytes to be transferred on a byte or block multiplexer channel, or the number of 30- or 36-bit words transferred on a word channel ISI interface, or the number of quarter words or half words transferred on a word channel ESI interface.

**Special Flags:** Bits 52–67 contain flags that specify data formats, special handling of an operation by the channel, and the action to be taken once the present operation is completed.

**Chain Data (CD):** Bit 67 specifies that upon completion of the portion of a data transfer operation being controlled by the current CCW, a new CCW is to be read from storage and the operation is to be continued under control of the new CCW.

**Chain Command (CC):** Bit 66 specifies that upon completion of the operation, a new CCW is to be read from storage and the operation specified by the new command code is to be initiated. If the chain data flag is set, the Chain Command flag is ignored.

**Suppress Length Indication (SLI):** (Byte and block multiplexer channels only. Bit 65) – Specifies that if command chaining conditions are present, a command chain operation be initiated regardless of the residual byte count. The absence of the Suppress Length Indication bit specifies that if command chaining conditions are present, a command chain operation be initiated only if the residual byte count equals zero. If the command chaining conditions are present, and the Suppress Length Indication bit is not set, and the residual byte count does not equal zero, the execution of the CCW list is terminated, subchannel status is generated, and an interrupt is presented to the processor. The suppress length indication flag is ignored if the truncated search flag is set in the same CCW.

**NOTE:**

*Command chaining conditions are defined as Channel End and Device End status, the command chain flag set, and the data chain flag clear in the active CCW.*

**EI Chain (EIC):** (Word channel only and ESI subchannels only) – Bit 65 specifies that upon completion of the operation at the ESI word channel device, the operation specified by the new command is to be initiated. The external interrupt presented by the device is stored in the status table prior to chaining, and a tabled interrupt request is presented to the processor.

The external interrupt chain is not executed if:

- a. The status table subchannel is not active,
- b. A hardware error is detected when entering the external interrupt in the status table,
- c. A hardware or software error is detected during the retrieval of the new CCW from storage, or
- d. The subchannel was not in an active state before receiving the external interrupt.

The EI chain flag is not interpreted on ISI word channels.

**Skip Data (SK):** Bit 64 specifies that data will not be written in storage for input operations. However, the device and subchannel are handled in the same manner as during conventional input operation. For all other operations, the skip data flag is ignored.

**Program Controlled Interrupt (PCI):** Bit 63 specifies that the channel shall store a PCI subchannel status indication and generate an interrupt as soon as possible after a CCW containing this flag is obtained.

**Truncated Search (TS):** (Block multiplexer channel only) – Specifies that a special chaining operation is to be executed. The channel saves the command in the CCW with the truncated search flag and also saves the command from the preceding CCW. These two commands are then reissued to the control unit during a truncated search operation. See 6.8.4. for a detailed description of truncated search operations.

**Monitor (MON):** (Word channel only Bit 62) – Specifies that the channel shall store subchannel status and generate an interrupt when the data count in the final CCW has been exhausted.

**Data Address Decrement (DAD):** Bit 61 specifies that the data address be decreased by one for each full data word transferred under control of the current CCW. This flag is ignored if the data address lock (DAL) flag is set. If neither the DAD or DAL flags are set, the data address will be incremented by one for each full data word transferred.

**Data Address Lock (DAL):** Bit 60 specifies that the contents of the data address field remain unchanged for each word transferred under control of the current CCW.

**Emulation Mode (E, bit 59):** – On byte and block multiplexer channels only E equals zero specifies 1100 Series mode for data transfer operations and the 36-bit data packing format is selected. E equals one if invalid.

On word channels the E bit is ignored and the mode of operation is specified by patch wire. Each word channel 1100 Series ESI interface operates via patch wire in either quarter-word mode or half-word mode.

Format Flags (A, bit 58; B, bit 57; and C, bit 56): (Byte and block multiplexer channels only) – Specify the packing format of data bytes. (See Tables 6-6, 6-7, and 6-8.) Format A or format B must be selected on the byte multiplexer channel. The format flags are ignored on a word channel.

The contents of bit positions 32-35, 52-55, and 68-71 of the CCW are ignored.

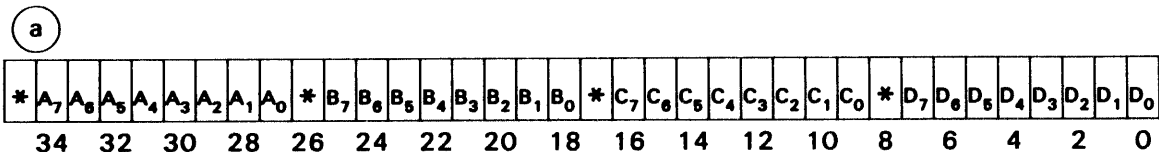
### 6.5.2. CCW Completion

A CCW operation can be terminated by the channel or by the device. A CCW operation may also be terminated by the HDV or HCH instructions. Termination by the HCH and HDV instructions is covered in the instruction descriptions. A channel terminates the operation when the data count is exhausted. A device terminates the operation by presenting status. When a CCW operation is terminated, either a new CCW is fetched and a new operation is initiated, or an interrupt is generated. Unfortunately, a subchannel on a word channel may, instead of fetching a new CCW or generating an interrupt, return to the available state with no other action being taken. This occurs when the data count for the current operation is exhausted, the data chain, command chain, and monitor flags are all cleared, and the device does not present an external interrupt. This condition can be prevented by setting the monitor flag. The monitor flag set and the data chain and command chain flags cleared specify that an interrupt is to be generated when the data count of the present operation is exhausted.

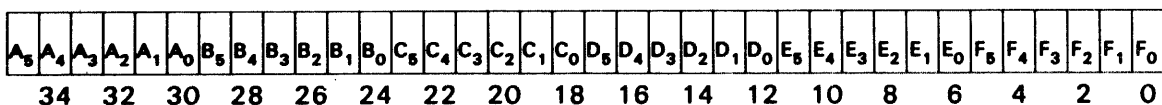
When an operation is terminated, the action taken by the subchannel is determined by the condition that caused the operation to be terminated and by the chain data, chain command, truncated search, SLI, EI chain, PCI, and monitor flags of the CCW flag field. Tables 6-9 and 6-10 illustrate the relationship between the CCW flags and the action taken.

Table 6-6. MSU Data Format - 36-Bit Format, Forward Operation

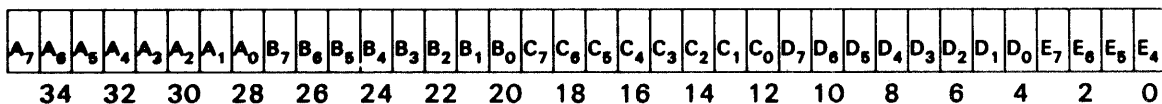
Format A



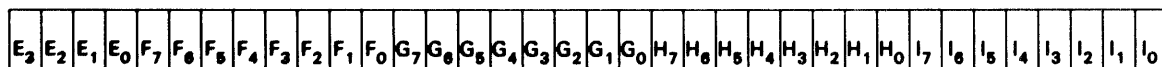
Format B



Format C



Format C.



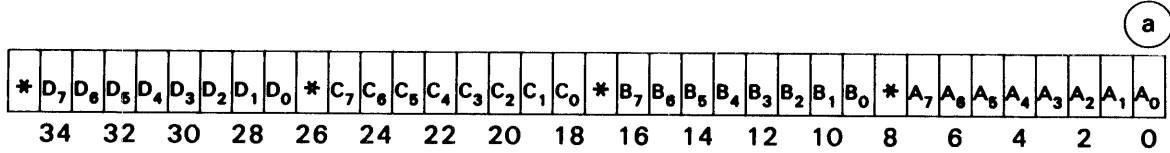
\* For input operations, bits with an asterisk will be written to zero by the IOU. For output operations, bits with an asterisk are ignored.

a The letters indicate the order in which bits are transferred. A indicates the first byte transferred, B indicates the second byte transferred, C indicates the third byte transferred, etc. Numbers indicate the bit position in the byte. A 7 is the most significant bit in the byte, a 6 is the second most significant bit in the byte, a 5 is the third most significant bit in the byte, etc.

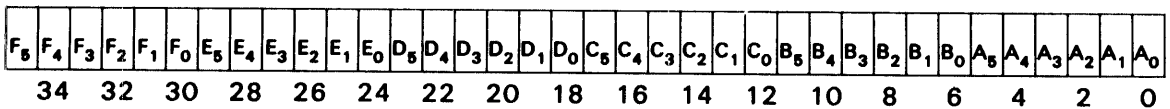


Table 6-7. MSU Data Format - 36-Bit Format, Backward Operation

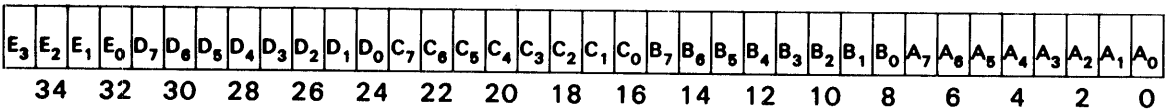
Format A



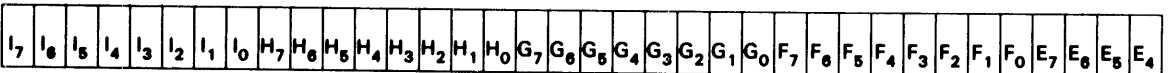
Format B



Format C



Format C



\* For input operations, bits with an asterisk will be written to zero by the IOU. For output operations, bits with an asterisk are ignored.

a The letters indicate the order in which are transferred. A indicates the first byte transferred, B indicates the second byte transferred, C indicates the third byte transferred, etc. Numbers indicate the bit position in the byte. A 7 is the most significant bit in the byte, a 6 is the second most significant bit in the byte, a 5 is the third most significant bit in the byte, etc.

Table 6-8. Format Flags vs Type of Channel

Emulate	Format A	Format B	Format C	Type of Channel	Result
X	X	X	X	Word	The E, A, B, and C flags are ignored on a word channel. The mode (36-bit) is determined by hardware feature.
0	1	0	0	Byte/Block	36-bit quarter-word format (4 bytes per word)
0	0	1	0	Byte/Block	36-bit six-bit packed format (6 bytes per word)
0	0	0	1	Block only	36-bit eight-bit packed format (4 1/2 bytes per word)
X	0	0	1	Byte only	The operation is not initiated and the Program Check subchannel status bit is set.
All other combinations					The operation is not initiated and the Program Check subchannel status bit is set.

X Can be either 0 or 1 (don't care).

Table 6-9. CCW Flags vs Termination Conditions on Byte or Block Multiplexer Channel

CC Chain Data	CC Chain Command	TS Truncated Search	SLI Suppress Length Indication	Data Count Exhausted - No Device Status	Data Count Exhausted - Chaining Status	Data Count Exhausted - Terminate Status	Data Count Not Exhausted - Chaining Status	Data Count Not Exhausted - Terminate Status
0	0	0	0	Stop	End	End	End, IL	End, IL
0	0	0	1	Stop	End	End	End	End
0	0	1	0	Stop	End	End	Truncated Search	End, IL
0	0	1	1	Stop	End	End	Truncated Search	End
0	1	0	0	Wait	Command Chain	End	End, IL	End, IL
0	1	0	1	Wait	Command Chain	End	Command Chain	End
0	1	1	0	Wait	Command Chain	End	Truncated Search	End, IL
0	1	1	1	Wait	Command Chain	End	Truncated Search	End
1	0	0	0	Data Chain	*	*	End, IL	End, IL
1	0	0	1	Data Chain	*	*	End, IL	End, IL
1	0	1	0	Data Chain	*	*	Truncated Search	End, IL
1	0	1	1	Data Chain	*	*	Truncated Search	End, IL
1	1	0	0	Data Chain	*	*	End, II Search	End, IL
1	1	0	1	Data Chain	*	*	End, IL	End, IL
1	1	1	0	Data Chain	*	*	Truncated Search	End, IL
1	1	1	1	Data Chain	*	*	Truncated Search	End, IL

Footnotes for Table 6-9

End	The operation is terminated. If the operation is immediate and has been specified by the first CCW associated with a Sense Release instruction, a condition code of 1 is set, and the status portion of the CSW is stored as part of the execution of the Sense Release instruction. In all other cases, an interrupt is generated in the subchannel when the channel accepts device status.
Stop	The device is signaled to terminate transfer of data, but the subchannel remains in the working state until device status is accepted; at this time an interrupt is generated in the subchannel.
Wait	The device is signaled to terminate transfer of data, but the subchannel remains in the working state until device status is accepted; if the device status is chaining status, a command chain operation is initiated; if the device status is terminate status, an interrupt is generated.
IL	Incorrect length subchannel status is indicated with the interrupt or condition code of 1.
Chain Command	The channel initiates a command chain operation upon receipt of Device End.
Truncated Search	The channel initiates a truncated search operation.
Chain Data	The channel immediately fetches a new CCW for the same operation.  The situation where the count is zero but data chaining is indicated at the time the device provides status cannot validly occur. When data chaining is indicated, the channel fetches the new CCW after transferring the last byte of data designated by the current CCW but before the device provides the next request for data or status transfer. As a result, the channel recognizes the status from the device only after it has fetched the new CCW, which cannot contain a count of zero unless a programming error has been made.

Table 6-10. CCW Flags vs Termination Conditions on Word Channel

```

C C E M P
h h l o C
a a n l
i i C i
n n h t
  a o
D D i r
a a n
t t *
a a
0 0 X 0 0
0 0 X X 1
0 0 X 1 X
1 X X X X
0 1 X X X

```

Data Count Exhausted

(1)  
(2)  
(2)

Data Chain  
Command Chain

Data Count Not Exhausted  
and External Interrupt \*\*

(2)

EI Chain and Tabled Interrupt

```

X X 0 X X
X X 1 X X

```

1. No action is taken. The subchannel is returned to the available state.
  2. The operation is terminated and an interrupt is generated
- \* EI Chain Flag is valid only on ESI subchannels.
- \*\* The situation where the data count is zero and an external interrupt is detected cannot validly occur. When the data count is exhausted, the flags are immediately inspected and the appropriate action is taken before the external interrupt is accepted.

## 6.6. COMMAND CODE

The command code specifies to the channel, and on a byte or block multiplexer channel also to the device, the operation to be performed. The command code assignment is listed in Table 6-11. The symbol X indicates that the bit position is ignored; M identifies a modifier bit used by the control unit or device on the byte or block multiplexer channels. The M bits are ignored on a word channel.

Table 6-11. CCW Command Code

Byte or Block Multiplexer Channel.

<u>Command</u>	<u>Code</u>	<u>Command</u>
Invalid	XXXX 0000	Invalid
Sense	MMMM 0100	Invalid
Transfer in Channel (TIC)	XXX0 1000	Transfer in Channel (TIC)
Store Subchannel Status	XXX1 1000	Store Subchannel Status
Read Backward	MMMM 1100	Invalid
Write	MMMM MM01	Write
Read Forward	MMMM MM10	Read
Control	MMMM MM11	Forced External Function
X = Not Used	M = Not Used (Word Channel)	
	M = Modifier (Byte or Block Multiplexer Channel)	

On a byte or block multiplexer channel, commands that initiate I/O operations (Write, Read, Read Backward, Control, and Sense) cause all eight bits of the command code to be transferred to the I/O device. The modifier bits specify to the device how the operation is to be performed.

Whenever the channel detects an invalid command code during the initiation of a command, the Program Check bit in the subchannel status field is set and the operation is terminated. If the first CCW designated by the CAW contains an invalid command, the operation is terminated, the Program Check subchannel status bit is set and reported by either a condition code of one and an associated CSW or an interrupt. When the invalid code is detected during command chaining, the new operation is not initiated, and an interrupt is presented with the Program Check bit in the subchannel status field set. The command code is ignored during data chaining, unless the Transfer in Channel command is specified.

### 6.6.1. Transfer in Channel Command – TIC

The Transfer in Channel command provides a branching function in the channel. The TIC command provides for chaining between CCWs not located in sequential storage locations. This allows command and buffer loops. A new CCW is fetched from the location designated by the data address field of the TIC command. A new CCW and CCW list are immediately initiated. The TIC command can occur during data chaining or command chaining. The data count field, the format flags, and the CCW flags of a TIC command are not interpreted. The data chain or command chain operation is carried through the Transfer in Channel CCW to the new CCW.

If consecutive TIC commands are detected or if the CCW address of a TIC command is not on a double word boundary, the execution of the CCW list is terminated, and an interrupt is presented with the Program Check bit of the subchannel status field set.

### 6.6.2. Store Subchannel Status Command – SST

The Store Subchannel Status command provides a means of obtaining the data count of a subchannel within a CCW list without having to terminate the execution of the CCW list. When an SST command is detected, a double word CSW is stored at the location specified by the data address field of the Store Subchannel Status CCW. The device and subchannel status fields of the CSW will always be invalid. The device number and next CCW address of the CSW will be valid, and the data count will be the residue data count from the previous CCW. After storing the CSW, the execution of the CCW list is continued.

The data count field, the format flags, and all the CCW flags of an SST command are not interpreted. The SST command is detected only during command chaining.

If the data address field does not specify a double word boundary, the execution of the CCW list is terminated. The Program Check bit of the subchannel status field is set and reported by either a condition code of one and an associated CSW or an interrupt.

## 6.7. DATA TRANSFER

Data transfers are controlled by the data address and data count fields of the CCW. The data address field contains the storage address of the first data to be transferred. The data count field of a CCW specifies the number of bytes or words to be transferred. On a byte or block multiplexer channel, the data count specifies the number of bytes to be transferred. On a word channel ISI interface, the data count specifies the number of 36-bit words to be transferred. On a word channel ESI interface, the data count specifies the number of quarter words or half words to be transferred.

### 6.7.1. Format Flags (E, A, B, and C)

On a word channel the format flags E, A, B, and C are not interpreted. Emulation mode on a word channel is determined on a channel basis by hardware patch wire. No formatting of data is done on a word channel. All transfers of ISI and ESI data are compatible with 1100 Series I/O data transfers for 36-bit operations.

On a byte or block multiplexer channel, the emulation flag (E) and the format flags (A, B, and C) control the formatting of data. If the emulation flag is zero, the data word width is 36 bits. An emulation flag of one is invalid.

The format flags select either the quarter-word format (A), the six-bit packed format (B), or the eight bit-packed format (C). Tables 6-6 and 6-7 illustrate the data formats for forward and backward operations with the 36-bit mode.

Unused bits in format A are zero filled on input operations. With the 36-bit mode of operation, when bytes are transferred to bits 00-07, 09-16, 18-25, and 27-34, the respective bits 08, 17, 26, and 35 will be written to zero.

If an input operation is not completed on an address boundary, leftover bytes and bits within a word are not affected in format A. On the block multiplexer channel with formats B and C, leftover bytes and partial bytes within a full word are zero filled and only full words are written into storage. On the byte multiplexer channel with format B, leftover bytes and bits within a word are left unchanged. With format A individual quarter words (9 bits) are written for the 36-bit mode of operation.

Formats A and B (36-bit mode) are the only valid formats on a byte multiplexer channel. If format C is selected on a byte multiplexer channel or if more than one format is selected, the operation is terminated and the Program Check bit in the subchannel status field is set.

### 6.7.2. Skip Data – SK

The skip data flag in the CCW when set specifies that no data is to be transferred to storage. The skip data flag is defined only for read, read backward, and sense operations. The skip data flag is ignored in all other operations. Skipping affects only the handling of data by the channel. The operation at the I/O device proceeds normally, and data is transferred to the channel. The channel keeps updating the data count, but does not place the information in main storage. When the data count is exhausted, a new CCW is obtained if either the command chain or the data chain flag is set.

Each CCW is controlled by its individual skip data flag. Thus skipping, when combined with data chaining, permits the program to place throughout storage selected portions of a block of data from an I/O device.

### 6.7.3. Data Address Decrement – DAD

The data address decrement flag in the CCW when set specifies that the data address when being updated is to be decremented rather than incremented. The DAD flag is valid for all data transfer operations. The DAD flag affects only the handling of data by the channel. The operation at the I/O device proceeds normally. When the data count is exhausted, a new CCW is obtained if either the command chain or the data chain flag is set, and the new CCW is under the control of its DAD flag.

### 6.7.4. Data Address Lock – DAL

The data address lock flag in the CCW when set specifies that the data address is never updated. The DAL flag is valid for all data transfer operations. The DAL flag affects only the handling of data by the channel. The operation at the I/O device proceeds normally. When the data count is exhausted, a new CCW is obtained if either the command chain or the data chain flag is set, and the new CCW is under the control of its DAL flag.

## 6.8. CHAINING OPERATIONS

When a channel has performed the transfer of data specified by a CCW, it can continue the activity initiated by the SIOF instruction by fetching a new CCW (chaining). Chaining occurs only between CCWs located in successive double word locations in storage. A CCW list is executed in an ascending order of addresses. The address of a new CCW is obtained by adding two to the address of the current CCW. Two CCW lists in noncontiguous storage locations can be connected for chaining purposes by a TIC command. On a byte or block multiplexer channel, all CCWs of a CCW list apply to the I/O device specified in the original SIOF instruction. On a word channel all CCWs of a CCW list apply to the subchannel specified in the original SIOF.

Three types of chaining are provided: data chaining, command chaining, and EI chaining (valid only on an ESI subchannel). The specification of chaining is effectively propagated through a TIC command. When in the process of chaining a TIC command is detected, the CCW designated by the TIC is used for the type of chaining specified in the CCW preceding the TIC CCW.

A chaining operation is initiated when the operation on the present CCW is completed. A CCW operation is completed when either the data count is exhausted or the device presents status. The combination of the data count, device status, chain data flag, chain command, and EI chain flags determine what type of chaining, if any, occurs. Tables 6-9 and 6-10 outline what action is taken under all the combinations of CCW flags and termination conditions.



### 6.8.1. Data Chaining

Data chaining provides software with the capability of changing the data address at any time during the transfer of a block of data. Data chaining may be used to rearrange information as it is transferred between main storage and an I/O device. Data chaining permits blocks of information to be transferred to or from noncontiguous areas of storage, and when used with the skip data flag, data chaining allows the software to place selected portions of a block of data in main storage.

For data chaining, the new CCW fetched by the channel defines a new storage area for the original I/O operation. Execution of the operation at the I/O device is not affected. The contents of the command code field of the new CCW is ignored unless it specifies a TIC command.

Data chaining on the byte multiplexer channel and on word ESI subchannels is executed immediately after the last byte or partial word under control of the current CCW has been transferred to storage or to the device. The old CCW is replaced by the new CCW before another data request from the device is handled. If the device presents status after exhausting the count of the current CCW, but before transferring any data to or from the storage area designated by the new CCW, the action taken by the channel is controlled by the new CCW flags. If a hardware or software error is detected when fetching the new CCW, the operation is terminated and an interrupt is generated. The channel status word (CSW) or tabled status word (TSW) associated with the interrupt will indicate why the operation was terminated.

On the block multiplexer channel and on word ISI subchannels data chain CCWs are prefetched by the channel hardware. Each block multiplexer channel and each word ISI subchannel has an eight word data buffer in the channel hardware to decrease the probability of data overruns. During output operations a data chain is executed immediately after the last byte or word under control of the current CCW has been transferred to the data buffer. The old CCW is replaced by the new CCW and the data buffer is now kept full under control of the new CCW. During input operations the channel hardware prefetches one data chain CCW ahead during channel idle time. After the last byte or word under control of the current CCW has been transferred from the device to the data buffer, the channel continues to accept data under control of the new CCW if the new CCW has been prefetched. If a hardware or software error is detected when fetching the new CCW, the operation is terminated and an interrupt is generated. The CSW associated with the interrupt will indicate why the operation was terminated.

During data chaining the channel hardware prefetches only one CCW ahead and the data associated with that CCW. If the byte count or word count in a data chain CCW is smaller than the data buffer (8 words), the buffering provided is limited to the size of the byte count or word count. For example, if a data chain CCW contains a byte count of one, only one byte of data buffering will be provided for that CCW. When transferring data to or from time dependent devices, the use of data counts smaller than the buffer size increases the probability of data overruns and the smaller the data count, the higher the probability of a data overrun.

### 6.8.2. Command Chaining

A subchannel executes a command chain operation by retrieving a new CCW and beginning execution of the command specified by that CCW. The subchannel is activated and begins executing the new command. On a byte or block multiplexer channel the new command is also passed directly to the device.

Command chaining makes it possible for software to initiate the transfer of multiple blocks of data by means of a single SIOF instruction. It also allows a subchannel to initiate the execution of auxiliary functions and data transfer operations without software interference at the end of each operation. On a byte or block multiplexer channel, command chaining, in conjunction with the status modifier condition, allows the channel to modify the normal sequence of operations in response to signals provided by the I/O device.

During command chaining, the new CCW fetched by the channel specifies a new I/O operation. On a byte or block multiplexer channel, command chaining occurs only when the I/O device presents chaining status, the command chain flag is set, and the data chain flag is not set, and either the byte count equals zero or the SLI flag is set. On a word channel, command chaining occurs only when the data count of the current operation is exhausted, the command chain flag is set, and the data chain flag is not set. When command chaining takes place, the completion of the current operation does not cause an interrupt. The data count indicating the amount of data transferred during the current operation can be obtained by using an SST command as part of the command chain.

Command chaining takes place, and the new operation is initiated only if no hardware error has been detected during the current operation. If a hardware error has been detected, the operation is immediately terminated and an interrupt is generated. Also, if a hardware or software error is detected when trying to initiate the new CCW of a command chain, the operation is terminated, and an interrupt is generated.

An exception to the sequential chaining of CCWs occurs on a byte or block multiplexer channel when the I/O device presents the status modifier condition along with chaining status. When command chaining is specified, the combination of chaining status and the status modifier condition causes the channel to fetch and chain to the CCW whose main storage address is four higher than that of the current CCW.

### 6.8.3. EI Chaining (ESI Word Interface Only)

The EI chaining flag is interpreted only on an ESI subchannel. A subchannel executes an EI chain operation by placing the external interrupt in the status table if the status table subchannel is active, retrieving a new CCW from a storage address four higher than the storage address of the current CCW, and executing the command specified by that CCW. An EI chain is executed only if all of the following conditions are met:

1. The EI chain CCW flag is set.
2. An ESI external interrupt is presented.
3. The status table subchannel is active.
4. No hardware error is detected when making an entry into the status table for the external interrupt.
5. The subchannel is active.

**NOTE:**

*If the data count on a word subchannel is exhausted and neither data chaining nor command chaining is specified, the subchannel is immediately changed from active mode to either status pending or idle mode.*

6. No hardware or software error is detected when retrieving the new CCW.

Extreme care must be taken if the EI chain flag is set along with the chain data and/or chain command flags. If the channel detects exhaustion of the data count first, the data chain or command chain will be performed, and any subsequent EI chain will be under the control of the EI chain flag in the new CCW. If the channel detects the external interrupt first, an EI chain will be performed if the EI chain flag in the current CCW is set. The new CCW used by the channel is different depending on whether the chain is an EI chain, a data chain, or a command chain. If a data chain or command chain is executed, a new CCW whose storage address is two higher than that of the current CCW

is executed. If an EI chain is executed, a new CCW whose storage address is four higher than that of the current CCW is executed.

If a software or hardware error is detected when trying to initiate the new CCW during an EI chain, the operation is terminated, a TSW indicating why the operation was terminated is stored in the status table, and a tabled interrupt request is generated. The subchannel is returned to the available state.

#### 6.8.4. Truncated Search

The truncated search capability provides software with a simple yet effective method of reading or writing multiple records on a disk control unit. A truncated search operation is executed under control of the truncated search CCW flag. A block multiplexer channel executes a truncated search operation by reissuing the command in the preceding CCW and the command in the current CCW when proper status is received from the device before the data count is exhausted.

The following is an example of a truncated search operation:

<u>CCW</u>	<u>COMMAND</u>
1	Search command (This CCW has the CC flag set.)
2	TIC command (TIC back to CCW 1.)
3	Read or Write command (This and only this CCW must have the truncated search flag set.)

The channel retrieves CCW 1 and issues the Search command and search bytes to the device. If the device does not make a compare on the search bytes, device status of Channel End and Device End is presented to the channel. The channel executes a command chain, retrieves the TIC command, and then retrieves the Search command and reissues the Search command to the device. This loop continues until the device makes a compare (finds the correct record) and presents device status of Channel End, Device End, and Status Modifier. Because of the special device status, the channel skips the next CCW (CCW 2) and executes CCW 3. The Read or Write command is issued to the device and the device begins transferring data.

When the device detects the end of a record, device status of Channel End and Device End is presented to the channel. The channel checks the device status, and detects the truncated search CCW flag. The channel begins the truncated search operation by reissuing the previous command (the Search command from CCW 1) and responding to the first request for data with the "command out" interface line. The device automatically makes a compare and presents device status of Channel End, Device End, and Status Modifier. The channel reissues the Read or Write command and data transfer is initiated starting with the residual data count and data address from the previous read/write operation. This procedure is repeated at the end of each record until the byte count is exhausted.

Data chaining after the byte count is exhausted is allowed; however, further truncated search operations are executed only if the active CCW has the truncated search flag set. Command chaining after the byte count is exhausted is also allowed. See Table 6-9 to determine what action the channel takes when either the byte count is exhausted or device status is presented.

The channel unconditionally sets the most significant bit (the M bit) of the Search command every time the Search command is issued to the device as part of a truncated search operation. The M bit set allows the disk device to switch heads when an index mark is detected.

### 6.8.5. Truncated Search Restrictions

There are several programming restrictions for truncated search operations:

1. The truncated search flag is valid only for the block multiplexer channel. If the truncated search flag is set in a CCW for a byte multiplexer channel, the operation is terminated and program check subchannel status is presented to the software.
2. Split status from the control unit (only Channel End for device status) immediately terminates execution of the subchannel program. If the truncated search flag is set and a control unit splits status, the channel does not execute the truncated search operation.
3. In any CCW the suppress length indication flag is ignored if the truncated search flag is set. When the truncated search flag is set in a CCW, a command chaining operation is initiated only if the chain data flag is clear, the chain command flag is set, and the byte count for that CCW has been exhausted.
4. Presentation of any unexpected or abnormal device status immediately terminates execution of the truncated search operation. The execution of the subchannel program is also immediately terminated and the status is presented to the software by way of instruction or interrupt.
5. Truncated search is treated as a strictly byte operation. The formatting and packing is unaffected by truncated search operations. There is one exception that is described in item 6.
6. Truncated search is not treated as a strictly byte operation if the record length in bytes is on a word boundary for format C; i.e., the record length in bytes is one of the following counts:

<u>Format</u>	<u>Byte Counts</u>	<u>General Formula</u>
Format C - 36 bit	3 5,9,14,18,23,27,32,36,40,45	$9x/2 + 5y$ For $x = 0,2,4,6,8,10$ ... (any even number) and $y = 0$ or $1$

If the record length in bytes is on a word boundary for format C, the first byte of data transferred after a truncated search operation is stored or read from the initial byte position for format C. This ensures that each record begins on a word boundary.

If the record length in bytes is on a word boundary in format C, the data is treated as words, not bytes, and the only valid byte counts are byte counts that fit the general formula listed above.

## 6.9. INTERRUPT GENERATION FLAGS

The program controlled interrupt (PCI) flag and the monitor (MON) flag (word channel only) are interrupt generation flags that cause the subchannel to generate an interrupt. The PCI flag generates an interrupt with the PCI bit set in the subchannel status field and the MON flag generates an interrupt with the MON bit set in the subchannel status field.

### 6.9.1. Program Controlled Interruption - PCI

The program controlled interruption provides the software with a means of causing an I/O interrupt during the execution of a CCW list. The PCI flag can be in any CCW of a CCW list, but is ignored on a TIC command or an SST command. Neither the PCI flag nor the associated interrupt affects the execution of the CCW list.

The channel attempts to interrupt the program whenever the PCI bit of the CCW flags is detected during a data transfer. On all shared subchannels and nonshared block multiplexer subchannels, if the channel is presenting an interrupt for another subchannel, no action is taken. The channel will then interrupt the program when the PCI flag is detected during a data transfer and no channel interrupt request is outstanding. Nonshared subchannels on a byte multiplexer or word channel will attempt to table the interrupt after a data transfer. If the status table is valid, a TSW for that subchannel is stored in the status table. If the status table is not valid, no status is presented and the PCI flag is cleared.

A CSW containing the PCI bit may be stored by an interrupt while the operation is still proceeding or by a completion interrupt. Also, the PCI bit of the subchannel status field may accompany other valid subchannel and device status. The PCI condition cannot be detected by an instruction while the subchannel is in the working state.

If chaining occurs before the interrupt due to the PCI flag has been handled, the PCI condition is carried over to the new CCW. This carry-over occurs on both data and command chaining, and in either case, the condition is propagated through the TIC and SST commands. The PCI conditions are not stacked; if another CCW is fetched with the PCI flag set before the interrupt due to the PCI flag of the previous CCW has been handled, only one interrupt takes place. Thus, multiple PCI flags in a CCW list may result in only one interrupt.

### 6.9.2. Monitor – MON (Word Channel Only)

The monitor (MON) flag specifies that an interrupt be generated with the Monitor subchannel status bit set. The interrupt is not presented until the CCW operation with the monitor flag is completed. The Monitor bit of the subchannel status field may accompany other valid subchannel and device status. The monitor flag is interpreted only in the final CCW of a CCW list. If the data chain or command chain flag is set in a CCW, the monitor flag is ignored and no interrupt is presented. If the EI chain and monitor flags are set in a CCW or an ESI subchannel, the execution of the operation determines the subchannel's response. If the data count is exhausted before the external interrupt is received, an interrupt with Monitor subchannel status is generated. If an external interrupt is presented later, the external interrupt will be stored in the status table, but the EI chain will not be executed. If the data count is not exhausted when the external interrupt is received, the external interrupt will be stored in the status table and the EI chain will be executed.

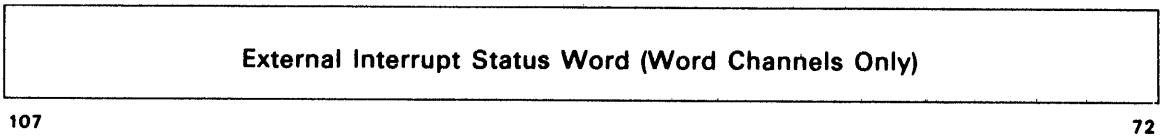
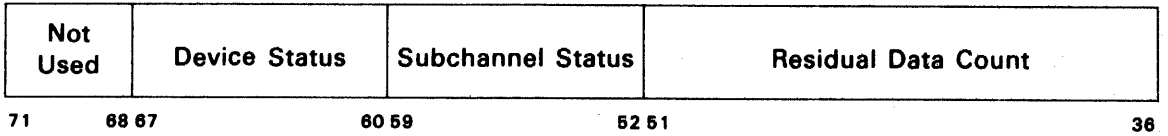
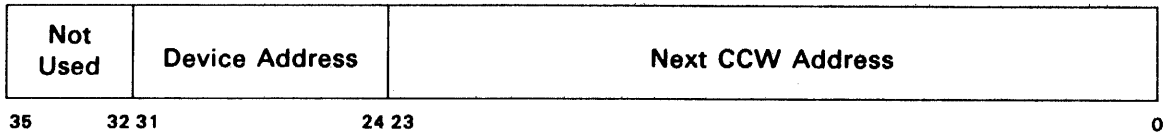
## 6.10. STATUS

I/O status can be separated into the following four categories:

1. Channel Status – Hardware error that cannot be associated with a particular device or subchannel.
2. Status for Noncommunications Subchannels – Status caused by a device, CCW flags, or a hardware or software error on block multiplexer subchannels, byte multiplexer shared subchannels, and word ISI subchannels.
3. Status for Communications Subchannels – Status caused by a device, CCW flags, hardware error, or software error on byte multiplexer nonshared subchannels, and word ESI subchannels.
4. Status for Status Table Subchannel – Status caused by the PCI CCW flag, hardware error, or software error on the status table subchannel.

I/O status is presented either by instruction, SST command, status table, or interrupt. (See Table 6-12.) A standard Channel Status Word (CSW) or Tabled Status Word (TSW) is generated in all five cases. The format is:

CSW or TSW



Subchannel Status

- 52 SIOF Check (Byte and block multiplexer channels only)
- 53 Interface Check
- 54 Control Check
- 55 Data Check
- 56 Not Used
- 57 Program Check
- 58 Monitor (Word channel only)/Incorrect Length (Byte and block multiplexer channels only)
- 59 Program Controlled Interrupt

Table 6-12. IOU Status

Type of Status	Status Generated By	Status Presented to Software
Channel Status	<ol style="list-style-type: none"> <li>Hardware error on the channel to device interface that cannot be associated with a particular device or subchannel.</li> <li>Storage error that cannot be associated with a particular device or subchannel.</li> </ol>	<ol style="list-style-type: none"> <li>Machine Check interrupt</li> </ol>
Status for Noncommunications Subchannels <ol style="list-style-type: none"> <li>All block multiplexer subchannels</li> <li>Byte multiplexer shared subchannels</li> <li>Word channel ISI subchannel</li> </ol>	<ol style="list-style-type: none"> <li>Device</li> <li>CCW flags (PCI or MON)</li> <li>Software error</li> <li>Hardware error</li> </ol>	<ol style="list-style-type: none"> <li>Instruction, or</li> <li>Normal interrupt</li> </ol>
Status for Communications Subchannels <ol style="list-style-type: none"> <li>Byte multiplexer nonshared subchannels</li> <li>Word channel ESI subchannels</li> </ol>	<ol style="list-style-type: none"> <li>Store subchannel status command</li> </ol>	<ol style="list-style-type: none"> <li>Store subchannel status open</li> </ol>
Status for Communications Subchannels <ol style="list-style-type: none"> <li>Byte multiplexer nonshared subchannels</li> <li>Word channel ESI subchannels</li> </ol>	<ol style="list-style-type: none"> <li>Device</li> <li>CCW flags (PCI or MON)</li> <li>Software error</li> <li>Hardware error</li> </ol>	<ol style="list-style-type: none"> <li>Instruction, or</li> <li>Status Table Entry and Table interrupt</li> </ol>
Status for Status Table Subchannel	<ol style="list-style-type: none"> <li>Store subchannel status command</li> </ol>	<ol style="list-style-type: none"> <li>Store subchannel status operation</li> </ol>
Status for Status Table Subchannel	<ol style="list-style-type: none"> <li>CCW flag (PCI)</li> <li>Software error</li> <li>Hardware error</li> </ol>	<ol style="list-style-type: none"> <li>LTCW Instruction, or</li> <li>Normal interrupt with bit 24 the IAW set</li> </ol>

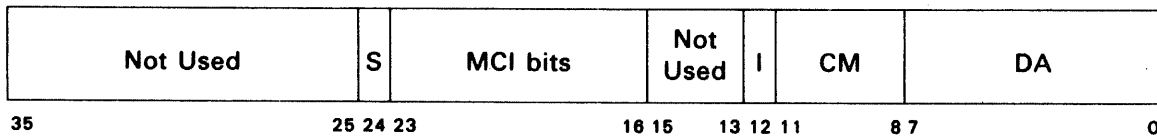
Device Status

60	Unit Exception	Byte and block multiplexer channels only
61	Unit Check	Byte and block multiplexer channels only
62	Device End	Byte and block multiplexer channels only
63	Channel End	Byte and block multiplexer channels only
64	Busy	Byte and block multiplexer channels only
65	Control Unit End	Byte and block multiplexer channels only
66	Status Modifier	Byte and block multiplexer channels only
67	Attention	

On a byte or block multiplexer channel, the device status field contains the actual status presented by the device. On a word channel the only valid device status bit is the Attention bit (bit 67). On a word channel the Attention bit set indicates that bits 72-107 of the CSW contain an external interrupt status word. The Attention bit cleared indicates that bits 72-107 of the CSW are invalid.

When an I/O interrupt is acknowledged by the processor, an interrupt address word (IAW) is stored in a fixed location of low order storage. For any I/O interrupt the standard IAW format is:

IAW



S Status Table Indicator

MCI Machine Check Indicator Bits

I IOU Number

CM Channel Module Number

DA Device Address

Each one of a possible four processors has one reserved address for the IAW and three reserved addresses for the CSW (See Table 6-13). Status is presented to the processor that caused a CSW or IAW to be stored. For example, if processor 0 executes an I/O instruction and receives a condition code of 1, the CSW is stored in the reserved CSW address locations of processor 0. Or if processor 3 acknowledges an I/O interrupt, the IAW and CSW are stored in the reserved IAW and CSW address locations of processor 3. Status is reported only once and only through one path. If a subchannel that is holding status and presenting an interrupt request is then interrogated by an I/O instruction, it will present that status to either the interrupt or the I/O instruction, but not both. The channel attempts to cancel the interrupt. If the interrupt cancellation is successful, the status will be presented to the instruction and the condition code will equal 1. If the interrupt cancellation fails, the status will be presented to the interrupt. The I/O instruction will then receive a condition code of 2.



Table 6-13. IOU Fixed Addresses

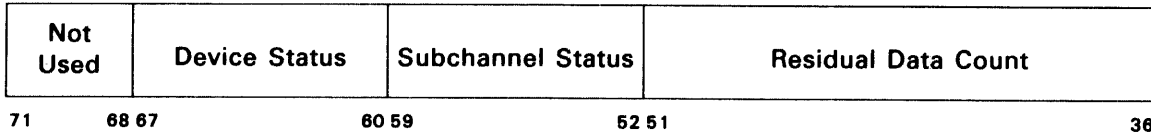
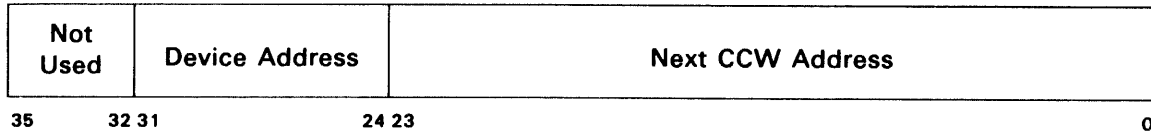
XXXXX 240	PROCESSOR 0 CAW WORD 0
XXXXX 241	PROCESSOR 0 CAW WORD 1
XXXXX 242	
XXXXX 243	
XXXXX 244	PROCESSOR 1 CAW WORD 0
XXXXX 245	PROCESSOR 1 CAW WORD 1
XXXXX 246	
XXXXX 247	
XXXXX 250	
XXXXX 251	
XXXXX 252	
XXXXX 253	
XXXXX 254	
XXXXX 255	
XXXXX 256	
XXXXX 257	
XXXXX 260	PROCESSOR 0 IAW
XXXXX 261	PROCESSOR 0 CSW WORD 0
XXXXX 262	PROCESSOR 0 CSW WORD 1
XXXXX 263	PROCESSOR 0 CSW WORD 2
XXXXX 264	PROCESSOR 1 IAW
XXXXX 265	PROCESSOR 1 CSW WORD 0
XXXXX 266	PROCESSOR 1 CSW WORD 1
XXXXX 267	PROCESSOR 1 CSW WORD 2
XXXXX 270	
XXXXX 271	
XXXXX 272	
XXXXX 273	
XXXXX 274	
XXXXX 275	
XXXXX 276	
XXXXX 277	

### 6.11. INSTRUCTION STATUS

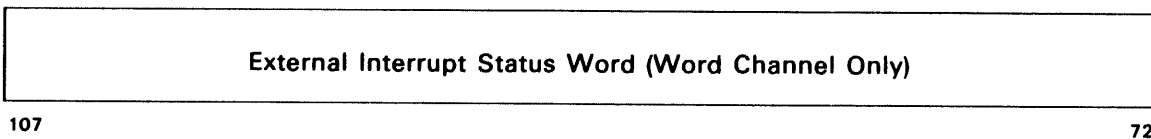
If the addressed subchannel or, in the case of a TIO or HDV instruction, the addressed device is in the interrupt pending state for an I/O instruction, or if the execution of the I/O instruction generates status conditions, the channel stores a CSW in the proper reserved storage addresses and presents a condition code of 1. There are two exceptions: a CSW is not stored for a subchannel in the interrupt pending state if an interrupt request for that subchannel has been presented to the processor and cancellation of that interrupt request was unsuccessful. Also, a CSW is not stored if the addressed subchannel is holding status for a device other than the addressed device. Note that a TSC instruction addresses only a subchannel and not a device.

The response of a condition code of 1 to an I/O instruction always indicates that a CSW has been stored by the channel, and that the addressed subchannel has been set to the available state. The status in the CSW may pertain to either a previous operation or the attempted execution of the present instruction.

CSW for I/O Instruction



On word channels bits 60–66 will be cleared. Bit 67 will be set only if bits 72–107 of the CSW contain an external interrupt.

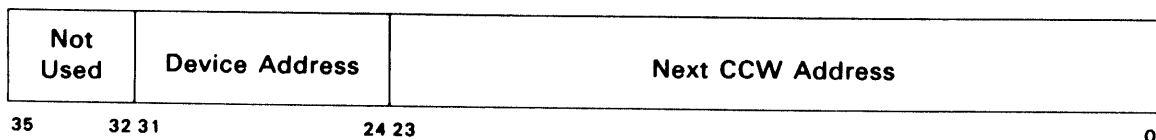


**6.12. STATUS TABLE**

The status table subchannel controls the status table. The status table subchannel is loaded by an LTCW instruction. An LTCW instruction initiates the execution of a status table control word (STCW) list by the status table subchannel. Bits 36–59 of the CAW contain the address of the first STCW of the STCW list. An STCW contains the data count and the starting address for the status table. The command code field is checked only for a TIC command. Any other command code value activates the status table subchannel. The chain data (CD) and program controlled interruption (PCI) flags are the only valid flags in an STCW. All the other STCW flags are ignored. Even if the status table subchannel is active, a new LTCW instruction is accepted, a new STCW list is initiated, and the status table subchannel is loaded with the first STCW of the new list.

After each entry (TSW) stored in the status table, the data address field of the status table is incremented by two (byte multiplexer channel) or four (word channel) and the data count is decremented by two (byte multiplexer channel) or four (word channel). If the status table subchannel detects a data count of zero or a hardware error when attempting to make an entry in the status table, the operation of the status table subchannel is terminated and a normal interrupt request is generated.

TSW for Nonshared Byte Multiplexer Subchannels



Not Used	Device Status	Subchannel Status	Residual Data Count
71	68 67	60 59	52 51
			36

**TSW for ESI Word Subchannels**

Not Used	Device Address	Next CCW Address
35	32 31	24 23
		0

Not Used	EI	0 0 0 0 0 0 0	Subchannel Status	Residual Data Count
71	68 67 66	60 59	52 51	
				36

EI 0 means bits 72-107 are meaningless

1 means bits 72-107 contain external interrupt status word

External Interrupt Status Word if Bit 67 is Set
107
72

Not Used
144
108

On a word channel the status table subchannel must be active before any ESI device requests for either data or status are handled. This restriction prevents ESI device requests from corrupting initial load operations.

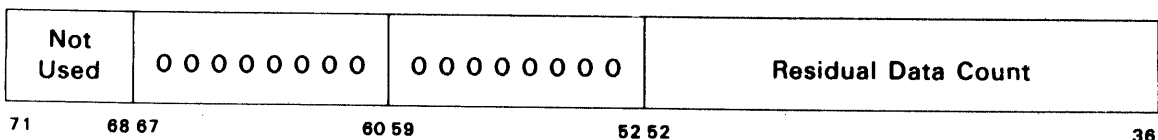
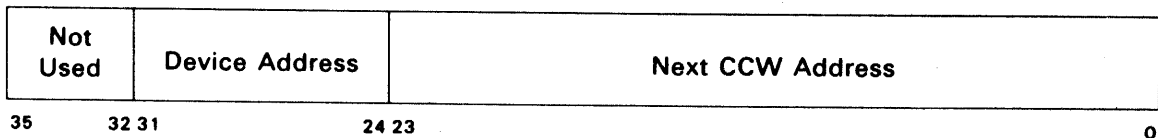
Status from the status table subchannel is reported via I/O instruction or normal interrupt. The device status field of a status table subchannel CSW will always be zero filled. The only valid subchannel status bits and the conditions generating the particular subchannel status bit are the following:

1. Channel Control Check (Bit 54)
  - a. A hardware fault was detected during retrieval of a status table control word.
2. Channel Data Check (Bit 55)
  - a. A hardware fault was detected when making an entry in the status table.
3. Program Check (Bit 57)
  - a. The status table CAW first STCW address did not specify a double word boundary.
  - b. The original STCW data count field equaled zero and the command was not Transfer in Channel.
  - c. The Transfer in Channel command was specified in successive STCWs.
  - d. The STCW address specified by a TIC command is not on a double word boundary.
  - e. The byte multiplexer channel STCW data address did not specify a double word boundary.
  - f. The ESI word channel status table CCW data address did not specify a quadruple word boundary.
  - g. The byte multiplexer channel STCW data count was not a multiple of two.
  - h. The ESI word channel STCW data count was not a multiple of four.
  - i. The channel attempted to read STCW from a location outside of available storage.
  - j. The STCW data address specified a location outside the available storage.
  - k. The STCW data count field was decreased to zero and data chaining was not indicated.

### 6.13. STORE SUBCHANNEL STATUS – SST

The only useful status resulting from an SST command will be the residual data count. The device and subchannel status fields will always be zero. Valid device status, subchannel status, or external interrupt status cannot be presented by way of an SST command. This status is reported only by way of interrupt, instruction, or the status table.

#### CSW for the Store Subchannel Status Command



## 6.14. SUBCHANNEL STATUS

The subchannel status bits are set when particular CCW flags are set or when a hardware or software error is detected by the channel. Subchannel status indications are presented to the software in the subchannel status field of a CSW or TSW.

### 6.14.1. SIOF Device Check (Bit 52) (Byte or Block Multiplexer Channel Only)

The SIOF Device Check subchannel status bit indicates that a device that was to be initiated by a previously issued SIOF instruction is not operational (not installed or offline).

### 6.14.2. Interface Control Check (Bit 53)

The Interface Control Check subchannel status bit indicates that a hardware error in the channel to device interface was detected by the channel. The hardware error was detected during one of the following operations:

1. A byte or block multiplexer channel detected a device interface parity error during the transfer of device status.
2. A word channel detected a device interface parity error during the transfer of an external interrupt status word.
3. A byte peripheral device responded with an address other than the address specified by the channel during a channel initiated selection sequence.
4. A byte peripheral device became non-operational during a command chaining.
5. A byte peripheral interface control signal sequence error was detected.

### 6.14.3. Channel Control Check (Bit 54)

The Channel Control Check subchannel status bit indicates that a hardware error was detected by the channel when attempting to read or write a control word from storage. The hardware error was detected during one of the following operations:

1. The channel was attempting to read the second word of the CAW for an SIOF instruction.
2. The channel was attempting to read a CCW or STCW.
3. The channel was attempting to store a status word for the store subchannel status command.
4. The channel was attempting to read from main storage a control word for one of the 128 nonshared subchannels.

### 6.14.4. Channel Data Check (Bit 55)

The Channel Data Check subchannel status bit indicates that a hardware error was detected during the transfer of data from the device to the channel, from the channel to main storage, or from main storage to the channel.

#### 6.14.5. Not Used (Bit 56)

#### 6.14.6. Program Check (Bit 57)

The Program Check subchannel status bit indicates that software error was detected by the channel. The software error was one of the following:

1. The CAW first CCW address did not specify a double word boundary.
2. The status table CAW first STCW address did not specify a double word boundary.
3. A CCW or STCW contained an invalid command for an operation other than a data chain.
4. The CCW or STCW data count equaled zero and the command was neither Transfer in Channel nor Store Subchannel Status.
5. The Transfer in Channel command was specified in successive CCWs or STCWs.
6. An ESI word channel CCW contained the Forced EF command with a data count not equal to one.
7. The CCW or STCW address specified by a TIC command was not on a double word boundary.
8. The Store Subchannel Status command data address field did not specify a double word boundary.
9. The truncated search (TS) flag was specified in a byte multiplexer CCW.
10. Any byte multiplexer channel CCW or a block multiplexer channel command CCW did not specify only one format.
11. A byte multiplexer channel CCW specified format C.
12. The byte multiplexer channel STCW data address field did not specify a double word boundary.
13. The ESI word channel status table CCW data address field did not specify a quadruple word boundary.
14. The byte multiplexer channel STCW data count field was not a multiple of two.
15. The ESI word channel STCW data count field was not a multiple of four.
16. The channel attempted to read a CCW or STCW from a location outside of available storage.
17. A CCW or STCW data address specified a location outside the available storage.
18. The STCW data count field was decreased to zero and data chaining was not indicated.

#### 6.14.7. Monitor (Bit 58) (Word Channel Only)

The Monitor subchannel status bit indicates that a CCW list on a word subchannel has been completed. The Monitor subchannel status bit is set and an interrupt is generated when an operation is completed on a CCW that has the monitor CCW flag set and does not have either the data chain flag set or the command chain flag set. If a CCW has either the data chain or the command chain flag set, the monitor flag is not interpreted and no interrupt is generated.

### 6.14.8. Incorrect Length (Bit 58) (Byte and Block Multiplexer Channels Only)

The Incorrect Length subchannel status bit indicates that the number of bytes specified in the CCWs for an I/O operation was not equal to the number of bytes requested or offered by the device. The incorrect length indication is presented only if the active CCW has neither the truncated search flag nor the suppress length indication flag set and the data chain flag is not set. Detection of an incorrect length condition causes the operation to be terminated and an interrupt to be generated. See Table 6-9 for the affect of the CD, CC, SLI, and TS flags on the indication of incorrect length.

### 6.14.9. Program Controlled Interrupt (Bit 59)

When the channel detects the PCI flag during a data transfer, an interrupt and/or a table entry in the status table is attempted for that subchannel. The PCI subchannel status bit indicates that a PCI flag had been detected in a CCW or STCW list. Because the program controlled interruption does not affect the execution of a CCW or STCW list, the detection of several PCI flags in a CCW or STCW list may result in only one interrupt with the PCI subchannel status bit set. However, each PCI flag never generates more than one interrupt or one TSW entry.

## 6.15. DEVICE STATUS

If the device presents termination status on a byte multiplexer shared subchannel or a block multiplexer subchannel, the operation is terminated and an interrupt is generated if an interrupt from another subchannel is not already being presented. If an interrupt is being presented, the status is stacked in the device. When the interrupt mechanism becomes available, the status is accepted from the device, and an interrupt is generated. If the device presents termination status on a byte multiplexer nonshared subchannel, the operation is terminated, and a status table entry is executed if the status table subchannel is active. If the status table subchannel is inactive, no status table entry is made, and the device status is lost.

When a device on a byte or a block multiplexer channel presents chaining status, a command chain is performed only if the command chain flag is set and the data chain flag is not set. Otherwise, the operation is terminated, and an interrupt is presented as previously explained.

On a word channel ISI interface an external interrupt terminates the operation and generates an interrupt. On a word channel ESI interface an external interrupt terminates the operation and generates a status table entry if the status table is active. If the status table is inactive, no table entry is made, and the external interrupt is lost. There is one exception for terminating an operation. An external interrupt will generate an entry into the status table and cause an EI chain to be performed if the operation is still active, the EI chain flag is set, and the status table subchannel is active.

The byte or block multiplexer channel device status bits are:

<u>Status Codes</u>	<u>Status</u>
1000 0000	Attention
0100 0000	Status Modifier
0010 0000	Control Unit End
0001 0000	Busy
0000 1000	Channel End

0000 0100	Device End
0000 0010	Unit Check
0000 0001	Unit Exception

For the word channel the device status field will always be zero with the exception of bit 67. If bit 67 of the CSW or TSW is set, bits 72–107 contain an external interrupt status word. If bit 67 is cleared, bits 72–107 of the CSW or TSW are meaningless.

After presenting an interrupt with any device status or any subchannel status other than the PCI bit, the subchannel is returned to the available state. The associated device, however, may not be available because device status may have split or the device may have not presented status by the time the interrupt was acknowledged. After initiating a CCW list, the software may have to handle any number of interrupts before both the subchannel and device are available.

**Programming Note:** All bits of the subchannel status and device status fields must be investigated for each CSW or TSW because several device status bits and/or several subchannel status bits may be set in one CSW or TSW.

## 6.16. DATA CHAINING PRECAUTIONS

There are several precautions that should be taken during data chaining operations. Data chaining with small data counts on high-speed devices that are capable of data overruns should be avoided. Also, the following precautions should be noted when data chaining on the byte or block multiplexer channels:

1. On the byte multiplexer channel switching of the byte packing formats (the E, A, B, and C CCW flags) is allowed between data chained CCWs and command chained CCWs. On the block multiplexer channel the format flags in the command CCW specify the format for the entire data chained CCW list. On the block multiplexer channel the format flag field in all data chained CCWs is ignored and the format flags can only be changed by executing a command chained CCW.
2. Format A (36-bit mode) – If a CCW byte count for an operation is not completed on a full word boundary, leftover bytes in the final data word are unaffected.
3. Byte multiplexer channels – Format B (36-bit mode) – If a CCW byte count for an input operation is not completed on a full word boundary, leftover bytes in the final data word are unaffected.
4. Block multiplexer channels – Format B (36-bit mode) – If a CCW byte count for an input operation is not completed on a full word boundary, leftover bytes in the final data word are zero filled. Only full 36-bit words are transferred to main storage in format B on the block multiplexer channel.
5. Formats A and B (36-bit mode) – If a CCW byte count is not completed on a full word boundary and data chaining is indicated, the first byte transferred under control of the new CCW is stored in or read from the initial byte position for that format.
6. Format C (36-bit mode) – If a CCW byte count for an input operation is not completed on a full word boundary, the leftover bytes and partial bytes to complete only that word are zero filled. Only full words are transferred to storage in format C. The one exception is partial byte remainders. See item 7.



7. **Format C (36-bit mode)** – If a CCW byte count is exhausted on a partial byte boundary (a boundary that requires the final byte to be split into two words for input or a boundary that requires part of the final byte to be taken from another word for output) and data chaining is not indicated, the partial byte remainder is thrown away on input. On output a new word is not retrieved from storage and the final byte is formed by concatenating the partial byte with unpredictable data.
8. **Format C (36-bit mode)** – If a CCW byte count is not completed on a full word boundary and data chaining is indicated, the first byte transferred under control of the new CCW is stored or read from the initial byte position for format C. There is one exception. If a CCW byte count is exhausted on a partial byte boundary and data chaining is indicated, the packing or unpacking of data continues as if the data chain never occurred. For example, on input the first part of the final byte (the byte that is on the partial byte boundary) is transferred to storage under control of the original CCW. The partial byte remainder is then transferred to storage under control of the new CCW.
9. **Block multiplexer channel and ISI word interfaces** – Because the channel prefetches one CCW ahead during data chaining, there is one restriction on dynamically modifying CCW lists during input data chaining operations. If a channel is executing CCW A and data chaining to CCW B is specified, CCW B cannot be part of the data buffer being retrieved by CCW A because the channel may have prefetched CCW B before CCW B was transferred to storage as part of the data buffer of CCW A.
10. **Word channels** – Certain chaining operations can be executed by either a data chain or command chain. In these situations always use data chaining.

See Table 6-14 for examples of the above items.

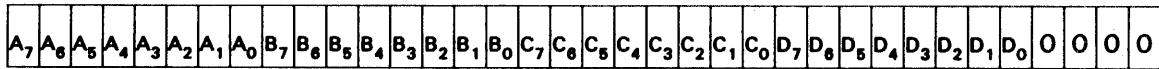




Table 6-14. Byte Data Packing on Abnormal Boundaries (continued)

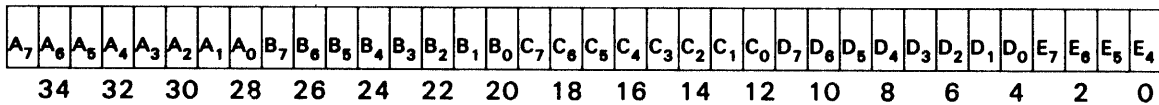
36-Bit Format C – Forward Operation

Storage Address Specified by CCW1



Data count = 4 or status after 4 bytes

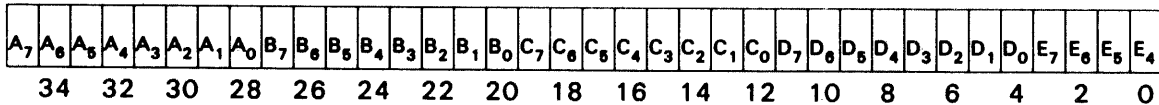
Storage Address Specified by CCW1



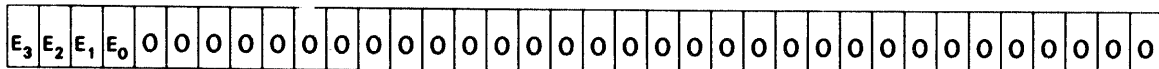
Data count = 5

E<sub>3</sub> to E<sub>0</sub> are thrown away on input and created from A<sub>7</sub> to A<sub>4</sub> on output.

Storage Address Specified by CCW1



Storage Address Specified by CCW1



Status after 5 bytes with data count greater than 5

E<sub>3</sub> to E<sub>0</sub> is stored on input, E<sub>3</sub> to E<sub>0</sub> is transferred to the device.

0 These bit positions set to zero by hardware on input and are ignored on output.



Table 6-14. Byte Data Packing on Abnormal Boundaries (continued)

36-Bit Format C – Forward Operation

Storage Address Specified by CCW1

A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	B <sub>7</sub>	B <sub>6</sub>	B <sub>5</sub>	B <sub>4</sub>	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	C <sub>7</sub>	C <sub>6</sub>	C <sub>5</sub>	C <sub>4</sub>	C <sub>3</sub>	C <sub>2</sub>	C <sub>1</sub>	C <sub>0</sub>	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	E <sub>7</sub>	E <sub>6</sub>	E <sub>5</sub>	E <sub>4</sub>					
34	32	30	28	26	24	22	20	18	16	14	12	10	8	6	4	2	0																							

Storage Address Specified by CCW2

E <sub>3</sub>	E <sub>2</sub>	E <sub>1</sub>	E <sub>0</sub>	F <sub>7</sub>	F <sub>6</sub>	F <sub>5</sub>	F <sub>4</sub>	F <sub>3</sub>	F <sub>2</sub>	F <sub>1</sub>	F <sub>0</sub>	G <sub>7</sub>	G <sub>6</sub>	G <sub>5</sub>	G <sub>4</sub>	G <sub>3</sub>	G <sub>2</sub>	G <sub>1</sub>	G <sub>0</sub>	H <sub>7</sub>	H <sub>6</sub>	H <sub>5</sub>	H <sub>4</sub>	H <sub>3</sub>	H <sub>2</sub>	H <sub>1</sub>	H <sub>0</sub>	I <sub>7</sub>	I <sub>6</sub>	I <sub>5</sub>	I <sub>4</sub>	I <sub>3</sub>	I <sub>2</sub>	I <sub>1</sub>	I <sub>0</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

Data count = 5    Data chained to data count = 4

Storage Address Specified by CCW1

A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	B <sub>7</sub>	B <sub>6</sub>	B <sub>5</sub>	B <sub>4</sub>	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	C <sub>7</sub>	C <sub>6</sub>	C <sub>5</sub>	C <sub>4</sub>	C <sub>3</sub>	C <sub>2</sub>	C <sub>1</sub>	C <sub>0</sub>	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	E <sub>7</sub>	E <sub>6</sub>	E <sub>5</sub>	E <sub>4</sub>						
34	32	30	28	26	24	22	20	18	16	14	12	10	8	6	4	2	0																								

Storage Address Specified by CCW1

E <sub>3</sub>	E <sub>2</sub>	E <sub>1</sub>	E <sub>0</sub>	F <sub>7</sub>	F <sub>6</sub>	F <sub>5</sub>	F <sub>4</sub>	F <sub>3</sub>	F <sub>2</sub>	F <sub>1</sub>	F <sub>0</sub>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
34	32	30	28	26	24	22	20	18	16	14	12	10	8	6	4	2	0																								

Storage Address Specified by CCW2

G <sub>7</sub>	G <sub>6</sub>	G <sub>5</sub>	G <sub>4</sub>	G <sub>3</sub>	G <sub>2</sub>	G <sub>1</sub>	G <sub>0</sub>	H <sub>7</sub>	H <sub>6</sub>	H <sub>5</sub>	H <sub>4</sub>	H <sub>3</sub>	H <sub>2</sub>	H <sub>1</sub>	H <sub>0</sub>	I <sub>7</sub>	I <sub>6</sub>	I <sub>5</sub>	I <sub>4</sub>	I <sub>3</sub>	I <sub>2</sub>	I <sub>1</sub>	I <sub>0</sub>	J <sub>7</sub>	J <sub>6</sub>	J <sub>5</sub>	J <sub>4</sub>	J <sub>3</sub>	J <sub>2</sub>	J <sub>1</sub>	J <sub>0</sub>	0	0	0	0
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	---	---	---	---

Data count = 6    Data chained to data count = 4

## 6.17. SUBCHANNEL EXPANSION FEATURE AND CHANNEL BASE REGISTER

As described in 6.2.2 and 6.3.1, the subchannel expansion feature enables the channel to maintain the control words for the four or eight (depending on the option) most recently active subchannels in the channel while storing the control words of the remaining 120 or 124 subchannels in main storage. Four storage addresses per subchannel and 512 addresses per channel must be reserved for each channel with the subchannel expansion feature. These addresses are a hardware scratch pad and are for hardware use only and should not be interrogated or changed by software except during initialization or error recovery. During system initialization the software must initialize these addresses by setting equal to  $1_{16}$  bits 24–27 of every fourth address (each address with bits 00–01 equal to 0). The format of the control words held in storage is shown in Table 6–15.

If the mode bits 27–24 =  $0001_2$ , all four words are not used. All other values of the mode bits indicate a hardware fault.

Each channel that has the subchannel expansion feature has a channel base register that consists of 15 bits. The channel base register specifies the location of the 512 addresses that are the channel's scratch pad. The channel base register provides bits 09 through 23 of the storage addresses that are used when swapping subchannel control words between main storage and the channel.

## 6.18. MASK REGISTER

The interrupt mask register is loaded with the contents of bits 36–71 of the CAW during an LCR instruction that has bit 00 of the CAW set. One interrupt mask register is provided in each IOU. This register provides the capability of determining which channel modules are allowed to present communications or noncommunications interrupts. It also provides a means of selecting which processor or processors the interrupts are to be sent. The register is broken into four bytes. Two bytes for each processor. These two bytes are then separated into communication or noncommunication interrupts. The interrupt mask register has the format shown in Table 6–16. A set bit suppresses interrupts of the specified type from the corresponding channel module to the corresponding processor; i.e., a one bit in position 39 suppresses the reporting of noncommunications interrupts on channel module 3 to processor zero. If bit position 57 is also set, the reporting of noncommunications interrupts on channel module 3 is completely suppressed. An interrupt that is completely suppressed will be reported when either of the corresponding mask bits is later changed to zero. Any interrupts that are currently being presented to the processor when an LCR instruction is received will be reported before the interrupt mask takes effect. In a unit processor system, software must mask out both communications and noncommunications interrupts to the nonexistent processor during system initialization and each time the LCR instruction is used.

Table 6-15. Scratch Pad Formats for Subchannel Expansion Feature

Not Used	Format Control	Mode	Data Address		
35	32 31	28 27	24 23		0

Not Used	CCW Flags	Format Flags	Not Used	Data Count
35	32 31	24 23	20 19	16 15

Not Used	Device Address	Next CCW Address		
35	32 31	24 23		0

Not Used				
35				0

Format above for:

Mode Bits 27-24 = 0010<sub>2</sub> or 1000<sub>2</sub> or 1001<sub>2</sub> or 1010<sub>2</sub> or 1100<sub>2</sub>

Not Used	Not Used	Mode	Device Address	Not Used	
35	32 31	28 27	24 23	16 15	0

Not Used	Device Status	Subchannel Status	Data Count	
35	32 31	24 23	16 15	0

Not Used	Device Address	Next CCW Address		
35	32 31	24 23		0

Not Used				
35				0

Format above for:

Mode Bits 27-24 = 0000<sub>2</sub> or 0011<sub>2</sub>



Table 6-16. Interrupt Mask Register

Processor 1	
Bit Positions of the CAW	71 70 69 68 67 66 65 64 63      62 61 60 59 58 57 56 55 54
Channel Module Number	* 7 6 5 4 3 2 1 0      * 7 6 5 4 3 2 1 0
Interrupt Type	Communications      Noncommunications
Processor 0	
Bit Position of the CAW	53 52 51 50 49 48 47 46 45      44 43 42 41 40 39 38 37 36
Channel Module Number	* 7 6 5 4 3 2 1 0      * 7 6 5 4 3 2 1 0
Interrupt Type	Communications      Noncommunications

### 6.19. INITIAL LOAD

The initial load capability is included in each channel. The initial load operation is performed in only the 36-bit mode of operation. On a byte or block multiplexer channel, format C (8-bit packed) is specified, the starting data address is set to MSR, and the byte count is set to  $4608_{10}$ . On a word channel, the starting data address is set to MSR and the word count is set to  $1024_{10}$ . The size of the initial load boot block is determined by the channel and the device. The channel will terminate the operation after  $1024_{10}$  full 36-bit words have been loaded into storage. If the device presents status before  $1024_{10}$  words have been loaded, the channel immediately terminates the operation and presents an interrupt to the processor.

### 6.20. BACK-TO-BACK OPERATION (Word Channel Only)

The word channel back-to-back capability is installed by hardware feature and allows the software to execute block transfers or scatter/gather operations via an I/O channel. Two ISI interfaces on the same channel are required, an output interface and an input interface. The back-to-back interfaces must be initialized after the IOU is master cleared, after any hardware or software error on the back-to-back interfaces, or after a back-to-back operation that did not have the output buffer data count equal to the input buffer data count. The back-to-back interfaces are initialized by using the following procedure:

1. To the output interface, issue an SIOF instruction that initiates the execution of one CCW. The CCW must have a forced external function command and a data count of one.
2. Handle the external interrupt from the input interface. This external interrupt was generated by the forced external function operation on the output interface.

To execute a back-to-back operation the following procedure must be followed:

1. Issue an SIOF instruction to the input interface to activate the input buffer.
2. Issue an SIOF instruction to the output interface to activate the output buffer.

Once the back-to-back interfaces are initialized, many back-to-back operations may be performed by following the above procedure for each operation. Data chaining is allowed on both the output and input interfaces for each back-to-back operation, but the total output buffer data count must equal the total input buffer data count. The use of the DAD, SK, DAL, PCI, and MON CCW flags is allowed, but command chaining is prohibited.

## 6.21. PRIORITIES

The control module establishes data handling priority among the eight channel modules. Channel module 0 has the highest priority and channel module 7 has the lowest priority. The channel module gives highest priority to data transfers, second highest priority to interrupts, and lowest priority to instructions. Word channel data handling priority is a homing priority with interface A having the highest priority, then interfaces B, C, and D with interface D having the lowest priority.

## 6.22. BASIC PROGRAMMING PROCEDURE

The programmer should use the following basic procedure in order to execute a series of operations on a byte or block multiplexer device or a word subchannel:

1. Build the list of CCWs, making sure that the correct flags in each CCW are set. Also, build any necessary external function words or data buffers.
2. Load the address of the first CCW in Xa bits 00-23.
3. Load the u and Xm registers such that u + Xm bits 00-12 specifies the IOU, channel, and device numbers.
4. Disable I/O interrupts.
5. Issue the SIOF instruction.
6. Test the condition code to determine the result of the SIOF instruction. (Note that the next instruction is skipped if the condition code equaled 0 and the processor did not time out the instruction.)
7. If the instruction is not timed out by the processor and a condition code of 0 is received, enable I/O interrupts and continue with the processor program. If another condition code is received or the instruction is timed out, the appropriate action should be taken.
8. Wait for the I/O interrupt or interrupts. Use the resultant status to determine if the CCW list was executed successfully. If the CCW list was terminated before it was completed, the status will contain enough information to determine how much of the CCW list was executed, and why the CCW list was terminated before it was completed.

### 6.23. PROGRAMMING EXAMPLES

For an example of the block multiplexer channel CCW list see Figure 6-3. The execution of this CCW list is initiated by an SIOF instruction with a CCW address of  $B0_{16}$ . The CCW list is executed as follows:

1. The channel reads the first CCW and issues the read command to the device.
2. Nine bytes are transferred from the device to the channel, but none of the bytes are written into storage because the skip data flag is set.
3. The device presents Channel End and Device End status (chaining status).
4. The channel initiates the command chain and issues the Write command to the device.
5. Thirty-six bytes are transferred from the channel to the device. All the bytes are transferred from the same storage address because the data address lock flag is set.
6. The channel executes a data chain and transfers two bytes from storage address  $FO_{16}$  and two bytes from storage address  $EF_{16}$  because the data address decrement flag is set.
7. The device presents Channel End and Device End status (chaining status).
8. The channel initiates the command chain and executes the Store Subchannel Status command. A two word CSW is stored at storage address  $44_{16}$ . In the CSW the next CCW address field equals  $B8$ , the data count field equals zero, and the subchannel and device status fields equal zero.
9. The channel continues the command chain and issues the read backward command to the device and then terminates the operation because an illegal combination of format flags is detected.
10. The device presents Channel End and Device End status.
11. The channel accepts the device status and presents an interrupt request to the processor.
12. The interrupt is acknowledged and a CSW is written with the next CCW address field equal to  $BA_{16}$ , the data count field equal to  $1_{16}$ , the Program Check and PCI subchannel status bits set, and the Channel End and Device End device status bits set.



The interrupt mechanism was assumed to be busy during the execution of this CCW list causing the PCI's to be overlaid.

For an example of the word channel ISI interface CCW list see Figure 6-4. The execution of this CCW list is initiated by an SIOF instruction with a CCW address of A2. The CCW list is executed as follows:

1. The channel reads the first CCW and issues the forced external function to the device. The forced external function contains a Write command for the device.
2. The channel then executes the command chain. The TIC command is executed and the CCW address field is changed to  $54_{16}$ .
3. The channel continues the command chain and executes the Write command by activating the output data buffer.
4. Four words are transferred from the channel to the device. All the words are transferred from the same storage address because the data address lock flag is set.
5. The channel executes a data chain and transfers three words from storage addresses  $29_{16}$ ,  $28_{16}$ , and  $27_{16}$  because the data address decrement flag is set.
6. The channel executes a command chain and issues a forced external function to the device. The forced external function contains a Read command for the device.
7. The channel executes the Read command by activating the input buffer.
8. Two words are transferred from the device to the channel but not to storage because the skip data flag is set.
9. The channel terminates the operation and presents an interrupt request to the processor when the interrupt mechanism is free.
10. The interrupt is acknowledged and a CSW is written with the next CCW address field equal to  $62_{16}$ , the data count field equal to zero, and the Monitor and PCI subchannel status bits set.

The interrupt mechanism was assumed to be busy during the execution of the CCW list causing the PCIs to be overlaid. The device was assumed to be non-interrupting. On an ISI interface an external interrupt immediately terminates the execution of a CCW list.

35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command Code												Data Address																							

71	70	69	68	67	66	65	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36		
Word Count																																					

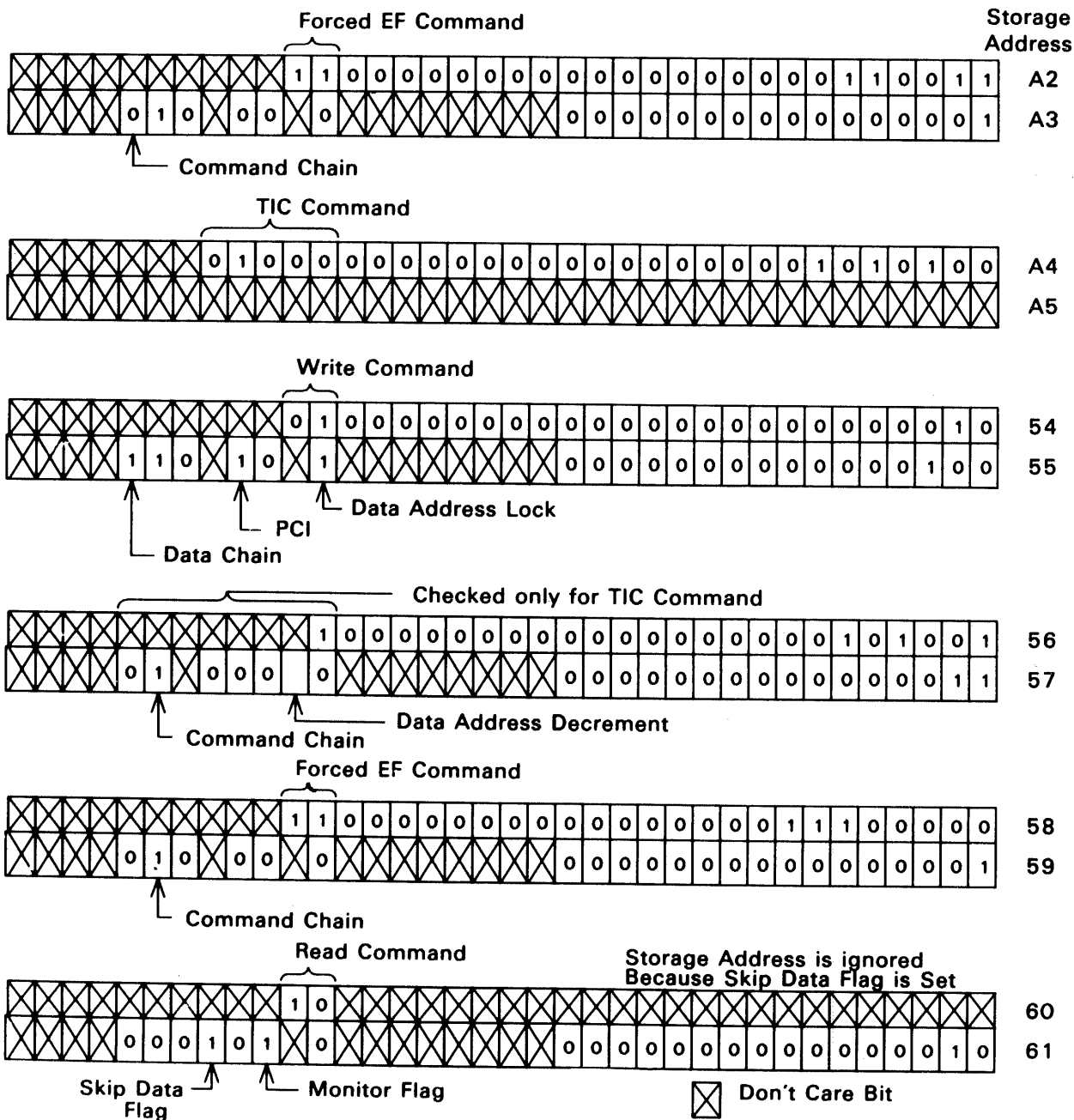


Figure 6-4. Word Channel ISI Interface Example CCW List

## 7. Interrupts

### 7.1. INTRODUCTION

An interrupt causes the current instruction sequence to be suspended and an instruction sequence starting at a fixed storage location to be initiated; the fixed address replaces the value in the program address register. The fixed storage address is associated with the event or condition that caused the interrupt to be generated, and thereby allows switching to a program to respond to that condition or event. Excepting those instructions that are explicitly named as interruptible, such as repeated instructions like BLOCK TRANSFER, the Processor honors interrupts only after the current instruction is completed and only if the interrupt to be honored is allowed. See Table 7-1 for interrupt priorities. The following interrupts are always allowed:

- All program exception interrupts, including Guard Mode and Addressing Exception interrupts.
- All arithmetic exception interrupts, including Characteristic Overflow, Characteristic Underflow, and Divide Check interrupts.
- Certain program initiated interrupts, including Executive Request, Test and Set, Quantum Timer, Breakpoint, and Emulation interrupts.
- Storage Check interrupts caused as the result of transfers between the processor and SIU.

Certain interrupts are disallowed between the execution of a Prevent All Interrupts and Jump (PAIJ) instruction or the occurrence of an interrupt, and the execution of an Allow All Interrupts and Jump (AAIJ) instruction or User Return (UR) or Load Designator Register (LD) instruction that sets D3. These include the following:

- All I/O interrupts, including those for Normal status, Tabled status, and Machine Check interrupts.\*
- Jump History Stack interrupts.
- Interprocessor interrupts.\*
- Dayclock and Real-Time Clock interrupts.
- Storage Check interrupts caused as the result of transfers from the SIU to storage units.
- Power Check interrupts.\*

\* Note that if interrupts are locked out and the processor is stopped via HJ, interprocessor interrupts and Power Check restoration interrupts are allowed; and if the processor is stopped in the cleared state, in addition to interprocessor interrupts, I/O normal status interrupts are allowed if the processor has been selected for initial load.

Table 7-1. Interrupt Priority

Priority	Interrupt Type
0	Immediate Storage Check (Oper Port) Immediate Storage Check (inst port)
1	Guard Mode (oper port) Guard Mode (inst port)
2	Addressing Exception Invalid Instruction Executive Request Test and Set Characteristic Overflow Characteristic Underflow Divide Check
3	Emulation
4	Breakpoint
5	Quantum Timer
6	Jump History Stack
7	Power Restored Power Loss
8	Real Time Clock
9	Dayclock
10	Delayed Storage Check Lower Delayed Storage Check Upper
11	IOU 0 Machine Check
12	IOU 1 Machine Check
13	IOU 0 Normal Status
14	IOU 1 Normal Status
15	IOU 0 Tabled Status
16	IOU 1 Tabled Status
17	IPI P
18	IPI P+1/P-3
19	IPI P+2/P-2
20	IPI P+3/P-1

**NOTE:**

Priority levels 0 through 5 are internal interrupts, which can be neither locked out nor deferred (always allowed).

Priority levels 6 through 20 are external interrupts, which can be both locked out and deferred.

P is the processor number for Interprocessor Interrupts.



## 7.2. INTERRUPT SEQUENCE

When the central processor unit (CPU) honors an interrupt request, the following events occur:

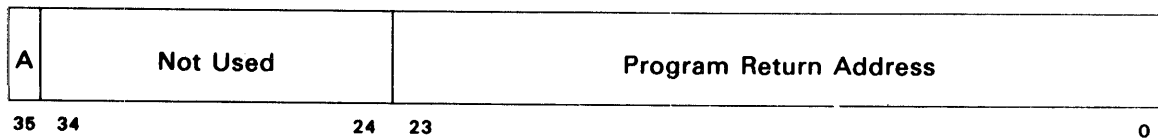
- The processor state is stored in GRS in three groupings: program status (041–044, 050–052), addressing status (040, 045–047), and interrupt status (053–055).
- All designator register bits are set to zero except D6 and D7, which are set to one, and D12, which is not altered.
- External interrupts are prevented from occurring until allowed by an AAIJ or UR or LD instruction.
- Control is transferred to the associated interrupt location. Note that this instruction must be unconditional jump.

### 7.2.1. Program Status

Program status is stored for all interrupts, and includes the following information:

- Program Return Address
  - GRS Location 043 for Normal interrupts
  - GRS Location 051 for Guard Modes
  - GRS Location 041 for Immediate Storage Checks
- Quantum Timer Value
  - GRS location 050 for all interrupts.
- Designator Register Value
  - GRS Location 044 for Normal interrupts
  - GRS Location 052 for Guard Modes
  - GRS Location 042 for Immediate Storage Checks

A program return address is the address of the instruction following the last instruction that was fully executed; program control would normally be returned at this point for recoverable errors. The program return address stored in GRS location 041 or 043 is in the following format:



The program return address value will vary depending on the operation being performed at the time of interrupt:

- If an incomplete Block Transfer, search instruction or byte instruction is interrupted, the return address will be P.
- If a satisfied skip instruction is interrupted, the return address will be P+2.
- If a satisfied jump instruction is interrupted, the return address will be U.

- If an instruction other than the above is interrupted, the return address will be P+1.

The contents of the program address register are changed only by a jump instruction (including User Return) or interrupt. Instruction references following a jump instruction are made under the same addressing constraints that conditioned the operand address of the jump instruction that began the straight line instruction stream.

Bit position 35 of the first word of the two-word packet contains a flag that identifies the correct program addressing mode. The two modes of program address generation include absolute (A=1), corresponding to D35=0 and D7=i=1, and relative (A=0), corresponding to D7=0 or i=0.

Bit positions 18 through 23 of the relative program address are zero unless absolute 24-bit indexing mode is selected. For straight line instruction sequencing, the relative program address is increased by one for each instruction that is executed or skipped. This increase is accomplished by twos complement addition with wraparound at 18 or 24 bits, depending on the value of the A flag.

### 7.2.2. Addressing Status

Addressing status is not actually stored during the interrupt sequence. The information within this group is placed in GRS by the software either directly (load, store) or indirectly (LBJ, LAE) and is used from GRS by the processor for addressing operations. Addressing status includes the following information:

- Executive bank descriptor table pointer.
- User bank descriptor table pointer.
- Bank descriptor specifications in the following format:

E0	0-0	0-0	BDI 0				E2	1	0	0-0	BDI 2				
E1	0	1	0-0	BDI 1				E3	1	1	0-0	BDI 3			
35	34	33	32	30	29	18	17	16	15	14	12	11	0		

### 7.2.3. Interrupt Status

Interrupt status is information associated with a particular type of interrupt, and is stored only when its type of interrupt occurs. Immediate Storage Check status is stored in GRS location 054, Guard Mode status is stored in GRS location 053, all other processor generated interrupt status is stored in the Normal status location, GRS 055. Interrupt status is associated with the following types of interrupts:

- Immediate Storage Check interrupts
- Guard Mode interrupt

- Executive Request, Test and Set, and Invalid Instruction interrupts
- Delayed Storage Check interrupts
- Breakpoint and Emulation interrupts
- Power Check interrupt
- Addressing Exception interrupt
- Interprocessor interrupt

### 7.3. INTERRUPT TYPES

The processor provides twenty interrupt priorities. The interrupt types are shown in Table 7-1.

#### 7.3.1. Program Exception Interrupts

**Invalid Instruction** – This interrupt occurs when the processor attempts to execute an instruction with an invalid function code. The operand address of the instruction (24 bits of U) is stored in GRS as interrupt status.

**Guard Mode** – This interrupt occurs in the following cases:

- When D2 equals one, and the execution of a privileged instruction is attempted.
- When violating the storage limits when designator register bit D7 is zero.
- When any reference is made to an SIU that has its interface to the processor disabled.
- When attempting to store in GRS locations other than those allowed for the user (40<sub>6</sub> through 100<sub>6</sub> and 120<sub>6</sub> through 177<sub>6</sub> when the designator register bit 2 (D2) is one.
- When attempting to write into a storage area specified by bank descriptor register (BDRO, 1, 2, or 3) for which the corresponding Write Protect designator bit (D13 through D16) is one.

See Figure 7-1 for the format of the Guard Mode interrupt status stored in GRS during the interrupt sequence.

**Addressing Exception** – This interrupt occurs in the following cases:

- E bit violation – If the E bit (bit 35) from Xa of an LBJ, LIJ or LDJ instruction is one and D19 is zero.
- Table length violation – If a bank descriptor index value from Xa of an LBJ, LIJ or LDJ instruction is greater than the selected BDT pointer length value.
- Residency interrupt – If the R flag of the new bank descriptor is one.
- Entry point violation – If the V flag of the new bank descriptor is one.
- Use count overflow on LBJ, LIJ, or LDJ new bank descriptor.

- Use count underflow on LBJ, LIJ, or LDJ old bank descriptor.
- Use count decreased to zero and C flag was one.

See Figure 7-2 for the format of the Addressing Exception interrupt stored in GRS during the interrupt sequence. The program return address stored for this interrupt is  $P + 1$  for E bit or table length violations, and the jump to address for all other violations.

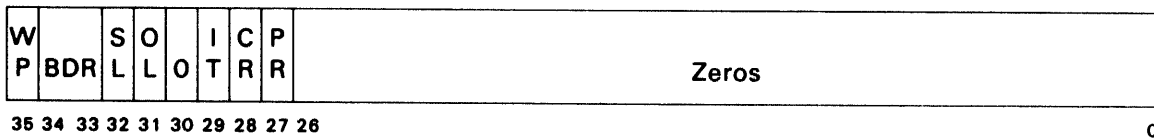
### 7.3.2. Arithmetic Exception Interrupts

An interrupt occurs in the following cases only if designator register bit 20 (D20) is one.

**Characteristic Overflow** – Occurs when the exponent value of a floating-point result is greater than  $+127_{10}$  (single precision) or  $+1023_{10}$  (double precision). When this condition is detected, D22 is set to one.

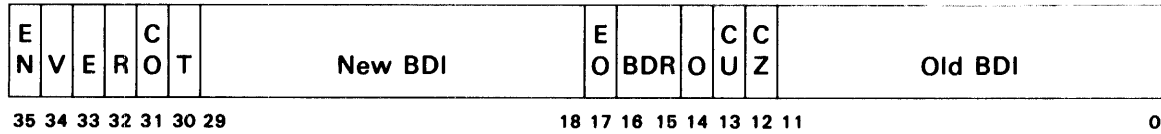
**Characteristic Underflow** – Occurs when the exponent value of a floating-point result is less than  $-128_{10}$  (single precision) or  $-1024_{10}$  (double precision). When this condition is detected, D21 is set to one.

**Divide Check** – Occurs when the magnitude of the quotient exceeds the range of the specified register. When this condition is detected, D23 is set to one.



- Bit 35            Write protection violation.
- Bits 34-33      BDR number associated with write protection violation.
- Bit 32            Storage limits violation.
- Bit 31            Reference to disabled storage.
- Bit 30            Is zero.
- Bit 29            Interrupt lockout exceeded.
- Bit 28            Control register violation.
- Bit 27            Privileged instruction violation
- Bit 26-0        Are zeros.

Figure 7-1. Format of Guard Mode Interrupt Status



- Bit 35            New bank descriptor E flag specification from Xa.
- Bit 34            The V flag indicates an entry point violation on the new bank descriptor.
- Bit 33            The E flag indicates an E bit violation on the new bank descriptor.
- Bit 32            The R flag indicates the residency flag of the new bank descriptor was one.
- Bit 31            The CO flag indicates a use count overflow on the new bank descriptor.
- Bit 30            The T flag indicates a table length violation on the new bank descriptor.
- Bits 29–18        New bank descriptor BDI specification from Xa.
- Bit 17            Old bank descriptor E flag specification from GRS.
- Bits 16–15        The bank descriptor register specification from Xa.
- Bit 14            Is zero.
- Bit 13            The CU flag indicates a use count underflow on the old bank descriptor.
- Bit 12            The CZ flag indicates the old bank descriptor use count was decreased from one to zero and the C flag was one.
- Bits 11–0        Old bank descriptor BDI specification from GRS.

**NOTE:**

This interrupt results only from the execution of an LBJ, LIJ or LDJ instruction. The new bank descriptor specifications are contained in Xa before execution; the old bank descriptor specifications are contained in GRS locations 046 and 047, and are placed in Xa during execution.

*Figure 7-2. Format of Addressing Exception Interrupt Status*

**7.3.3. Program-Initiated Interrupts**

**Executive Request** – This interrupt occurs as a result of executing an Executive Request (ER) instruction. This instruction allows a worker program to release control of the processor to the Executive System. The operand address of the instruction (24 bits of U) is stored in GRS (0222) as interrupt status.

**Test and Set** – This interrupt occurs as a result of executing a Test and Set (TS) instruction if bit 30 of the operand is one, or as a result of executing a Test and Set Alternate (TSA) instruction. The operand address of the instruction (24 bits of U) is stored in GRS (0224) as interrupt status.



### 7.3.5. Clock Interrupts

**Quantum Timer** – A Quantum Timer interrupt occurs when the quantum timer value reaches zero.

**Real-Time Clock** – This interrupt occurs when the contents of the lower 18 bits of the real-time clock (RTC) register (GRS address 100<sub>g</sub>) is decreased through zero. The value contained in the RTC is decreased by one every 200 microseconds. The RTC oscillator is accurate to  $\pm 0.02$  percent.

**Dayclock** – This interrupt request is made to all processors in the system once every 6.5536 seconds. Only one processor may honor each request. The dayclock value is increased by one every 200 microseconds.

### 7.3.6. Storage Check Interrupts

Storage Check interrupts are caused by conditions that are divided into two groups: immediate interrupt conditions, which are related to the current processor storage reference, and delayed interrupt conditions, which may or may not be related to the current operation. Immediate interrupt conditions include check conditions that occur on transfers between the processor and SIU. They may terminate the instruction and cannot be prevented by an interrupt or PAIJ instruction. Delayed interrupt conditions include internal SIU checks and check conditions that occur on transfers from the SIU to main storage units. These conditions do not affect the current instruction and may be deferred by an interrupt or PAIJ instruction.

#### 7.3.6.1. Immediate Storage Checks

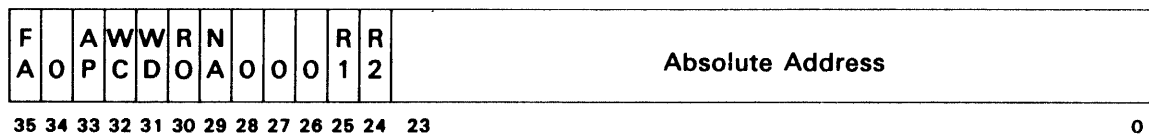
Immediate Storage Check interrupt status word is shown in Figure 7-5. An immediate check interrupt occurs:

- If the SIU detects a parity error on the address, controls, or write data from the processor.
- If the processor detects a parity error on the read data from the SIU.
- If the SIU detects an address that references a non-available storage location.

The Immediate Storage Check interrupt status word provides information to assist in performing software instruction retry (no retry is actually done by hardware). The retry information is provided only as a result of an Immediate Storage Check interrupt. On all Immediate Storage Checks the P value captured is P of the instruction having the error plus one, regardless of whether the error occurred on the instruction or on its operand(s). Two bits are used to define retry status, bit 25 defines whether the check occurred on an instruction fetch or an operand read/write. If bit 25 = 1, the check occurred on an instruction fetch and retry can be done by decrementing P by one and returning to that point without further analysis. If bit 25 = 0, the check occurred on an operand read/write and further analysis is required.

In general, bit 25 = 0 analysis must determine if the instruction on whose operand cycle the check occurred had the indirect bit or incrementation bit set. If indirection was specified retry will not be successful since one or more cycles of indirection may have occurred. If h was set, incrementation will have occurred, and will occur again if retry is attempted.

Bit 24 defines the state of execution at the point of Immediate check on an operand cycle. If bit 24 = 0, the initial values are intact (except as described above for indirect and increment) and retry can be attempted. If bit 24 = 1 execution will have proceeded to a state where retry is not possible.



- Bit 35** Indicates that the Immediate Storage Check occurred while fetching an instruction from any interrupt entrance fixed address.
- Bit 34** Is zero.
- Bit 33** Indicates an address parity check.
- Bit 32** Indicates a control parity check.
- Bit 31** Indicates a write data parity check.
- Bit 30** Indicates a parity check on read data received by the processor.
- Bit 29** Specifies that the addressed storage is not available.
- Bit 28–26** Are zero.
- Bit 25 (Retry 1)** = 0 Immediate check on operand.  
= 1 Immediate check on instruction.
- Bit 24 (Retry 2)** = 0 Retry possible on operand check.  
= 1 Retry not possible
- Bit 23–0 \*** Absolute address associated with check condition.
- \* When Bit 35 is a 1, the absolute address does not contain the MSR value, but is the fixed address relative to MSR.

*Figure 7-5. Format of Immediate Storage Check Interrupt Status*

### 7.3.6.2. Delayed Storage Check Interrupts

Delayed Storage Check interrupts occur when errors are detected either within the SIU or during data transfer between the SIU and MSU. This error condition is reported to the processor by a Storage Check Interrupt Status Request and a status word.

#### 7.3.6.2.1. SIU/MSU Errors and Internal SIU Errors

Any time an error occurs in a transfer between the SIU and an MSU, a Storage Check interrupt status request will be sent to the processors. The first error condition is retained in a Storage Check interrupt status format word (Figure 7-6) until read at which time the next condition will be loaded into the format word. The handling of errors is as follows:



### SIU/MSU Errors

- SIU to MSU address parity error (read cycle) – MSU read data is not loaded and a "not available" is sent to the requestor.
- SIU to MSU write data parity error – The MSU will write even if a write data parity error has occurred.
- SIU to MSU address parity error (write cycle) – The MSU will abort the write cycle, and abort the request which forced the write.
- MSU to SIU read data error (detected at the SIU) – The data is loaded in error into the buffer segments.
- Request to MSU that is not available – The requester's cycle is aborted and a "not available" is sent to that requester.
- MSU data check (corrected data) – Corrected data has been sent to the SIU.
- MSU data check (uncorrected data) – Uncorrected data has been sent from the MSU to SIU.

### SIU Internal Errors

- SIU control is unable to match the block address in both segments. – The requester's cycle is aborted and a "not available" is sent to that requester. A Storage Check interrupt status request is sent to the processors. The absolute address in the status format (Figure 7-6) contains the least recently used (LRU) block-and-set address of the buffer segment which caused the miss.
- Duplicate age detected – This error condition is displayed on the SIU maintenance panel and a Storage Check interrupt status request is sent to the processor. This error condition could result in the SIU control being unable to match the block address in both segments.
- Parity error in the block-set-writeback-validity bits during a match reference to the buffer access section. – This error results in a Storage Check interrupt status request being sent to the processor and a Storage Check interrupt status format word. An MSU writeback will be aborted if the error occurred during an SIU control read.

Request attempts after detection of an SIU internal error will not be honored until the affected SIU half has been cleared via a check reset, clear with initialize, or any clear from the exerciser. Before clearing the SIU half, the manual read tag buffer capability of the exerciser should be utilized.

Fail-Soft requirements can be met by utilizing the second SIU half via an auto recovery or manual reboot.

When an address or data parity error occurs between the SIU and MSU, the SIU will perform a retry. The Storage Check interrupt status request is sent to the processor and the Storage Check interrupt status word is loaded in the SIU register after the retry.

#### **7.3.6.2.2. Storage Check Interrupt Status**

Storage Check interrupt status is used to report SIU to MSU parity checks, MSU to SIU parity checks, offline or not installed MSUs, retry conditions on the SIU/MSU interface, and internal SIU errors. Figure 7-6 shows the format of the Storage Check interrupt status word as presented to the processor by the SIU.

The error conditions which affect the Storage Check interrupt status are as follows:

■ **Unsuccessful Retry**

A check condition was detected during the retry operation. Bit 35 in the Storage Check interrupt status is set to the active state.

■ **Parity Check in Tag Buffer**

A writeback operation (if needed) will not be initiated. A retry operation is not initiated. Bit 34 in the Storage Check interrupt status is set to the active state.

■ **Duplicate Age Detected**

A retry operation is not initiated. Bit 33 in the Storage Check interrupt status is set to the active state.

■ **SIU Unable to Match LRU Address in Both Segments**

An Address Not Available is sent to the requester. A retry operation is not initiated. Bit 32 in the Storage Check interrupt status is set to the active state.

■ **Request to MSU that is Not Available**

An Address Not Available is sent to the requester. A retry operation is not initiated. Bit 31 in the Storage Check interrupt status is set to the active state.

■ **SIU to MSU Write Address Check**

A retry operation is initiated. Bit 29 in the Storage Check interrupt status is set to the active state.

■ **SIU to MSU Read Address Check**

A retry operation is initiated. An Address Not Available is sent to the requester if the retry fails. Bit 28 in the Storage Check interrupt status is set to the active state.

■ **MSU to SIU Read Data Check (Detected at SIU Interface)**

A retry operation is initiated. The data is loaded parity incorrect into the SIU Buffer. If the parity check pertains to the word being addressed, the requester will detect a parity check at his interface. Bit 27 in the Storage Check interrupt status is set to the active state.

■ **MSU to SIU Connection interrupt**

A retry operation is not initiated. Bit 26 in the Storage Check interrupt status is set to the active state.

■ **MSU to SIU Multiple Uncorrected Error**

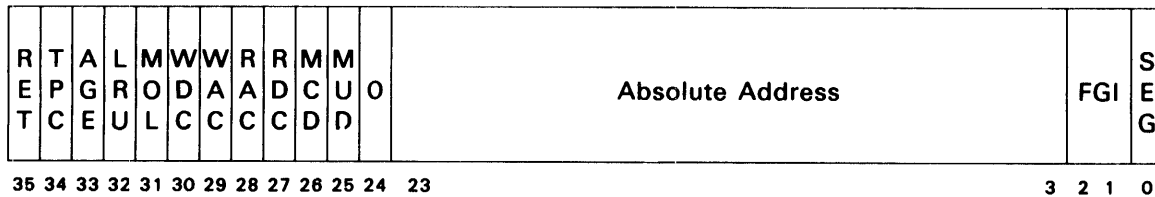
A retry operation is initiated. The data is loaded into the SIU data buffer as it was received from the MSU. Bit 25 in the Storage Check interrupt status is set to the active state.

More than one status bit can be set for one operation; i.e., the retry bit will augment original status if the retry was unsuccessful.

Only one Storage Check interrupt status request (SCISR) is sent to the processors for each requester-initiated operation. The SCISR is not raised until after the retry operation has been attempted.

A retry operation is initiated for all check conditions detected on the interface between the SIU and MSU if the addressed MSU is available.

The Storage Check interrupt status is not cleared until a processor acknowledges the request. Additional storage check status conditions are lost if they are detected after the Storage Check interrupt status request is made and before the SIU receives the acknowledge from a processor.



Bit

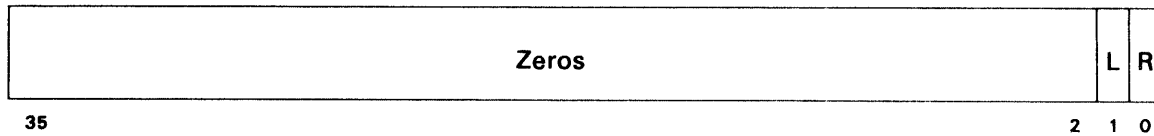
- 35 RET Unsuccessful Retry
- 34 TPC Parity Check in Tag Buffer
- 33 AGE Duplicate Age Detected
- 32 LRU SIU Unable to Match LRU Address in Both Segments
- 31 MOL Request to MSU that is Not Available
- 30 WDC SIU to MSU Write Data Check
- 29 WAC SIU to MSU Write Address Check
- 28 RAC SIU to MSU Read Address Check
- 27 RDC MSU to SIU Read Data Check
- 26 MCD MSU
- 25 MUD MSU
- 0 SEG Segment Designator for Internal SIU Checks (TPC, AGE, and LRU)
- Bits 1 and 2 MSU Failing Group Identification (MSU Read Data Only)
  - 00 First Double Word Check
  - 01 Second Double Word Check
  - 10 Third Double Word Check
  - 11 Fourth Double Word Check

Figure 7-6. Storage Check Interrupt Status Word

### 7.3.7. Power Check Interrupt

A Power Check occurs when a system or processor power check condition is detected due to a power service interruption or failure. Power Check interrupt status is stored in GRS during the interrupt sequence.

The format of the Power Check interrupt status word is shown in Figure 7-7 and provides for early warning power loss detection and power restoration indication.



Bits 35-2        Are zeros.

Bit 1            The L bit indicates that a power loss condition has been detected and processor power will drop after a 100ms grace period, unless within that period the power loss condition is removed (power restoration has occurred).

Bit 0            The R bit indicates that power has been restored following a power loss indication, but within the 100ms grace period. This interrupt condition is not locked out when the processor is stopped following a Power Check power loss interrupt.

**NOTE:**

If both L and R are one, it means that another power loss that had not yet been reported occurred after a recovery.

*Figure 7-7. Power Check Interrupt Status*

### 7.3.8. Byte status Code

A 7-bit status code is stored in the BB2 field of Staging Register 3 (SR3) either upon successful completion of the instruction or upon detection of an error condition which prevents completion during the execution of the following instructions: Byte to Binary Single Integer Convert (33, 10), Byte to Binary Double Integer Convert (33, 11), Byte to Single Floating Convert (33, 14), Byte to Double Floating Convert (33, 15), Byte Add (37, 06) and Byte Add Negative (37, 07). Table 7-2 contains a definition of the seven status bits. Table 7-3 shows a general input format which is acceptable to the byte-to-floating instructions 37, 14 and 37, 15.

Successful completion of an instruction results in the storing of an all-zero status word except for a decimal overflow for the byte add and add negative instructions (37, 06 and 37, 07) or a missing mantissa field for the byte-to-floating instructions (33, 14, and 33, 15).

Table 7-2. Byte Status Code Definition

**BB2 – Bit 0 – Format Error**

Set for instructions 33, 10 and 33, 11 if:	Byte not digit or blank (checked on all but last byte) or least significant four bits of last byte greater than nine.
Set for instructions 37, 06 and 37, 07 if:	Byte not digit (checked on all but first byte) or least significant four bits or first bytes (E and F) greater than nine.
Set for instructions 33, 14 and 33, 15 if:	<ol style="list-style-type: none"><li>Two signs in string not separated by at least one nonblank character.</li><li>Two decimal points in mantissa.</li><li>Significant character not found.</li><li>Illegal character in string.</li><li>Illegal character in exponent.</li><li>Decimal point last character and no digit in string.</li><li>Sign last character in string.</li></ol>

**BB2 – Bit 1 – Underflow**

Set for instruction 33, 14 if:	Magnitude of input too small to represent in single-precision floating point format.
Set for instruction 33, 15 if:	Magnitude of input too small to represent in double-precision floating point format.
Set for instructions 33, 14 and 33, 15 if:	Exponent negative and power of ten too small to represent in double-precision floating point format.

**BB2 – Bit 2 – Overflow**

Set for instruction 33, 10 if:	Magnitude of input too large to represent in 35 binary bits.
Set for instruction 33, 11 if:	Magnitude of input too large to represent in 71 binary bits.
Set for instructions 37, 06 and 36, 07 if:	Decimal add overflow.
Set for instruction 33, 14 if:	Magnitude of input too large to represent in single-precision floating point format.

*Table 7-2. Byte Status Code Definition (continued)*

<p>Set for instruction 33, 15 if:</p>	<p>Magnitude of input too large to represent in double-precision floating point format.</p>
<p>Set for instructions 33, 14 and 33, 15 if:</p>	<p>Mantissa interpreted as integer too large to represent in 60 binary bits.</p>
<p><b>BB2 – Bit 3 – Decimal Point Error</b></p>	
<p>Set for instructions 33, 14 and 33, 15 if:</p>	<p>a. Decimal point count greater than 31.</p> <p>b. Two decimal points in mantissa.</p> <p>c. Decimal point last character and no digit in string.</p>
<p><b>BB2 – Bit 4 – Not Significant Character Found</b></p>	
<p>Set for instructions 33, 14 and 33, 15 if:</p>	<p>a. Bits 0, 3, or 6 set and significant character not read yet.</p> <p>b. Mantissa field does not contain at least one digit (a blank following a decimal point is considered a digit).</p> <p>c. String does not contain at least one nonblank and nonsign character.</p>
<p><b>BB2 – Bit 5 – Exponent Found</b></p>	
<p>Set for instructions 33, 14 and 33, 15</p>	<p>Bits 0, 1, 2, 3, or 6 set and exponent field detected.</p>
<p><b>BB2 – Bit 6 – Mode Error</b></p>	
<p>Set for instructions 33, 10; 33, 11; 33, 14; and 33, 15 if:</p>	<p>Six or nine bit mode not selected (W bit) on one of the byte strings.</p>

*Table 7-3. General Input Format for Byte-to-Floating Instructions*

	Byte String					
Fields:	B	MS	M	ED	ES	E
Valid Characters:	ΔΔΔ...	±	Digit, Decimal Point, or ΔΔΔ...	DΔΔΔ... or EΔΔΔ...	±	Digits or ΔΔΔ...

*Table 7-3. General Input Format for Byte-to-Floating Instructions (continued)*

- B Leading blank ( $\Delta$ ) characters. Blanks in this field will be ignored.
- MS Mantissa sign: field may include one plus (+) or minus (-) character.
- M Mantissa: first digit or decimal point character indicates start of field. Blanks in this field will be interpreted as zeros.
- ED Exponent delineator: field may include either a D or E character followed by blanks. Blanks in this field will be ignored.
- ES Exponent sign: field may include one plus (+) or minus (-) character.
- E Exponent: field may include digits or blanks. Blanks in this field will be interpreted as zeros.

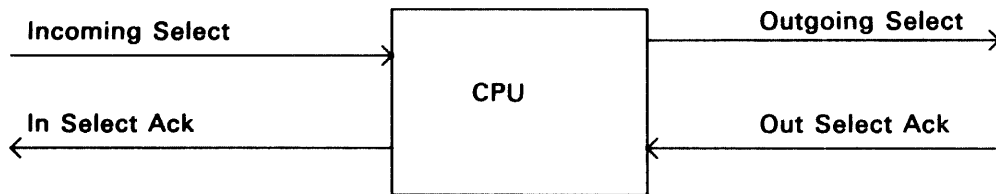
**NOTE:**

Any of the fields may be included or omitted in the input byte string subject only to the limitations listed below:

1. The valid characters indicated for each field are the only allowable characters.
2. The Mantissa Sign (MS) and the Exponent Sign (ES) must be separated by at least one nonblank character.
3. The last character in the string cannot be a sign character.
4. Overflow will occur if the number is too large to represent in single-precision format for the Byte to Single Floating Convert ( $j=33,14$ ) instruction or double-precision format for the Byte to Double Floating Convert ( $j=33,15$ ) instruction.
5. Underflow will occur if the number is too small to represent in single-precision format for the Byte to Single Floating Convert ( $j=33,14$ ) instruction or double-precision format for the Byte to Double Floating Convert ( $j=33,15$ ) instruction.
6. Underflow will occur if the exponent alone is too small to represent in double-precision floating-point format.
7. The mantissa must be representable in 60 binary bits when it is interpreted as an integer; i.e., ignoring the decimal point.
8. The decimal point count (number of digits or blanks to the right of the decimal point) must not be greater than 31.
9. Two decimal points in the mantissa will be detected as an error.
10. At least one nonblank and one nonsign character must be included in the string.
11. If the last character is a decimal point, it must be preceded by at least one nonblank and one nonsign character.

### 7.3.9. Multi-Processor-Interrupt Synchronization

All external interrupt requests; for example, those generated by IOUs, are presented to each processor in the System. Therefore, an interlocked synchronization mechanism is provided to assure that only one processor actually accepts the interrupt request. This interlock is provided as part of the interprocessor interrupt network and consists of four lines: one incoming select line, one incoming select acknowledge, one outgoing select line, and one outgoing select acknowledge:



The outgoing lines of one processor are connected to the incoming select lines of another processor, forming a ring. A processor is allowed to honor an interrupt request between the time the incoming select is acknowledged and the time the outgoing select is propagated. The time available for honoring interrupt requests shall be a function of processor design: sufficient to allow examination of requests but not so long that system interrupt response time is impaired. An acceptable value might range between 800 and 2000 nanoseconds per processor. If a decision is made by a processor to honor an interrupt request, the propagation of the outgoing select is delayed until the interrupt request is acknowledged and the interrupt request is dropped.

A processor will retain the interrupt interlock by not passing the interrupt select in a unit processor system; also in a multiprocessor system where the other processor is off line or powered down, and if, during initial load, that processor is selected for taking the initial load interrupt. The system transition unit will provide this information to each unit.

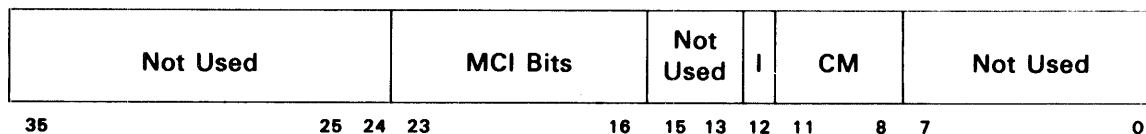
## 7.4. INPUT/OUTPUT INTERRUPTS

There are three classes of I/O interrupts: normal interrupts, tabled interrupts, and machine check interrupts. status conditions for the status table subchannel, all shared subchannels, and nonshared subchannels on a block multiplexer channel are reported by the normal interrupt mechanism. Status conditions for nonshared subchannels on a byte multiplexer channel or word channel are reported by the tabled interrupt mechanism. IOU or channel status conditions are reported by the machine check interrupt mechanism. IOU or channel status is any status not associated with a particular subchannel or device.

### 7.4.1. Machine Check Interrupts

If the control module or a channel detects status not associated with a particular device or subchannel, a Machine Check interrupt request is generated. When the Machine Check interrupt is acknowledged, an IAW is stored in the fixed IAW address of the processor that acknowledged the interrupt. Bits 00-07, 13-15, and 24-35 of the IAW are meaningless. Bits 08-12 of the IAW specify the IOU and channel number associated with the Machine Check interrupt. Bits 16-23 of the IAW are master bitted to indicate the condition or conditions that caused the Machine Check interrupt to be generated.



*Machine Check IAW*Machine Check Indicator Bits**CM** Bits 8–11 Channel module number**I** Bit 12 IOU number

**Bit 16** An interface control signal error or device address parity error prevented subchannel identification during a control unit initiated selection sequence. This bit is not used on a word channel.

**Bit 17** A storage error occurred when the channel attempted to read the second word of the CAW during a Load Channel Register operation.

**Bit 18** A storage error occurred when the channel attempted to write a channel status word for a Non-Tabled interrupt or an I/O instruction.

**Bit 19** A storage error occurred when the channel attempted to write a channel status word for a Tabled interrupt.

**Bit 20** A storage error occurred when the channel was attempting to write the preceding interrupt address word.

**Bit 21** A storage error occurred when the channel module attempted to write a control word for one of the 128 nonshared subchannels into storage.

**Bit 22** A storage error occurred when the IOU control module attempted to read the second word of the CAW during a Load Channel Register operation.

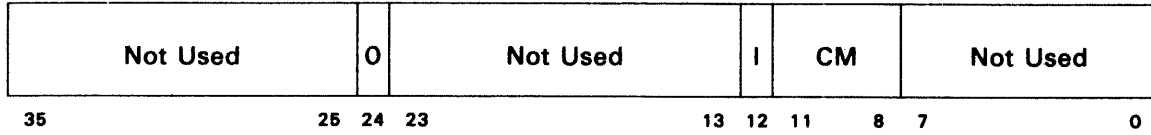
**Bit 23** A storage error occurred when the IOU control module attempted to read the first word of the CAW.

**7.4.2. Normal Interrupts**

If the status table subchannel, any shared subchannel, or a block multiplexer nonshared subchannel detects status conditions, a Normal interrupt request is generated. When the Normal interrupt is acknowledged, an IAW and CSW are stored in the fixed IAW and CSW addresses of the processor that acknowledged the interrupt. Bits 13–23 and 25–35 of the IAW are meaningless for a Normal interrupt. If bit 24 of the IAW is set, the interrupt is for the status table subchannel in the IOU and channel specified by bits 08–12 of the IAW. The device address field (bits 00–07) is not interpreted. The associated CSW contains status from the status table subchannel. If bit 24 of the IAW is not set, bits 00–12 contain the IOU and channel address associated with the interrupt. On a byte or block multiplexer channel, the device address field (bits 00–07) specifies the device. On a word channel, the device address field specifies only the subchannel. The associated CSW contains status for the device or subchannel specified by the device address field of the IAW.

Normal Interrupt IAW and CSW for a Byte or Block Multiplexer Channel

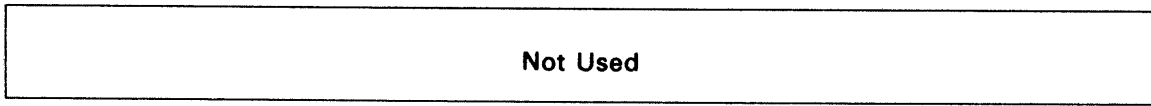
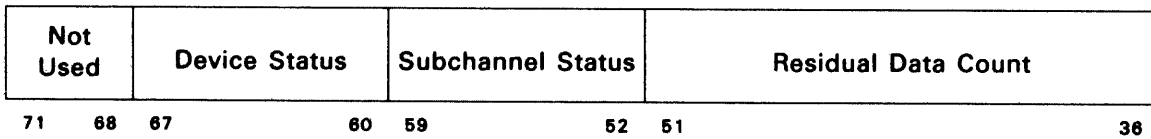
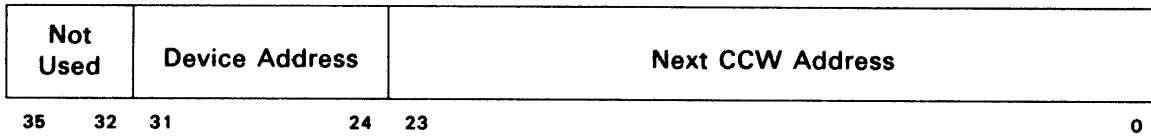
*IAW*



I IOU Number

CM Channel Module Number

*CSW*

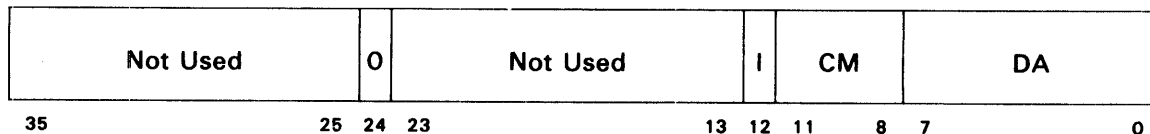


107

72

Normal Interrupt IAW and CSW for Word Channel

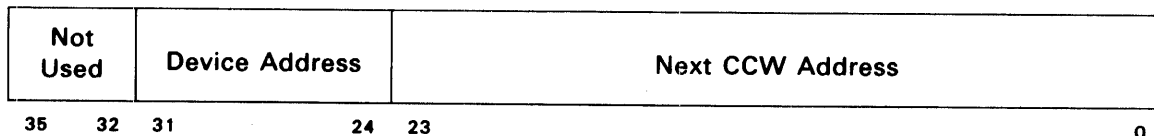
*IAW*



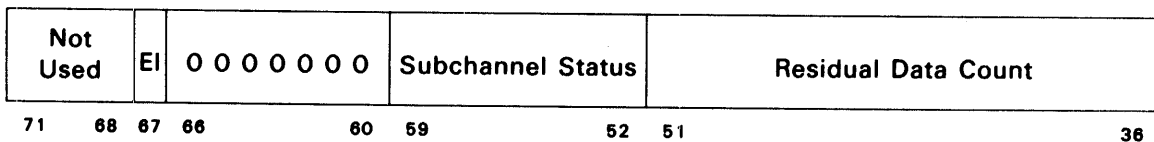
- I IOU Number
- CM Channel Module Number
- DA Device Address

ISI bits 4-7 contain the subchannel address and bits 0-3 are meaningless.  
ESI bits 0-7 contain the device address.

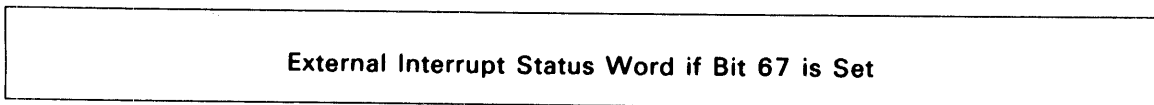
*CSW*



ISI bits 28-31 contain subchannel address and bits 24-27 are meaningless.  
ESI bits 24-31 contain device address.

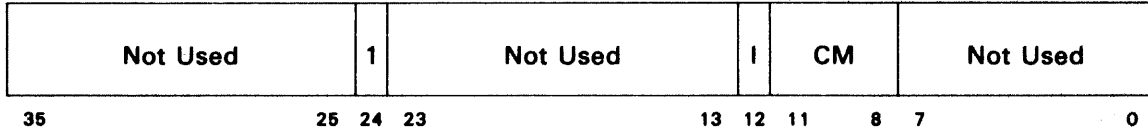


- EI 0 means bits 72 to 107 are meaningless
- 1 means bits 72 to 107 contain external interrupt status word



Normal Interrupt IAW and CSW for Status Table Subchannel

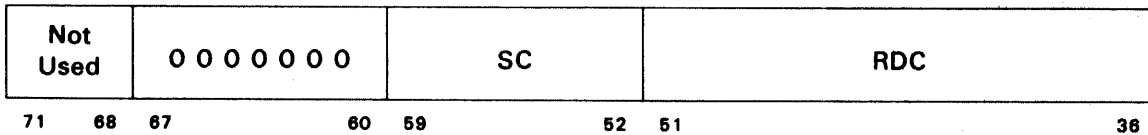
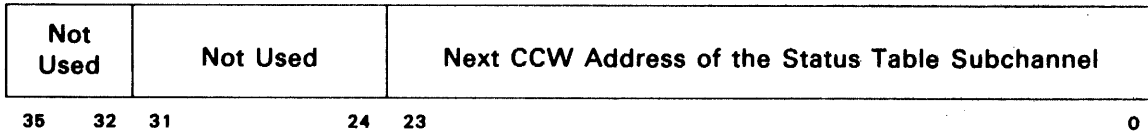
*IAW*



I IOU Number

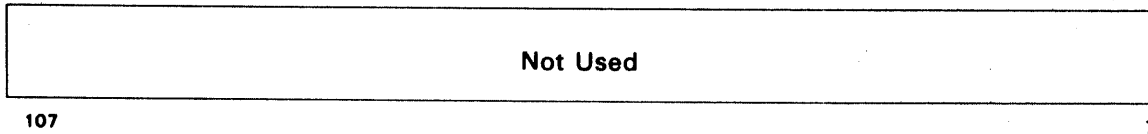
CM Channel Module Number

*CSW*



SC Subchannel status of the status table subchannel

RDC Residual data count of the status table subchannel



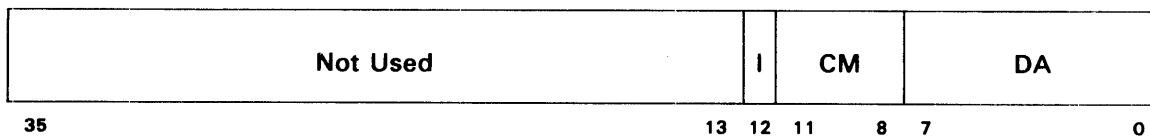
**7.4.3. Tabled Interrupts**

Status for a nonshared subchannel on a byte multiplexer channel or word channel (communications status) is stored in a status table under the control of the status table subchannel. There is one status table subchannel per channel. If the status table subchannel is not active when the nonshared subchannel status conditions are detected, the status is lost and the communications subchannel is returned to the available state. If the status table subchannel is active, a TSW containing the communications subchannel status is stored at the address specified by the status table and a Tabled interrupt request is generated. If another communications subchannel detects status conditions before the Tabled interrupt is acknowledged, its status is stored in a TSW at the address specified by the status table. The Tabled interrupt request is reset. Thus, a single Tabled interrupt request

may report several entries (TSWs) in the status table. When the Tabled interrupt is acknowledged, an IAW and CSW are stored in the fixed IAW and CSW addresses of the processor that acknowledged the interrupt. Bits 13-35 of the IAW are meaningless for a Tabled interrupt. Bits 08-12 specify the IOU and channel. Bits 00-07 specify the device address of the last entry in the status table. The associated CSW contains the status of the status table. The subchannel status field of the CSW is cleared.

Tabled interrupt IAW and CSW

*IAW*

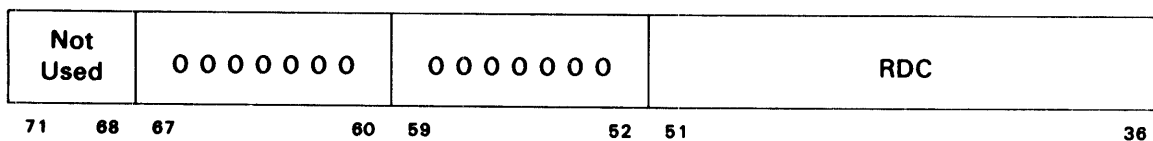
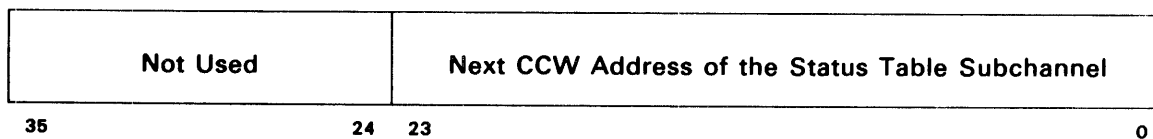


I IOU Number

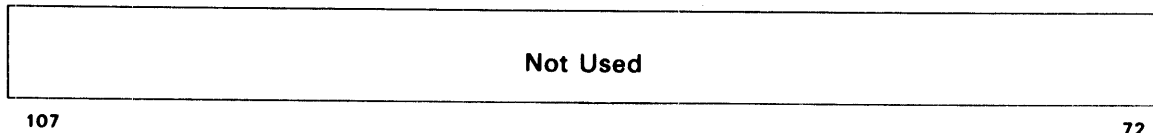
CM Channel Module Number

DA Device address of device that most recently made an entry in the status table.

*CSW*



RDC Residual data count of the status table subchannel





## 8. Executive Control

### 8.1. GENERAL

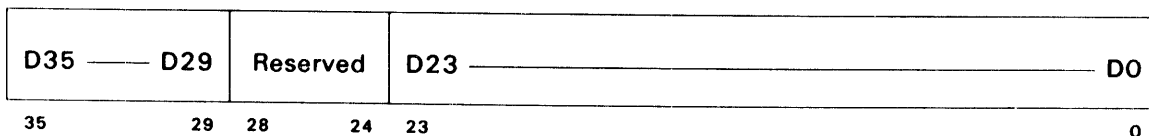
The 1100/80 Processor operates under the control of an Executive program which controls and coordinates the activities of the combined hardware and software systems and has exclusive use of certain control capabilities. By the use of bank descriptors, it can relocate any program in main storage. It provides storage protection through the use of storage limits registers. The bank descriptor specifications are contained in the general register stack (GRS). The bank descriptors are controlled by the Executive program for itself and for user programs. However, the user programs, through the LBJ, LIJ and LDJ instructions, are allowed to modify part of the designator register from a table prepared by the Executive program. The user programs are also allowed to modify several of the control bits by using the Load DR Designators instruction (LPD, see 5.13.1). The operations related to and affected by the contents of the bank descriptors are explained in this section.

### 8.2. PROCESSOR STATE

The processor state is defined as information contained in the processor and required to describe a program activity. This includes the bank descriptor information, the designator register, the relative program address and the general register stack (GRS). The processor state is automatically saved in GRS when an interrupt occurs. The designator register and relative program address, can also be stored by instruction. Each element of the program state can be loaded by instruction, and certain elements may be loaded in groups to facilitate the orderly sequencing of program control.

#### 8.2.1. Designator Register

The designator register contains information controlling functional characteristics of the processor. The designator register may be loaded by instruction, although certain bit combinations are not valid or may not be available to the user. In general, no hardware checks are made for these invalid combinations. When an interrupt occurs, the current value of the designator register is stored in GRS, and the register is cleared, unless otherwise specified in the following paragraphs, to establish the proper interrupt handling environment. The format of the designator register is:



D35– Must be zero  
D34

D33 Reserved

D32– Must be zero  
D30

D29 Quantum Timer Enable

When this designator is one, the quantum timer value is decreased by one for every one hundred nanosecond period that the processor is actually executing instructions. When the quantum timer value is zero, a Quantum Timer interrupt is generated. When D29 is zero, the quantum timer value is not altered.

D28– Reserved  
D24

D23 Divide Check Designator

This designator is set to one when the magnitude of the quotient exceeds the range of the specified register.

D22 Characteristic Overflow Designator

This designator is set to one when the characteristic of a floating-point result is greater than  $177_8$  (single-precision) or  $1777_8$  (double-precision).

D21 Characteristic Underflow Designator

This designator is set to one when the characteristic of a floating-point result is less than  $-200_8$  (single-precision) or  $-2000_8$  (double-precision).

D20 Arithmetic Exception Interrupt Designator

When this designator is zero, if an arithmetic exception occurs (D23, D22 or D21 set to one), the specified A-registers are cleared to zero and no interrupt occurs. When D20 is one, if an arithmetic exception occurs, the specified A-registers are left unchanged (except as specified by D5), and an interrupt occurs. When D20 is one, all instructions that can cause an arithmetic exception are executed without instruction overlap (i.e., completely executed prior to beginning the execution of the next instruction).

D19 EXEC Bank Descriptor Table Pointer Enable

When this designator is zero, only the user bank descriptor table (BDT) pointer may be selected during execution of the LBJ, LIJ or LDJ instructions. If an attempt is made to reference the EXEC bank descriptor table, an Addressing Exception interrupt is generated. When D19 is one, either the EXEC or user BDT pointer may be selected during execution of an LBJ, LIJ, or LDJ instruction.

D18 Reserved

D17 Enable residue store for single-precision floating-point instructions.



**D16**      **BDR3 Write Protection**

When this designator is one, a Guard Mode interrupt will occur if an attempt is made to write into the storage area specified by BDR3 and either D7 or the i-bit is zero.

**D15**      **BDR1 Write Protection**

When this designator is one, a Guard Mode interrupt will occur if an attempt is made to write into the storage area specified by BDR1 and either D7 or the i-bit is zero.

**D14**      **BDR2 Write Protection**

When this designator is one, a Guard Mode interrupt will occur if an attempt is made to write into the storage area specified by BDR2 and either D7 or the i-bit is zero.

**D13**      **BDRO Write Protection**

When this designator is one, a Guard Mode interrupt will occur if an attempt is made to write into the storage area specified by BDRO and either D7 or the i-bit is zero.

**D12**      **BDR Selector**

When this designator is one, BDR1 and BDR3 are selected as the primary pair of bank descriptor registers; when D12 is zero, BDRO and BDR2 are selected as the primary pair. The primary pair is selected over the secondary pair if the storage limits overlap. D12 will be toggled during the execution of a jump instruction if the jump operand address falls exclusively within the limits of the secondary pair. D12 is not altered when an interrupt occurs.

**D11**      **Reserved****D10**      **Quarter-Word Mode Designator**

When this designator is zero, for instructions with function codes less than  $70_8$  (not including 07, 33, and 37), the j-field values of 4, 5, 6, and 7 are interpreted as follows:

- j = 4**      Specifies half-word (18-bit) transfers to or from bits 35 through 18 of the operand.
- j = 5**      Specifies third-word (12-bit) transfers to or from bits 11 through 0 of the operand.
- j = 6**      Specifies third-word (12-bit) transfers to or from bits 23 through 12 of the operand.
- j = 7**      Specifies third-word (12-bit) transfers to or from bits 35 through 24 of the operand.

When D10 is one, for instructions with function codes less than  $70_8$  (not including 07, 33, and 37), the j-field values of 4, 5, 6, and 7 are interpreted as follows:

- j = 4**      Specifies quarter-word (9-bit) transfers to or from bits 26 through 18 of the operand.
- j = 5**      Specifies quarter-word (9-bit) transfers to or from bits 8 through 0 of the operand.

- $j = 6$  Specifies quarter-word (9-bit) transfers to or from bits 17 through 9 of the operand.
- $j = 7$  Specifies quarter-word (9-bit) transfers to or from bits 35 through 27 of the operand.

The value of D10 has no effect on an instruction in the following circumstances:

- When the f-field of the instruction contains a value in the range  $70_8$  through  $77_8$ , or 07, 33, or 37.
- When D4 is one.
- When the j-field contains a value other than 4, 5, 6, or 7.

D9 Reserved

D8 Floating-Point Zero Format Selection

This designator affects Floating Point Add, Floating Point Add Negative, Floating Point Multiply, Floating Point Divide, and Load and Convert to Floating (only single precision) instructions. When D8 is zero, and if the mantissa of the most significant word of a single-precision floating point result is  $\pm 0$ , the entire word is stored as all zero. When D8 is one, and if the mantissa of the most significant word of a single-precision floating-point result is  $\pm 0$ , the most significant word is packed and stored with the appropriate characteristic.

D7 Relocation and Storage Suppression

D7 controls 1100 mode index register length, relocatability, and limit violation checking. If  $D7=0$ , index registers are 18 bits long, relocation is performed through basing, and a limits violation will cause a Guard Mode interrupt. If  $D7=1$ , the same functions are dependent upon the i-bit of the instruction currently executing: If  $i=0$ , operation proceeds as if  $D7=0$ ; if  $i=1$ , index registers are 24 bits long, relocation is not performed (a base value is not added to the relative address), and relative addresses (which are not identical to absolute addresses) are not checked for limit violations.

Program addresses following a jump instruction are formed under the same D7 and i-bit conditions that were in effect for the jump instruction. That is, if an absolute jump occurred ( $D35=0$ ,  $D7=i=1$ ), subsequent instruction references will be absolute; if a relative jump occurred ( $D7$  or  $i=0$ ), subsequent instruction references will be relocated according to the current addressing control designators and BDR values. D7 is set to one on Master Clear or when an interrupt occurs.

D6 General Register (GRS) Selection Designator

When D6 is zero, the GRS addresses below are assigned for use by the user program and can be referenced by the a- and x-fields of the instruction.

Index (X) Registers	$0001_8 - 0017_8$
Accumulators (A-Registers)	$0014_8 - 0033_8$
Special (R) Registers	$0101_8 - 0117_8$



AN	(15)	Add Negative to A
AM	(16)	Add Magnitude to A
ANM	(17)	Add Negative Magnitude to A
AU	(20)	Add Upper
ANU	(21)	Add Negative Upper
AX	(24)	Add to X
ANX	(25)	Add Negative to X
DA	(71, 10)	Double-Precision Fixed Point Add
DAN	(71, 11)	Double-Precision Fixed Point Add Negative

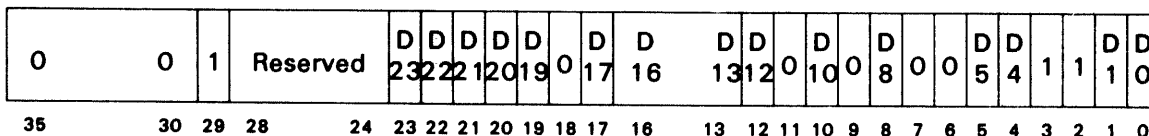
D1 is set to one if an overflow condition is detected during execution of any of the above instructions, and D0 is set to one if a carry condition is detected. When D0 or D1 are set, they remain so until another one of the above instructions is executed, or until the designator bits are directly altered by the program.

Figure 8-1 shows three basic conditions of the Designator Register.

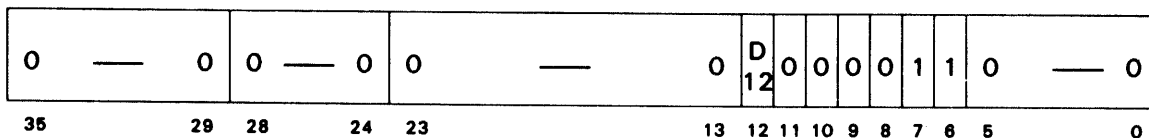
**NOTE:**

*D16-D13 are independent of D2.*

*User Mode*



*Interrupt Mode*



*Executive Mode*

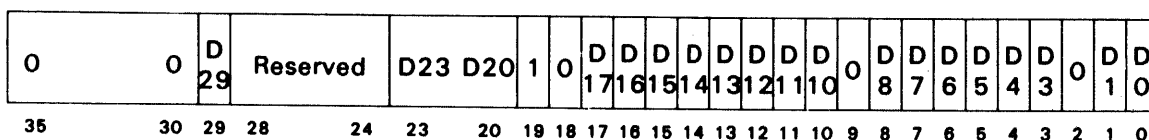


Figure 8-1. Basic Designator Register States

### 8.3. INTRODUCTION TO ADDRESSING

The CPU's hardware provides for relocating the instructions and/or data for any program in main storage. Also provided is the ability to specify that all area of main storage not assigned to a program are locked out to that program for read, write, and jump references. The main storage areas which may be assigned to a program are specified in 64-word granules beginning on any 512 word boundary and ending on any 64 word boundary.

#### 8.3.1. Main Storage Organization

The SPERRY UNIVAC 1100/80 System is designed as a modular system, permitting a variety of main storage configurations. The minimum main storage configuration comprises one 524K word basic module. The storage capacity can be expanded to eight MSUs in 262K word increments to a maximum of 4,194,304 words.

The two word bank descriptor register (BDR) provides the CPU with the flexibility for allocating storage for a program segment. The base value in conjunction with a relative address determines the absolute storage location. The upper and lower limits define the range of relative addresses within a program segment with the upper limit specified in 64 word increments and the lower limit specified in 512 word increments.

#### 8.3.2. Program Segmentation

A program may be written in segments which may be relocated in main storage. When the program is loaded into main storage, the executive program determines the number of 512 word granules and assigns them in contiguous blocks. Any unfilled portion of a granule is unavailable to another program if it is to be run with guard mode/storage limits protection.

#### 8.3.3. General Theory of 1100/80 Addressing

Normal 1100/80 programs are constructed without consideration for the physical area of storage they will occupy during execution. As the program is constructed, each address is mapped into a set of addresses called relative addresses. A relative address is actually used in an instruction within the program which references other locations or words in the program. Proper conversion from these relative addresses to the physical locations of the program will occur during execution using the bank descriptor register mechanism of the 1100/80. The range of relative addresses is from 0 to 262,143.

Relative address (U) is composed of the sum of the u-field of the instruction, the modifier field of the Xx-register, and in certain cases, the word offset (Ow) field of the Jj-register. A negative zero (all ones) cannot be generated as a relative address. For shift instructions, I/O instructions, or immediate operands ( $j = 16$  or  $17$ ), the relative address is generally used directly as an operand. If a relative address is less than 0200, it is generally used to reference GRS. If the relative address is greater than 0200, it is converted to an absolute address and used to reference storage.

An absolute address is normally composed of the sum of a relative address and a base value selected from one of the four available bank descriptor registers. It is also possible to have U generated as an absolute address and used directly to reference storage without being altered by addition of a base.

### 8.3.4. Bank Descriptor

A bank descriptor (BD) is a two word set of data defining storage allocation for a program segment. Bank descriptors are held in a bank descriptor table (BDT) which is located and defined by the bank descriptor table pointer (BDTP). The table address in the BDTP is the absolute address of the first word of the first BD in the BDT. The table length in the BDTP is in units of descriptors, not words. The BDs within the BDT are located by adding a bank descriptor index (BDI) to the table address in the BDTP. The BDI is also in units of descriptors. A table length value of zero defines a BDT containing only one BD. The BDT has a maximum length of 4K BDs (8K words). The BDT is expected to reside in storage (i.e., not in GRS). The BDTP and BD formats are shown in Figure 8-2.

### 8.3.5. Limits

The upper and lower limits define the range of relative addresses within a program segment. The check of a relative address against limits is inclusive, i.e., a relative address is within limits if it is greater than, or equal to, the lower limit; or less than, or equal to, the upper limit. The lower limit is in increments of 512 words, the upper limit is in increments of 64 words. Therefore, a program may contain any multiple of 64 words, beginning on any 512 word boundary and ending on any 64 word boundary.

### 8.3.6. Control Information

The flag and use count fields of the BD provide control pertaining to the relative space (segment) defined by the BD and a count of the activity or usage of this BD.

The B-flag indicates that an addressing exception interrupt is to occur if a reference is made to the segment through the new bank descriptor of an LIJ, LDJ, or LBJ instruction.

The W-flag indicates that a Guard Mode interrupt indicating write protection violation is to occur if a write is attempted into the segment.

The P-flag value is transferred to designator register bit 2 (D2 - privileged instruction and GRS protection designator) during the execution of an LBJ, LIJ, or LDJ instruction.

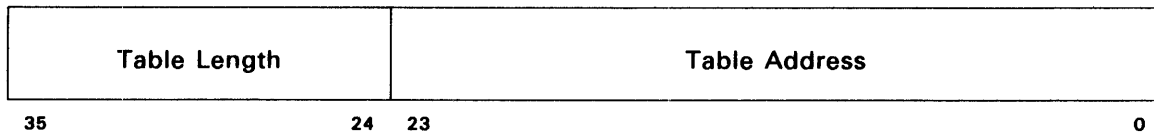
The V-flag indicates that entry point validation must be performed. This is accomplished by assuring that the relative operand address of the LBJ, LIJ, or LDJ instruction (jump address) that references the bank descriptor is equal to the bank descriptor lower limit value extended with low-order zeros (bits 8 through 0). This relative operand address must also select the BDR that is being loaded. If these conditions are not met and V is one, an addressing exception interrupt will occur. If the relative operand address is not within any limits a Guard Mode interrupt will occur.

The C-flag indicates that an interrupt is to occur if the use count is decreased to zero.

### 8.3.7. Bank Descriptor Registers

A bank descriptor register (BDR) contains the upper limit, lower limit, and base of a bank descriptor; these allow relative address limits checking for protection, base selection, and absolute address formation. Four bank descriptor registers; BDR0, BDR1, BDR2, and BDR3; are provided in the processor. The BDRs are loaded by the LAE, LL, or LB instructions. These instructions do not test the flags or change the use count.

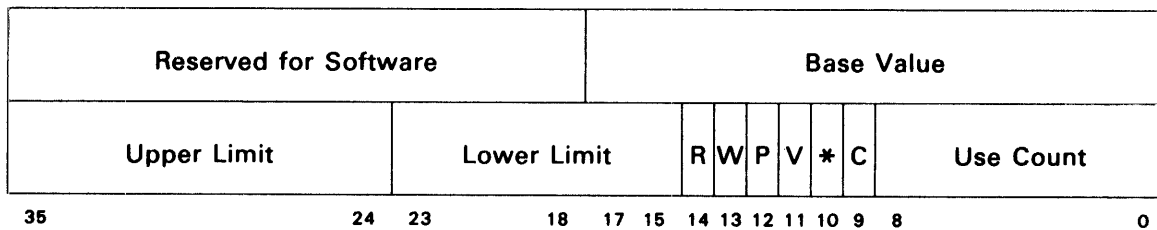
*Bank Descriptor Table Pointer Format*



Bits 35-24    Bank Descriptor Table Length (in Descriptors)

Bits 23-0    Bank Descriptor Table Address (Absolute)

*Bank Descriptor Format*



**First Word:**    Bits 35-18    Reserved for software  
                      Bits 17-0    Base value for relocation

**Second Word:**    Bits 35-24    Storage protection upper limit value  
                          Bits 23-15    Storage protection lower limit value  
                          Bit 14    Residency flag  
                          Bit 13    Write protection flag  
                          Bit 12    Privileged protection flag  
                          Bit 11    Validate entry point flag  
                          Bit 10    Reserved for software  
                          Bit 9    Use count interrupt on zero flag  
                          Bits 8- 0    Use count value

*Figure 8-2. Bank Descriptor and BDT Pointer Formats*

### 8.3.8. Address Generation

If base suppress conditions exist, the relative address is used as the absolute address, otherwise an absolute address is generated. To generate an absolute address, a relative address is added to a base value selected from those available in the four bank descriptor registers. To select which of the base values to use, a limits check is made between the relative address and the upper and lower limits. Designator D12 determines the order of BDR use as follows:

<u>Selector</u>	<u>Use (in order of preference)</u>
D12 = 0	BDR0, BDR2, BDR1, BDR3
D12 = 1	BDR1, BDR3, BDR0, BDR2

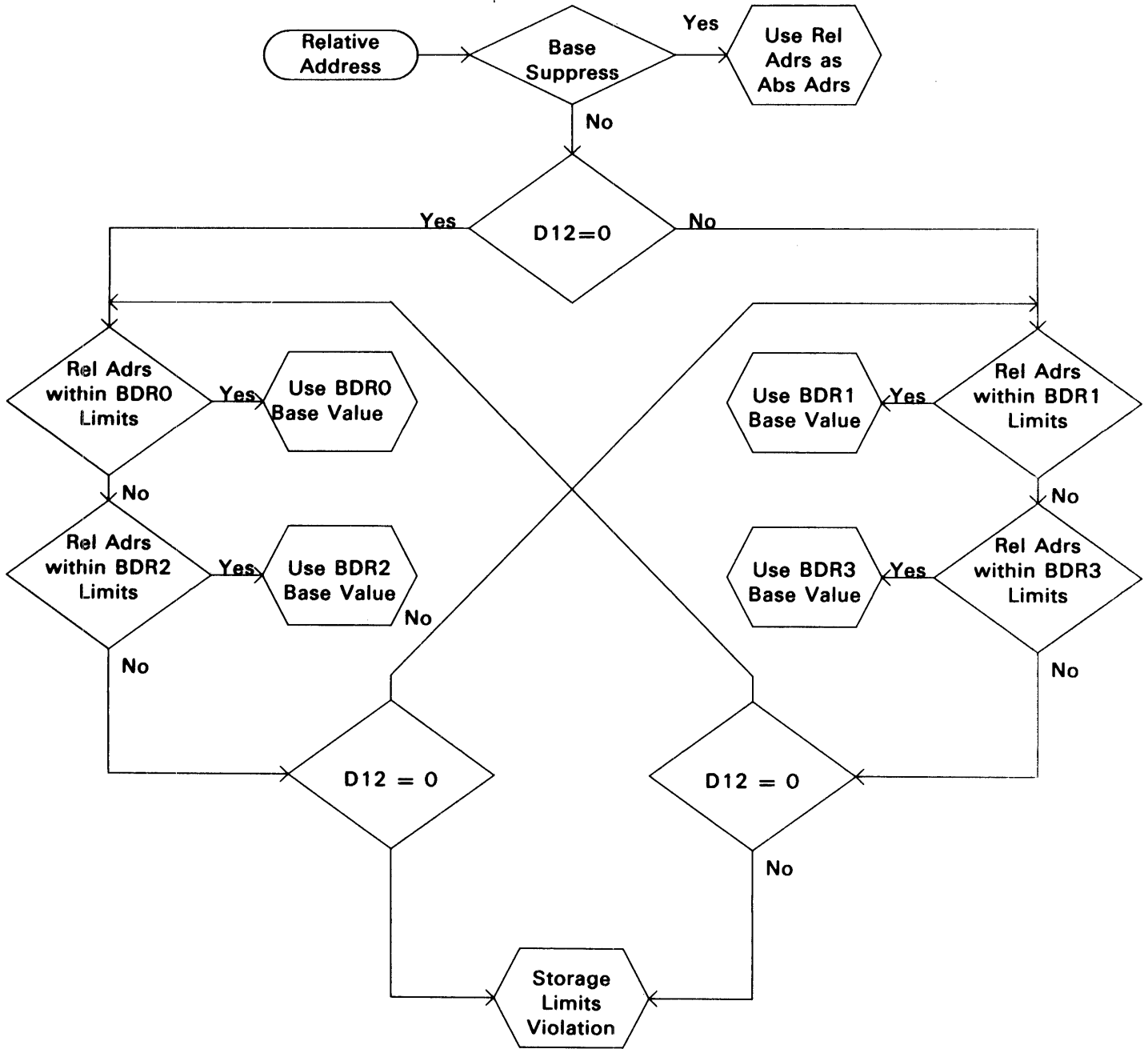
The lower limit (9 bits) of a BDR is checked against  $U_{17-9}$  and the upper limit (12 bits) is checked against  $U_{17-6}$ . The first BDR passing the limits check is used for absolute address generation. Figure 8-3 shows the base value selection in flow chart form. If a relative address is not within limits of any BDR a storage limits violation occurs. If a relative address is within limits then the base corresponding to those limits is used to convert the relative address to an absolute address with which to reference storage. Base values may be assigned in 64 word increments.

The base addition is done with base and relative address alignments shown below:

Relative address	zeroes	17	—	6	5	—	0	
Base value	17	—	12	11	—	0	zeroes	
Absolute address	23	—————						0

The base addition is end off; i.e., a carry produced out of bit 23 is not propagated into bit 0.





Define "Base Suppress" as:

$D7=1$  AND  $i=1$  AND NOT P-Fetch OR A-Flag AND P-Fetch

Figure 8-3. Base Value Selection

### 8.3.9. P-Capturing Instructions

The absolute address of each instruction read from main storage is normally held in a P-value register associated with the register holding the instruction. An exception is the case of an instruction read from an interrupt location when the CPU generates an interrupt: in this case the P-value register holds the absolute address of the most recently completed instruction rather than the absolute address of the interrupt location. When one of the P-capturing instructions (Store Location and Jump - SLJ, or Load Modifier And Jump - LMJ) is performed, the "relative P+1" is formed by subtracting either BI-1 or BD-1 from the contents of the associated P-value register. This relative P+1 address is stored by the SLJ or LMJ instruction.

When a program is operating with base register suppression ( $D7 = i = 1$ ), the base register suppression applies to each absolute address developed using the value in the u-field, but not to the captured relative address derived from the contents of the P-register. Base register suppression applies to the calculation of the absolute address at which the captured relative address is stored for the SLJ instruction. The jump to addresses for the LMJ and SLJ instructions are also calculated with base register suppression. The jump to address for the LMJ instruction is developed except that BI and BD are effectively zero so that  $(u + BI) + X_m$  and  $(u + BD) + X_m$  reduce to  $u + X_m$ . The jump to address for the SLJ instruction is developed except that BI and BD are effectively zero so that  $(0 + BI) + U + 1$  and  $(0 + BD) + U + 1$  reduce to  $U + 1$ .

If the SLJ instruction is used to capture the relative jump from address and transfer control to another sequence of instructions, the procedure for returning to the first sequence of instructions is simplified if the relative value captured is less than  $200000_8$ . If the relative value captured is  $200000_8$  or greater, it contains a 1 bit in bit 16 or 17 (or both). If this relative address is used as the right half of an instruction, a 1 bit in these positions will be interpreted for index register incrementation ( $X_m$  is modified) or indirect addressing (or base register suppression) rather than as bits used in developing an absolute address. However, if all the instructions for a program have relative addresses of  $177777_8$  or less, this situation will not arise.

## Appendix A. Glossary

A	An arithmetic register. GRS addresses $14 - 33_8$ and $154 - 173_8$ . Registers at addresses 34, 35, 174, and $175_8$ can be used either as general purpose registers or as extensions of the sets of A-registers. In some cases A is used to mean Aa.
Aa	The A-register specified explicitly by the a-field of an instruction word.
A+1 Aa+1	An A-register having an address one greater than the address of the A-register specified by the a-field of an instruction word.
A+2 Aa+2	An A-register having an address of two greater than the address of the A-register specified by the a-field of an instruction word.
Absolute Address	A 36-bit address which identifies a specific location in main storage, as opposed to the relative address.
a-field	A-register designator (bits 25-22) of an instruction word. The a-field is interpreted in one of several ways depending on the instruction word function code. The a-field may specify an A-register, an R-register, or an X-register. For the function code $70_8$ (JGD instruction), the j-field and a-field are combined to specify a GRS address. The a-field also is used to specify the I/O channels, a jump key, stop keys, or as an extension of the function code of the instruction.
<b>AND</b>	Logical product
ASCII	American Standard Code for Information Interchange (seven bits)
Bank	A set of main storage locations having consecutive addresses. Defined by a bank descriptor word (BDW). Bank addressing is achieved by loading a base value in a DR to be added to each bank relative address to produce the corresponding absolute address.
BD	Bank descriptor is a two word set of data defining storage allocation for a program segment.
BDI	Bank descriptor index. An integer value used as an index into a BDT.

<b>BDI Registers</b>	The two locations in the GRS which contain the BDIs (total of 4) for the banks currently addressable by the CPU. GRS locations 46 and 47 <sub>8</sub> .
<b>BDT</b>	Bank descriptor table.
<b>BDTP</b>	Bank descriptor table pointer
<b>Block Multiplexer</b>	A block multiplexer channel has multiple subchannels and always forces the I/O device to transfer data in multi-byte mode.
<b>Byte</b>	A unit of information which consists of eight bits data.
<b>Byte Count</b>	The number of bytes of data to be transferred to or from storage.
<b>Byte Multiplexer</b>	The byte multiplexer channel contains multiple subchannels and operates in either single or multi-byte mode.
<b>CAW</b>	The Channel Address Word contains the instruction, IOU and CPU number, channel address, device address and the address of the first CCW.
<b>CCW</b>	Channel command word. A control word located anywhere in storage (location specified by the CAW) used for channel operations. The CCW specifies the device command, data address, CCW flags, format flag and data count.
<b>Channel</b>	An I/O channel provides the hardware control and data paths required to direct the flow of data between a peripheral device and storage.
<b>Channel Base Register</b>	The contents of the channel base register are used to address control words in upper storage.
<b>Characteristic</b>	Biased exponent portion of a floating-point number.
<b>Condition Code</b>	Indicates the channels response during the execution of an instruction.
<b>Control Word</b>	Refer to CAW and CCW.
<b>Control Module</b>	The control module handles all I/O instructions and resolves storage request and interrupt conflicts for up to eight channel modules.
<b>Command Chaining</b>	Allows execution of a new channel command word whenever the present operation is complete at the device level. This will result in the specification of a new operation with the same device without program intervention.
<b>CPU</b>	Central processor unit.
<b>CSW</b>	Channel status word
<b>Data Chaining</b>	Specifies a new buffer area in storage and permits continuous operation of the device without program intervention.
<b>D-Bank</b>	A bank based on BD.

<b>Designator Bits D bits</b>	<b>These bits are used to establish and provide control of the processor operations and to report status. (See 8.2.1)</b>
D0	carry indicator
D1	overflow indicator
D2	guard mode and storage protection selector
D3	write-only storage protection selector
D4	character addressing mode selector
D5	double-precision underflow control (ignored if D20 = 0)
D6	A-, X-, and R-register set selector
D7	base register suppression control
D8	floating point zero control
D9	Reserved
D10	quarter-word mode selector
D11	Reserved
D12	BDR Selector
D13	BDR0 write protection
D14	BDR2 write protection
D15	BDR1 write protection
D16	BDR3 write protection
D17	floating-point residue store control
D18	Reserved
D19	BDTP selection control
D20	arithmetic exception interrupt control
D21	characteristic underflow indicator
D22	characteristic overflow indicator
D23	divide fault indicator
D29	quantum timer enable
<b>Device</b>	<b>A basic peripheral unit from or to which data is transferred in a system.</b>

Device Address	An address generated in the processor during an I/O instruction and by the control unit to indicate the address of the currently selected device. This is used to associate a particular device with a subchannel operation.
Double Word Boundary	Any even numbered storage address.
E bit	Bit 35 of the word in Xa for an LIJ/LDJ instruction and bits 35 and 17 of the BDI registers.
EF	External function. A control signal sent by an IOU to a peripheral control unit to identify the word on the output data lines as a function word rather than an output data word.
EI	External interrupt. A control signal sent to an IOU by a peripheral control unit which identifies the word on the input word lines as a status word rather than an input data word.
ESI	Externally specified index.
ESI Interface	Word channel interface capable of addressing up to 64 communications devices on one I/O interface.
f-field	Function code designator (bits 35-30) of an instruction word. The f-field specifies the particular type of operation or function to be performed. The j- and a-fields serve as minor function codes on certain instructions.
Granule	Any group of 512 contiguous words in main storage having addresses in the range XXXXX000 <sub>8</sub> through XXXXX777 <sub>8</sub> .
GRS	General register stack. A group of 112 addressable 36-bit control registers. The CpU uses these high-speed registers for holding intermediate results, indexing, and a variety of special functions such as repeat counting and holding status words.
h-field	Index register incrementation designator (bit 17) of an instruction word. The h-field controls index register modification and J-register modification.
lb	Increment in bytes. Bits 20-18 of a J-register. Used by byte instruction and other instructions operating in the character addressing mode (D4 = 1).
I bank	A bank based on BI.
i-field	Indirect addressing designator (bit 16) of an instruction word. The i-field normally controls indirect addressing. However, it may be used instead to specify base register suppression/24-bit indexing or use of the utility base for operands, depending on the values of D7, D9, and D11.
Immediate Command	An operation which will result in the subsystem generating an immediate status condition upon receipt of the command code.
Increment	The leftmost 18 bits (12 bits if D9 = D7 = i = 1) of an index register. Symbolized by Xi.
Instruction Word	A statement that specifies an operation and the values or locations of its operands.

I/O	Input/output
IOU	Input/output unit
ISI	Internally specified index
ISI Interface	A word channel interface which communicates with one peripheral control unit.
lw	Increment in words. Bits 31–21 of a J-register. Used by byte instructions and by instructions operating in the character addressing mode ( $D4 = 1$ ).
J	J-register (J0–J3) at GRS addresses 106–111 <sub>8</sub> or 126–131 <sub>8</sub> . Used by byte instructions and by instructions operating in the character addressing mode ( $D4 = 1$ ).
j-field	Operand qualifier, partial GRS address, or minor function code designator (bits 29–26) of an instruction word.
K	Used for notational convenience to replace the three low order digits of an intergral power of 2 or an integral multiple thereof. Thus 262K is used to represent 262,144( $2^{18}$ ).
LJ0 LJ1 LJ2	Indicates the number of bytes in string SJ0, SJ1, and SJ2, respectively. Stored in SR3 <sub>35–27</sub> , SR3 <sub>26–18</sub> , and SR3 <sub>17–8</sub> , respectively. Maximum value is 511.
Main Storage	The storage other than GRS registers that can be accessed directly by a CPU and IOU; it consists of primary storage and extended storage.
Major Function Code	The f-field of an instruction word.
Mantissa	The fractional part of a floating-point number.
Minor Function Code	A portion of an instruction word used with the f-field to specify the operation to be performed. For all instructions for which $f = 07, 33, \text{ or } 37_8$ or for which $f$ is greater than 70 <sub>8</sub> , the j-field contains a minor function code. For some instructions for which $f$ is greater than 70 <sub>8</sub> , the a-field also contains a minor function code.
MMA	Multiple module access unit.
Modifier	The rightmost 18 bits (24 bits if $D9 = D7 = i = 1$ ) of an index register. Symbolized by $X_m$ . It is added to the 16-bit address in the u-field of an instruction to produce a relative address ( $X_m$ is 18 bits) or absolute address ( $X_m$ is 24 bits).
MSR	Module select register
Multi-Byte Mode	A type of operation available on the byte channel which permits a control unit to transfer several bytes of data before releasing the channel.
NI	Next instruction
Nonresident Subchannel	A set of control words held in reserve storage.

Nonshared Subchannel	A subchannel intended to operate with communications type peripheral devices. These subchannels allow concurrent access in an interleaving manner by a multiple number of devices through a multiplexing control unit to main storage.
Normalize	To normalize a number in floating point format, the mantissa is shifted left or right until the leftmost bit of the mantissa is not identical to the sign bit.
Ob	Offset in bytes. Bits 2-0 of a J-register. Used by byte instructions and by instructions operating in the character addressing mode ( $D4 = 1$ ).
Option 0	Used with the subchannel expansion feature to provide four resident subchannels, four resident nonshared subchannels and 124 nonresident nonshared subchannels.
Option 1	Provides 128 nonshared subchannels. The eight most recently active are held in the channel. The remaining 120 subchannels are held in main storage.
<b>OR</b>	Logical inclusive OR
Ow	Offset in words. Bits 17-3 of a J-register. Used by byte instructions and by instructions operating in the character addressing mode ( $D4 = 1$ ).
P P Value Register	P is the absolute address of the current instruction. It is loaded in a P-value register associated with the instruction register which receives an instruction read from main storage and accompanies the instruction as it flows through the instruction stack. The contents of the P-value register are not changed when an instruction is read from an interrupt location or when an instruction is read for an Execute instruction.
Parity Bit	A binary digit appended to a group of bits to make the number of one bits always odd or always even.
PCI	Program controlled interrupt. (See 6.5.1.)
Program Controlled Interrupt	A program settable bit in a CCW. When set, an interrupt and/or a table entry in the status table is made for that subchannel.
R	A special purpose control register specified explicitly or implicitly by an instruction word. GRS addresses $100_8 - 117_8$ and $120_8 - 137_8$ .
Ra	The R-register specified by the a-field of an instruction word.
Relative Address	Normally, the address (U) formed by the addition of u, the address field of an instruction, and $X_m$ , the modifier portion of the index register specified by the instruction ( $U = u + X_m$ ). For byte instructions and instructions performed in the character addressing mode, the relative address is $U = u + X_m + Ow$ . A relative address is not produced for instructions performed with base register suppression.
Relative P+1	An 18-bit relative address captured by certain jump instructions. Formed by subtracting the active PSRs BI or BD value which corresponds to the value used to develop the absolute jump to address for the most recent previous jump instruction from the address of the instruction following the current jump instruction.



Resident Subchannel	A set of control word held in the channel module.
Residue	The least significant result word produced by a single-precision Floating Add or Floating Add Negative instruction.
RTC	Real time clock
R0	Real-time clock register at GRS address $100_8$ , or the control register at GRS address $120_8$ .
R1	Repeat count control registers at GRS addresses $101$ and $121_8$ . They are used during Block Transfer, search, and masked search instructions.
R2	Mask control registers at GRS addresses $102$ and $122_8$ . They are used during masked search instructions and the Masked Load Upper instruction.
R3 – R5	Staging Register 1–3 (SR1 – SR3). Used by byte instructions.
R6 – R9	J-registers J0 – J3. One or more of these registers are used by byte instructions and by instructions operating in the character addressing mode ( $D4 = 1$ ).
S	Sign bit or bit position
SD	The 24-bit D-bank absolute address developed through addition: $SD = (u + BD) + X_m$ or $SD = (u + BD) + X_m + O_w$ .
Shared Subchannel	A subchannel is shared if two or more devices use the same subchannel for I/O operations. On a shared subchannel only one device at a time can transfer data.
SI	The 24-bit I-bank absolute address developed through addition: $SI = (u + BI) + X_m$ or $SI = (u + BI) + X_m + O_w$ .
SIOF Queue	Used for storing the device address for SIOF instructions presented by the processor but not yet executed by the IOU.
SIU	Buffer storage
SJ0	A byte string whose starting word address is formed by summing the u-field of the instruction, the modifier of the index register specified by the instruction word, and the $O_w$ field of register J0. The $O_b$ -field of J0 points to a byte within a word.
SJ1	A byte string based on J1, $X+1$ , and $O_w$ in the same manner as SJ0 is based on J0, X, and $O_w$ .
SJ2	A byte string based on J2, $X+2$ , and $O_w$ in the same manner as SJ0 is based on J0, X, and $O_w$ .
SK	Skip data (See 6.5.1)
SR1	Staging register 1 (R3), GRS addresses $103_8$ or $123_8$ .
SR2	Staging register 2 (R4), GRS addresses $104_8$ or $124_8$ .

SR3	Staging register 3 (R5), GRS addresses 105 <sub>8</sub> or 125 <sub>8</sub> .
STCW	Status table control word
STU	System transition unit
Subchannel	A subchannel is an organization of uniquely addressable access paths which are capable of independently sustaining a single I/O operation concurrent with other I/O operations; i.e., a set of control words.
Subchannel Expansion Feature	Provides the capability for a channel module to operate with nonshared subchannels. (Refer to Option 0 and Option 1 for an explanation.)
Subsystem Clear	An I/O Clear signal originating at the IOU goes out on all 24 channels of that IOU.
System Reset	Clears all IOU registers and control designators, resets all peripheral subsystems and initializes all resident subchannels to idle mode.
TIC	Transfer in Channel. A command stored as part of the CCW to perform a branch between noncontiguous CCWs.
TIO	Test I/O
TSW	Tabled Status Word
U	The 18-bit value produced in the index subsection by adding the rightmost 18 bits (X <sub>m</sub> -field) of the index register specified by the x-field of the instruction (or by adding 0 if X = 0) to the 16-bit value in the u-field of the instruction (u-field is extended to 18 bits). $U = u + X_m$ or $U = u + X_m + O_w$ .
u-field	The contents of bit positions 15-0 of an instruction word.
V-field	The relative address contained in bits 17-0 of an ISI or ESI access control word.
W-field	The count field of an access control word. For ISI operations, the W-field is bits 33-18. For half-word ESI operations, the W-field is bits 32-18. For quarter-word ESI operations, the W-field is bits 29-18.
Word Interface	A set of cable drivers and receivers for communicating with one peripheral control unit.
X	Index register. GRS addresses 1 <sub>8</sub> - 17 <sub>8</sub> and 141 <sub>8</sub> - 157 <sub>8</sub> .
X+1	An index register having an address one greater than the address of the index register specified by the x-field of an instruction word.
X+2	An index register having an address two greater than the address of the index register specified by the x-field of an instruction word.
Xa	The X-register specified by the a-field of an instruction word.
Xi	Normally, bits 35-18 of an index register (bits 35-24 when D9 = D7 = i = 1). Used to increment or decrement X <sub>m</sub> (the modifier) when specified by the instruction word.

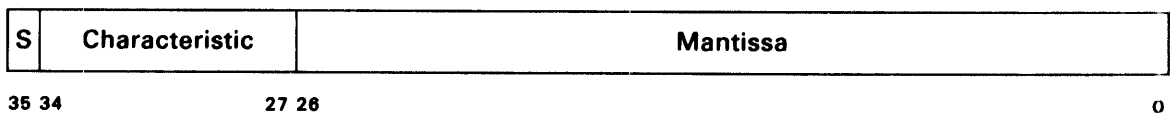
x-field	Index register designator (bits 21-18) of an instruction word.
X <sub>m</sub>	Normally, bits 17-0 of an index register (bits 23-0 when D <sub>9</sub> = D <sub>7</sub> = i = 1).
<b>XOR</b>	Logical exclusive OR
X <sub>x</sub>	The X-register specified by the x-field of instruction word. In some cases X is used to mean X <sub>x</sub> .
+0	Two words, one word, or a-field consisting of all 0 bits.
-0	Two words, one word, or a-field consisting of all 1 bits.
( )	The contents of the register or location identified by the symbol within the parentheses.
( )'	The ones complement of the register or location identified by the symbol within the parentheses.
( ) <sub>n</sub>	The contents of bit position n of the register or location identified by the symbol within the parentheses. For example, (A) <sub>35</sub> means the contents of bit position 35 of A.
( ) <sub>n-m</sub>	The contents of bit positions n through m of the register or location identified by the symbol within the parentheses. For example, (X) <sub>17-0</sub> means the contents of bit position 17 through 0 of X.
	Absolute value or magnitude
→	Direction of data flow



## Appendix B. Summary Of Word Formats

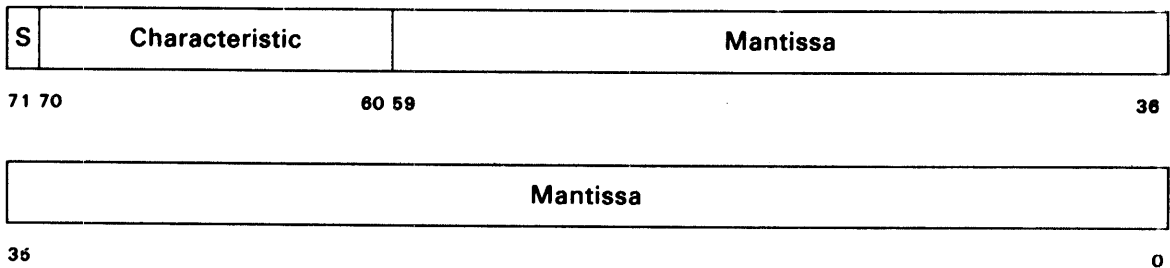
See 4.1.8 for the following:

### *Single-Precision Floating-Point Format*



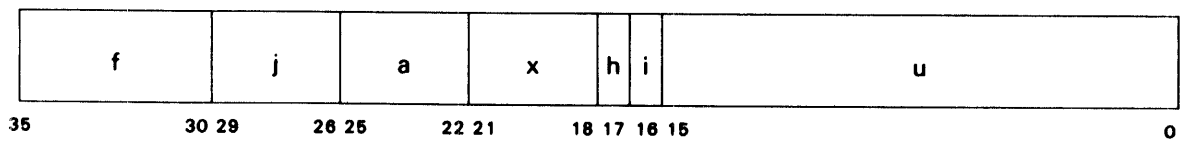
See 4.1.8 for the following:

### *Double-Precision Floating-Point Format*



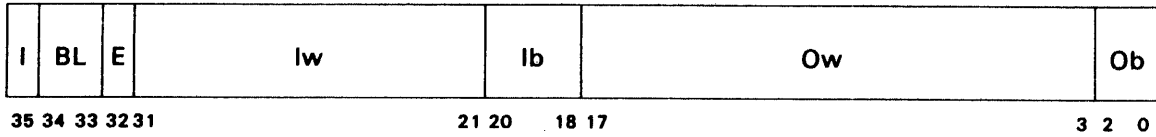
See 4.2.1 for the following:

### *Instruction Word Format*



See 4.2.2.2.2 for the following:

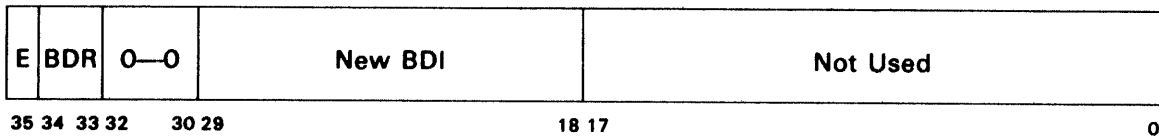
*J-Register Format For Character Addressing Mode*



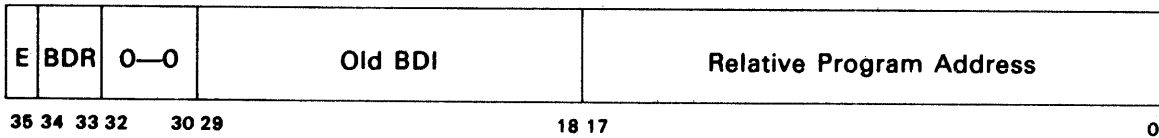
See 5.10.1 for the following:

*Load Bank And Jump*

*Xa Before Execution*

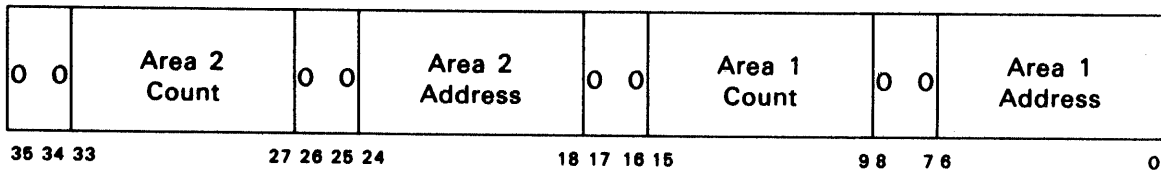


*Xa After Execution*



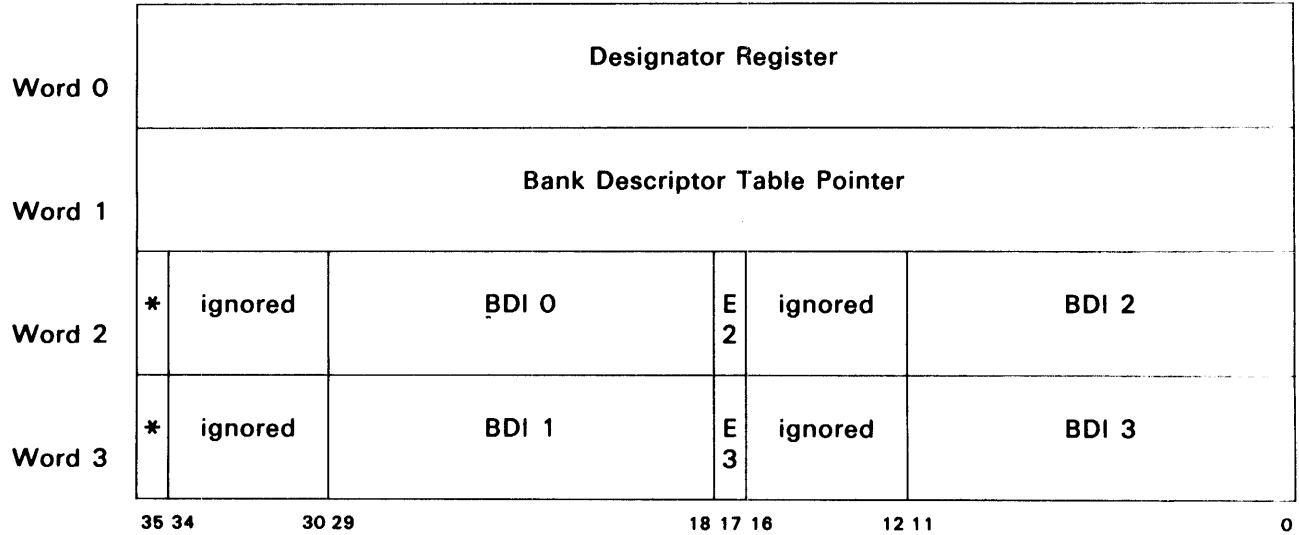
See 5.13.9 for the following:

*Aa Format For Store Register Set*



See 5.13.11 for the following:

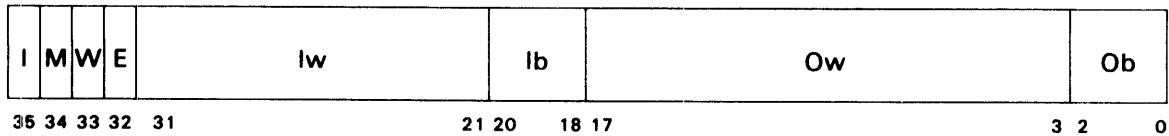
*Test Relative Address*



\* Not Used

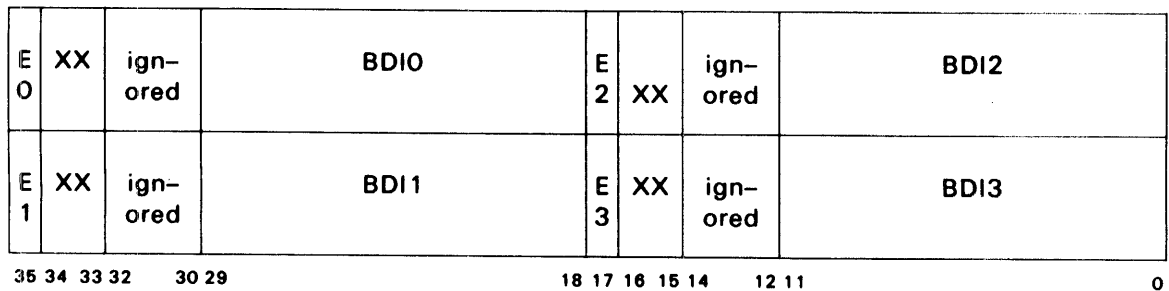
See 5.14 for the following:

*J-Register Format*



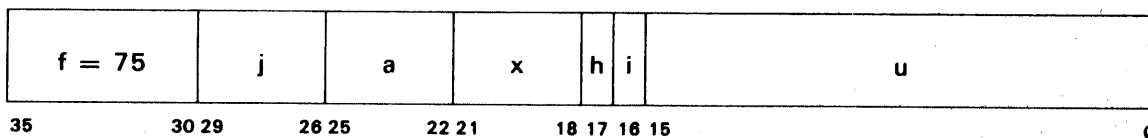
See 5.15.10 for the following:

*Load Address Environment*



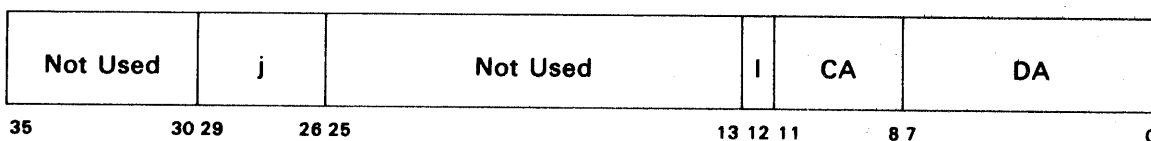
See 6.3.4 for the following:

*I/O Instruction Format*



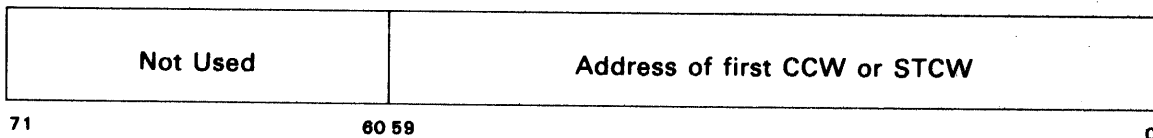
See 6.3.4 for the following:

*CAW*



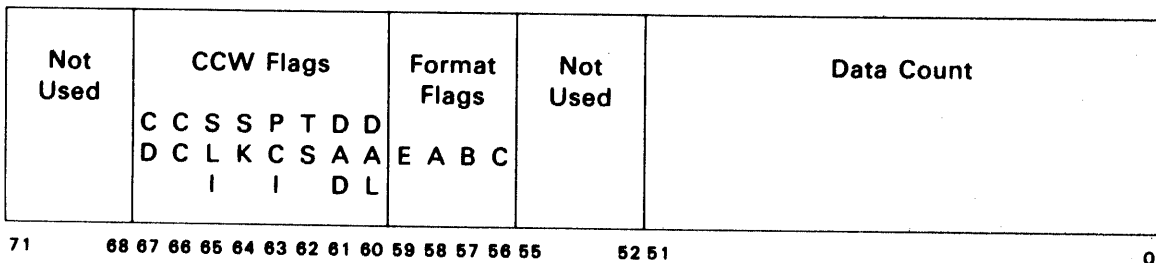
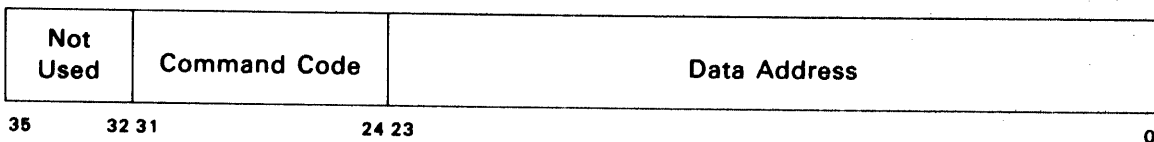
See 6.3.4 for the following:

*CAW 1*



See 6.5.1 for the following:

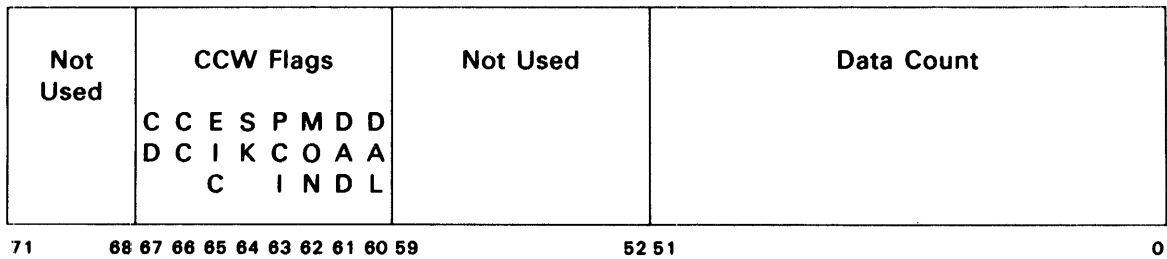
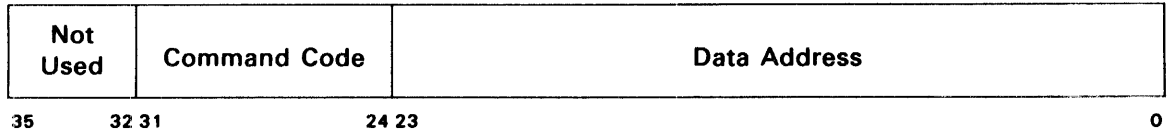
*Byte Or Block Multiplexer Channel CCW*





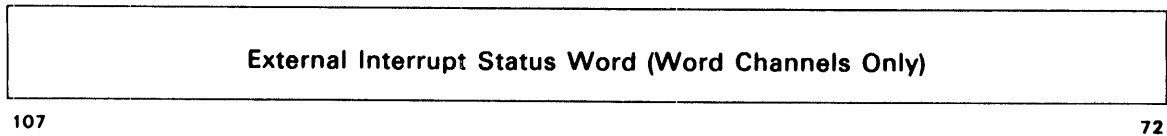
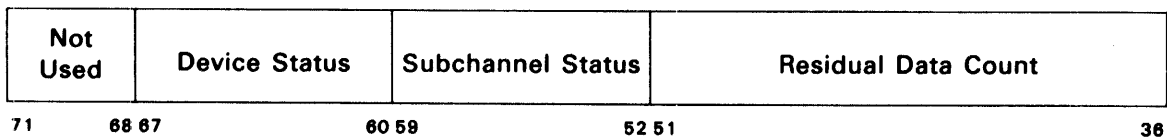
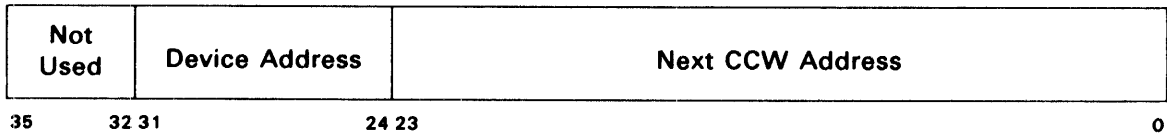
See 6.5.1 for the following:

*Word Channel CCW*



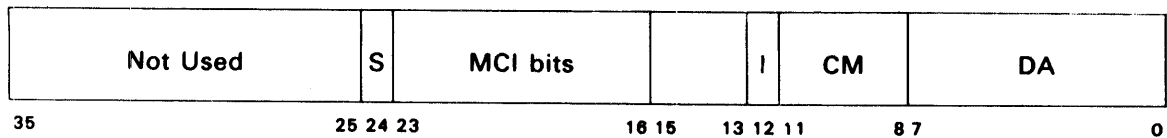
See 6.10 for the following:

*CSW or TSW*



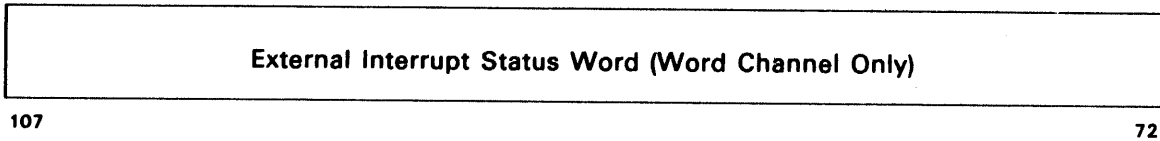
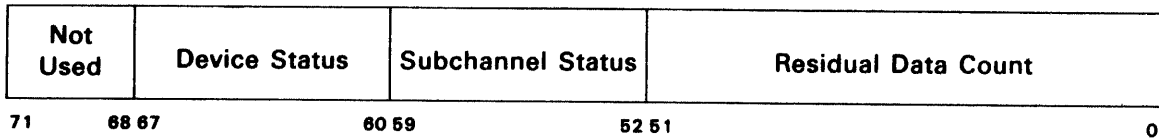
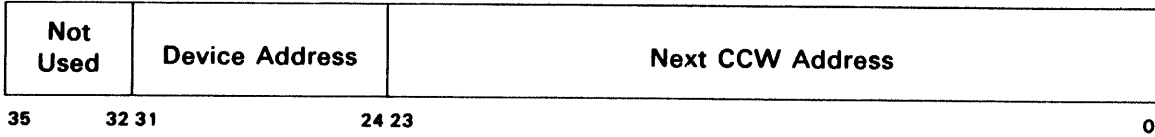
See 6.10 for the following:

*IAW*



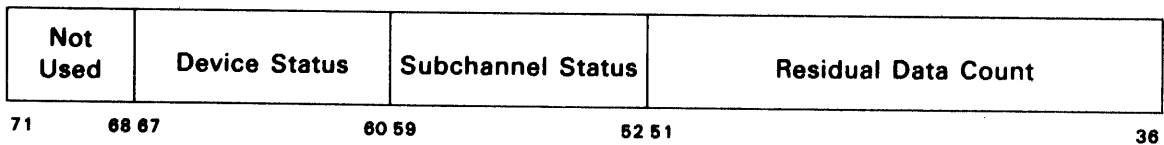
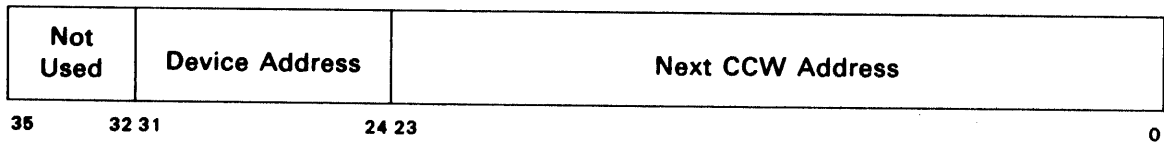
See 6.11 for the following:

*CSW For I/O Instruction*



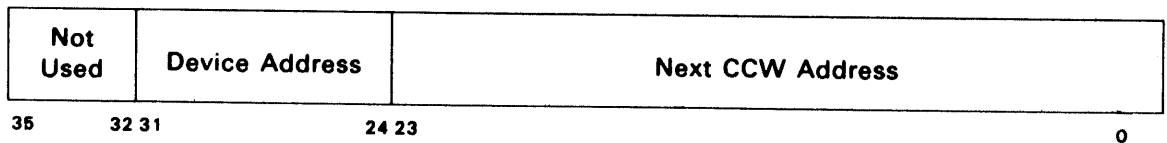
See 6.12 for the following:

*TSW for Non-shared Byte Mux Subchannels*



See 6.12 for the following:

*TSW for ESI Word Subchannels*



Not Used	EI 0 0 0 0 0 0 0	Subchannel Status	Residual Data Count
71	68 67 66	60 59	52 51
			36

External Interrupt Status Word if Bit 67 is Set
107
72

Not Used
144
108

See 6.13 for the following:

*CSW For The Store Subchannel Status Command*

Not Used	Device Address	Next CCW Address
35	32 31	24 23
		0

Not Used	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	Residual Data Count
71	68 67	60 59	52 51
			36

See 6.17 for the following:

*Scratch Pad Formats for Subchannel Expansion Feature*

Not Used	Format Control	Mode	Data Address		
35	32 31	28 27	24 23	0	

Not Used	CCW Flags	Format Flags	Not Used	Data Count	
35	32 31	24 23	20 19	18 15	0

Not Used	Device Address	Next CCW Address			
35	32 31	24 23	0		

Not Used					
35					0

Not Used	Not Used	Mode	Device Address	Not Used	
35	32 31	28 27	24 23	18 15	0

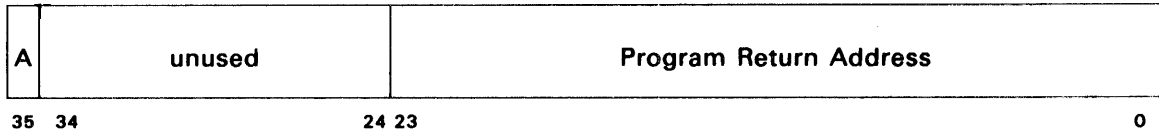
Not Used	Device Status	Subchannel Status	Data Count		
35	32 31	24 23	18 15	0	

Not Used	Device Address	Next CCW Address			
35	32 31	24 23	0		

Not Used					
35					0

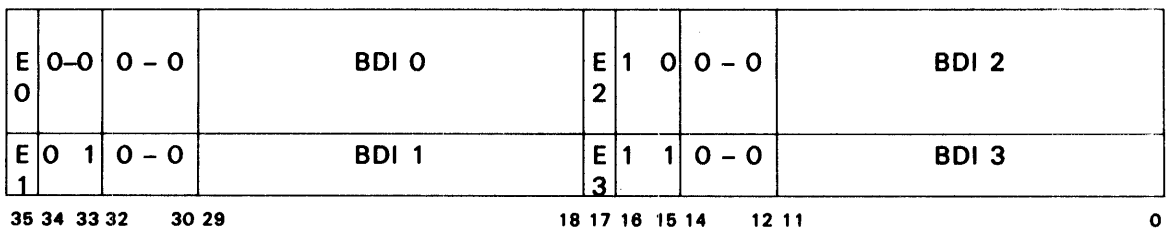
See 7.2.1 for the following:

*Program Return Address*



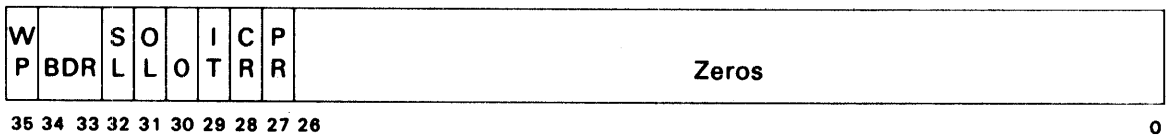
See 7.2.2 for the following:

*Addressing Status*



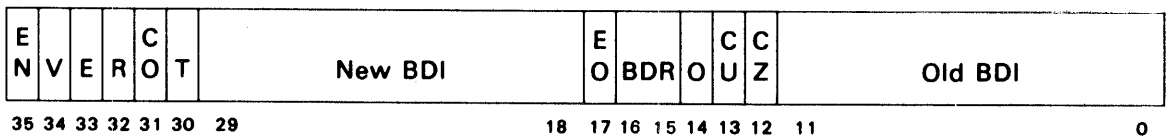
See 7.3.2 for the following:

*Format of Guard Mode Interrupt Status*



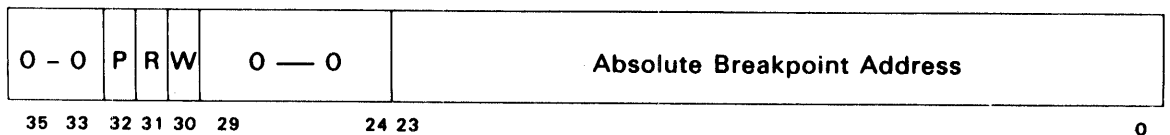
See 7.3.2 for the following:

*Format of Addressing Exception Interrupt Status*



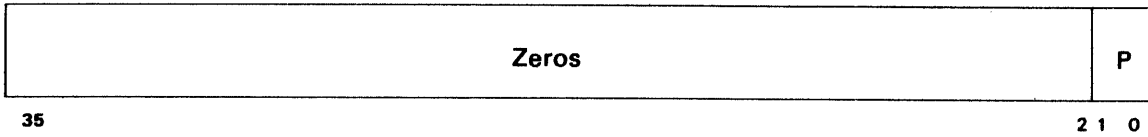
See 7.3.2 for the following:

*Format Of Breakpoint Interrupt Status*



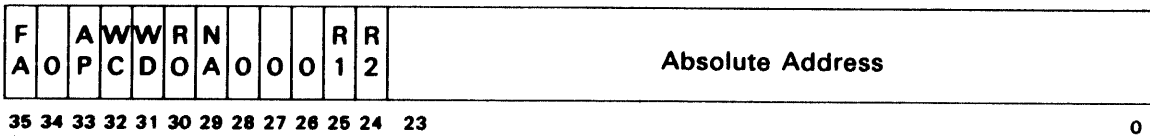
See 7.3.3 for the following:

*Format of Interprocessor Interrupt Status*



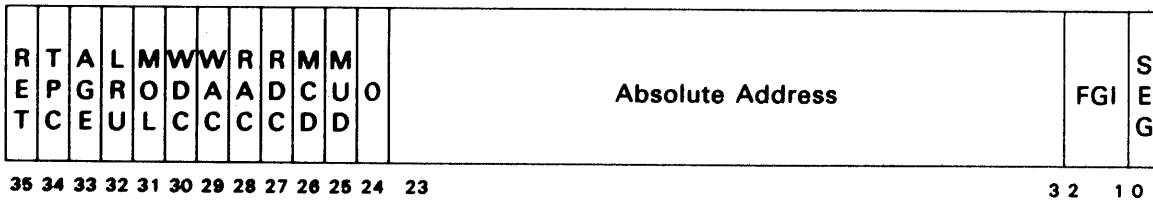
See 7.3.6.1 for the following:

*Format of Immediate Storage Check Interrupt Status*



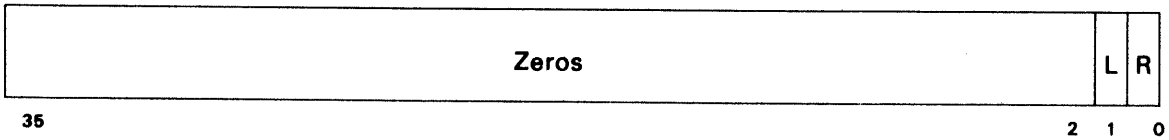
See 7.3.6.2.2 for the following:

*Storage Check Interrupt Status Word*



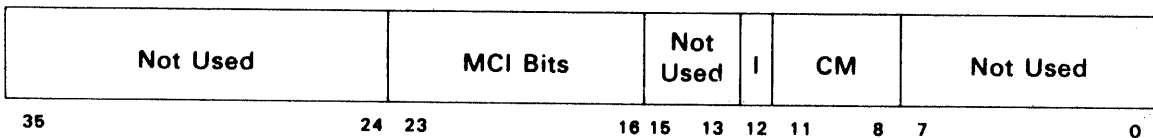
See 7.3.8 for the following:

*Power Check Interrupt Status*



See 7.4.1 for the following:

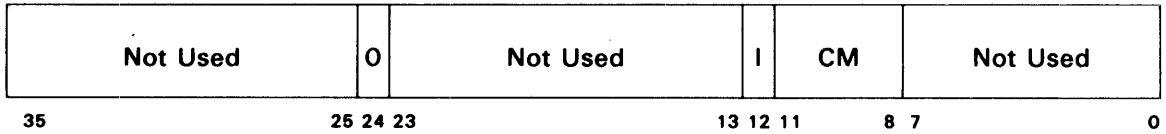
*Machine Check IAW*



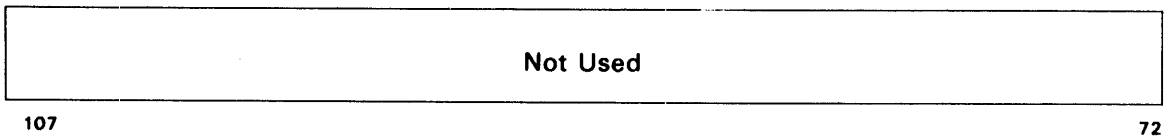
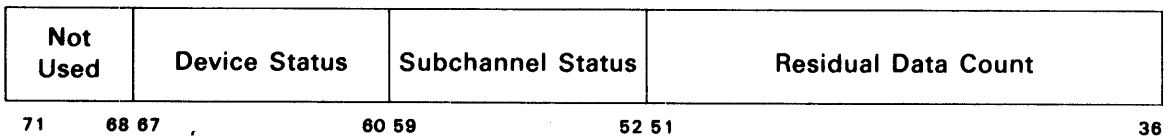
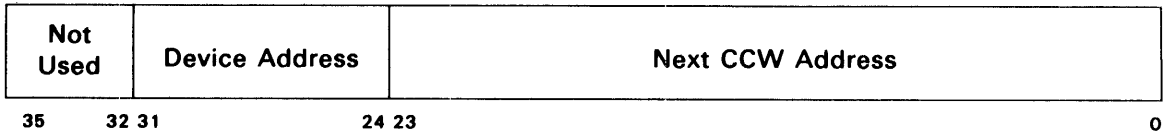
See 7.4.2 for the following:

*Normal Interrupt IAW and CSW for a Byte or Block Multiplexer Channel*

*IAW*



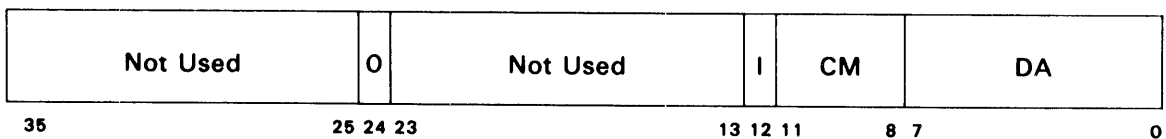
*CSW*



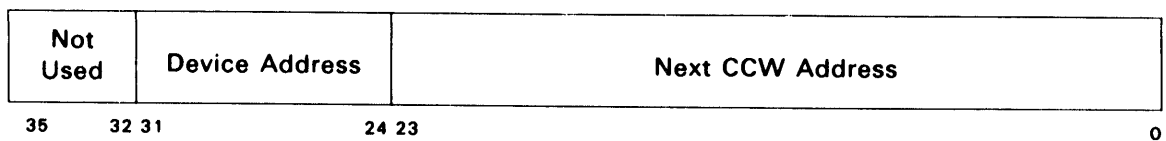
See 7.4.2 for the following:

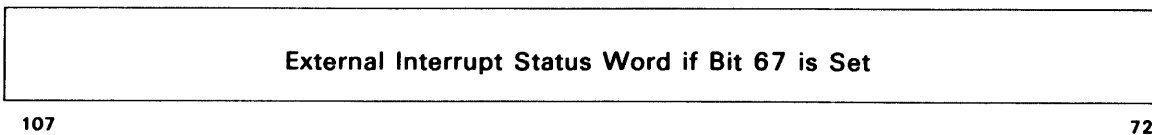
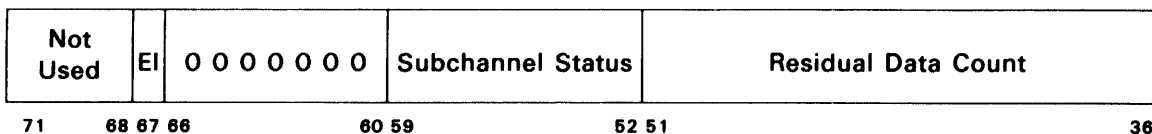
*Normal Interrupt IAW And CSW For Word Channel*

*IAW*



*CSW*

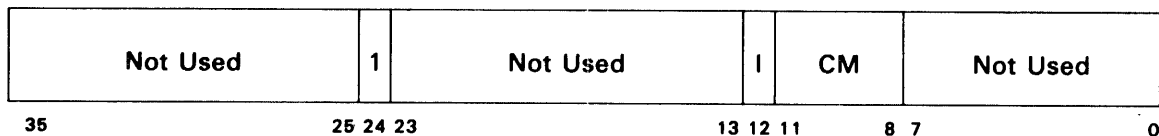




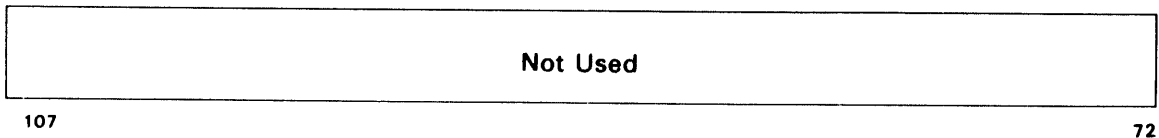
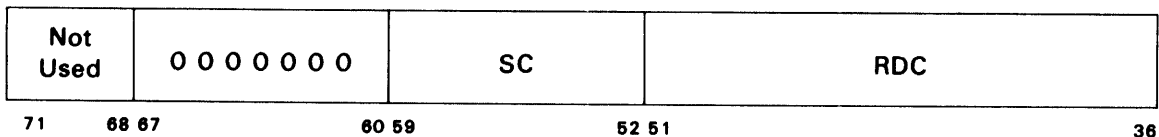
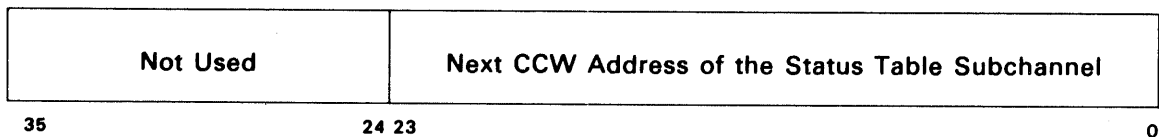
See 7.4.2 for the following:

*Normal Interrupt IAW and CSW for Status Table Subchannel*

*IAW*



*CSW*

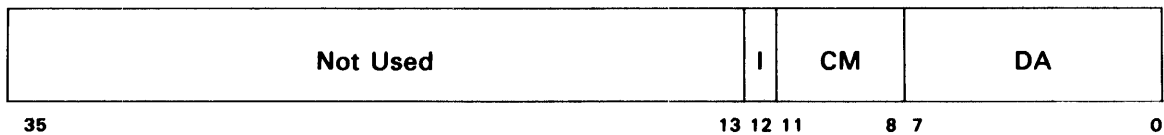




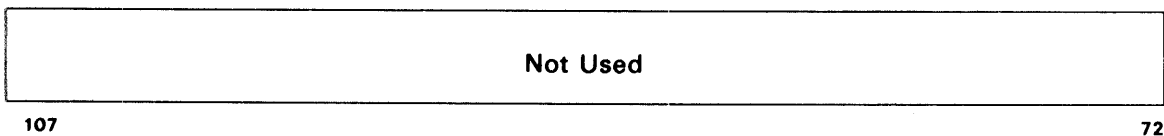
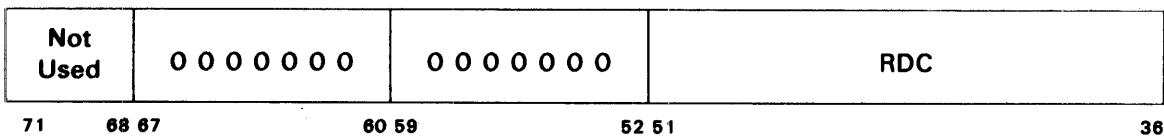
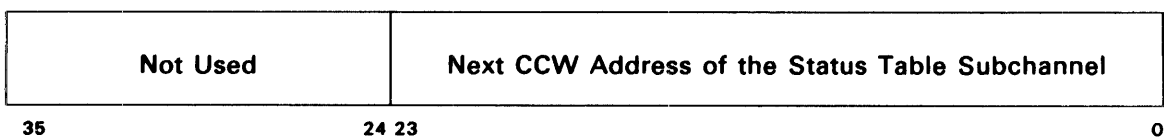
See 7.4.3 for the following:

*Tabled Interrupt IAW and CSW*

*IAW*

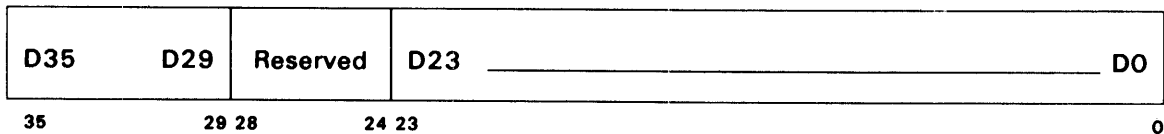


*CSW*



See 8.2.1 for the following:

*Designator Register*





## Appendix C. User Instruction Repertoire

Table C-1. Mnemonic/Function Code Cross-Reference

Mnemonic/Function Code Cross-Reference

Mnemonic	Function Code (Octal)		Paragraph Reference
	f	j	
A,AA	14		5.4.1
A,AX	24		5.4.7
AAIJ	74	07	5.9.3
AH	72	04	5.4.17
AM,AMA	16		5.4.3
AN,ANA	15		5.4.2
AN,ANX	25		5.4.8
AND	42		5.12.3
ANH	72	05	5.4.18
ANM,ANMA	17		5.4.4
ANT	72	07	5.4.20
ANU	21		5.4.6
AT	72	06	5.4.19
AU	20		5.4.5
BA	37	06	5.14.14
BAN	37	07	5.14.15
BC	33	04	5.14.4
BDF	33	15	5.14.11
BDI	33	11	5.14.7
BF	33	14	5.14.10
BI	33	10	5.14.6
BM	33	00	5.14.1
BMT	33	01	5.14.2
BT	22		5.3.8
BTC	33	03	5.14.3

Mnemonic/Function Code Cross-Reference (continued)

Mnemonic	Function Code (Octal)		Paragraph Reference
	f	j	
CDU	76	07	5.5.16
DA	71	10	5.4.15
DAN	71	11	5.4.16
DDC	73	14	5.15.2
	a = 12		
DF	36		5.4.14
DFA	76	10	5.5.3
DFAN	76	11	5.5.4
DFB	33	17	5.14.13
DFD	76	13	5.5.8
DFM	76	12	5.5.6
DFP,DLCF	76	15	5.5.12
DFU	76	14	5.5.10
DI	34		5.4.12
DIB	33	13	5.14.9
DJZ	71	16	5.11.2
DL	71	13	5.2.9
DLM	71	15	5.2.11
DLN	71	14	5.2.10
DLSC	73	07	5.8.8
DS	71	12	5.3.7
DSA	73	05	5.8.6
DSC	73	01	5.8.2
DSF	35		5.4.13
DSL	73	03	5.8.4

Mnemonic/Function Code Cross-Reference (continued)

Mnemonic	Function Code (Octal)		Paragraph Reference
	f	j	
DTE	71	17	5.7.14
EDC	73	14	5.15.2
	a = 11		
EDIT	33	07	5.14.5
ER	73	14	5.13.4
EX	72	10	5.13.3
FA	76	00	5.5.1
FAN	76	01	5.5.2
FB	33	16	5.14.12
FCL	76	17	5.5.14
FD	76	03	5.5.7
FEL	76	16	5.5.13
FM	76	02	5.5.5
HCH	75	05	6.4.6
HDV	75	04	6.4.5
HJ	74	05	5.11.10
IB	33	12	5.14.8
J	74	04	5.11.9
JB	74	11	5.11.12
JC	74	16	5.11.22
JDF	74	14	5.11.17
	a = 3		
JFO	74	14	5.11.16
	a = 2		
JFU	74	14	5.11.15
	a = 1		
JGD	70		5.11.1
JMGI	74	12	5.11.13
JN	74	03	5.11.8
JNB	74	10	5.11.11
JNC	74	17	5.11.23
JNDF	74	15	5.11.21
	a = 3		
JNFO	74	15	5.11.20
	a = 2		
JNFU	74	15	5.11.19
	a = 1		
JNO	74	15	5.11.18
	a = 0		
JNS	72	03	5.11.4
JNZ	74	01	5.11.6
JO	74	14	5.11.14
	a = 0		
JP	74	02	5.11.7
JPS	72	02	5.11.3
JZ	74	00	5.11.5
L,LA	10		5.2.1
L,LR	23		5.2.5

Mnemonic/Function Code Cross-Reference (continued)

Mnemonic	Function Code (Octal)		Paragraph Reference
	f	j	
L,LX	27		5.2.7
LAE	73	15	5.15.10
	a = 12		
LB	73	15	5.15.8
	a = 10		
LBJ	07	17	5.10.1
LBX	73	15	5.15.5
	a = 02		
LCF	76	05	5.5.11
LCR	75	10	6.4.7
LD	73	15	5.15.12
	a = 14		
LDJ	07	12	5.10.3
LDSC	73	11	5.8.10
LDSL	73	13	5.8.12
LIJ	07	13	5.10.2
LL	73	15	5.15.9
	a = 11		
LM,LMA	12		5.2.3
LMJ	74	13	5.9.2
LN,LNA	11		5.2.2
LNМ,LNMA	13		5.2.4
LPD	07	14	5.13.1
LQT	73	15	5.15.7
	a = 03		
LRS	72	17	5.13.10
LSC	73	06	5.8.7
LSSC	73	10	5.8.9
LSSL	73	12	5.8.11
LTCW	75	11	6.4.8
LUF	76	04	5.5.9
LXI	46		5.2.8
LXM	26		5.2.6
MASG	71	07	5.6.14
MASL	71	06	5.6.13
MCDU	76	06	5.5.15
MF	32		5.4.11
MI	30		5.4.9
MLU	43		5.12.4
MSE	71	00	5.6.7
MSG	71	03	5.6.10
MSI	31		5.4.10
MSLE,MSNG	71	02	5.6.9
MSNE	71	01	5.6.8
MSNW	71	05	5.6.12
MSW	71	04	5.6.11
NOP	74	06	5.13.8
OR	40		5.12.1

*Mnemonic/Function Code Cross-Reference (continued)*

Mnemonic	Function Code (Octal)		Paragraph Reference
	f	j	
PAIJ	72	13	5.15.1
RAT	73	15	5.15.15
	a = 15		
S,SA	01		5.3.1
S,SR	04		5.3.4
S,SX	06		5.3.6
SD	73	15	5.15.13
	a = 15		
SDC	73	14	5.15.3
SE	62		5.6.1
SG	65		5.6.4
SIL	73	15	5.15.4
SIOF	75	01	6.4.2
SLE,SNG	64		5.6.3
SLJ	72	01	5.9.1
SM,SMA	03		5.3.3
SN,SNA	02		5.3.2
SNE	63		5.6.2
SNW	67		5.6.6
SPD	07	15	5.13.2
SPID	73	15	5.15.6
	a = 0		
SQT	73	15	5.15.11
	a = 13		
SRL	75	00	6.4.1
SRS	72	16	5.13.9
SSA	73	04	5.8.5
SSC	73	00	5.8.1
SSL	73	02	5.8.3
SSS	73	15	5.15.17
	a = 17		
SW	66		5.6.5
TAP	73	15	5.15.16
	a =		
	07		
TCS	73	17	5.13.7
	a = 2		
TE	52		5.7.6
TEP	44		5.7.1
TG	55		5.7.9
TIO	75	02	6.4.3
TLE,TNG	54		5.7.8
TLEM,TNGM	47		5.7.3
TN	61		5.7.13
TNE	53		5.7.7
TNW	57		5.7.11
TNZ	51		5.7.5
TOP	45		5.7.2

*Mnemonic/Function Code Cross-Reference (continued)*

Mnemonic	Function Code (Octal)		Paragraph Reference
	f	j	
TP	60		5.7.12
TRA	72	15	5.13.11
TS	73	17	5.13.5
	a = 0		
TSC	75	03	6.4.6
TSS	73	17	5.13.6
	a = 1		
TW	56		5.7.10
TZ	50		5.7.4
UR	73	15	5.15.14
	a = 16		
XOR	41		5.12.2
XX	05	00-17	5.3.5.
	a = 00-07		
XX	05	00-17	5.13.12
	a = 10-17		
-	73	14	5.15.20

Table C-2. Instruction Repertoire

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
00	0-17	-	Illegal Operation	Causes Illegal Operation Fault interrupt to MSR + 241 <sub>g</sub>
01	0-15	S,SA	Store A	(Aa) → U
02	0-15	SN,SNA	Store Negative A	- (Aa) → U
03	0-15	SM,SMA	Store Magnitude A	(Aa)   → U
04	0-15	S,SR	Store R	(Ra) → U
05	00-17	XX	Increase Instructions	Constant → U
05	00-17	SZ a = 00	Store Zero	Store constant 000000 000000, Zeros, in location specified by operand address
05	00-17	SNZ a = 01	Store Negative Zero	Store constant 777777 777777, all Ones, in location specified by operand address
05	00-17	SP1 a = 02	Store Positive One	Store constant 000000 000001, positive One, in location specified by operand address
05	00-17	SN1 a = 03	Store Negative One	Store constant 777777 777776, negative One, in location specified by operand address
05	00-17	SFS a = 04	Store Fielddata Spaces	Store constant 050505 050505, Fielddata Spaces, in location specified by operand address
05	00-17	SFZ a = 05	Store Fielddata Zeros	Store constant 606060 606060, Fielddata Zeros, in location specified by operand address
05	00-17	SAS a = 06	Store ASCII Spaces	Store constant 040040 040040, ASCII Spaces, in location specified by operand address
05	00-17	SAZ a = 07	Store ASCII Zeros	Store constant 060060 060060, ASCII Zeros, in location specified by operand address

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
05	00-17	XX	Increase Instructions	
05	00-17	INC a = 10	Increase Operand by one	Increase operand by one. If initial operand or result is zero, execute NI; if not zero, skip NI.
05	00-17	DEC a = 11	Decrease Operand by one	Decrease operand by one. If initial operand or result is zero, execute NI; if not zero, skip NI.
05	00-17	INC2 a = 12	Increase Operand by two	Increase operand by two. If initial operand or result is zero, execute NI; if not zero, skip NI.
05	00-17	DEC2 a = 13	Decrease Operand by two	Decrease operand by two. If initial operand or result is zero, execute NI; if not zero, skip NI.
05	00-17	ENZ a = 14-17	Increase Operand by zero	Increase operand by zero. If initial operand or result is zero execute NI; if not zero, skip NI.
06	0-15	S,SX	Store X	(Xa) → U
07	12	LDJ	Load D-Bank Base and Jump	Ignore Xa bit positions 34-33; if D12 = 0, select BDR2; if D12 = 1, select BDR3
07	13	LIJ	Load I-Bank Base And Jump	Ignore Xa bit positions 34-33; if D12 = 0, select BDR0; if D12 = 1, select BDR1
07	14	LPD	Load PSR Designators	U <sub>6,5,3-0</sub> → PSRM Bit 6 → D20    Bit 2 → D8 Bit 5 → D17    Bit 1 → D5 Bit 3 → D10    Bit 0 → D4
07	15	SPD	Store PSR Designators	PSRM D-bits → U <sub>6-0</sub> D20 → Bit 6    D8 → Bit 2 D17 → Bit 5    D5 → Bit 1 D12 → Bit 4    D4 → Bit 0 D10 → Bit 3
07	17	LBJ	Load Bank And Jump	Load BDR; jump to location specified by the operand address
10	0-17	L,LA	Load A	(U) → A

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
11	0-17	LN,LNA	Load Negative A	$-(U) \rightarrow A$
12	0-17	LM,LMA	Load Magnitude A	$ (U)  \rightarrow A$
13	0-17	LNMA	Load Negative Magnitude A	$- (U)  \rightarrow A$
14	0-17	A,AA	Add to A	$(A) + (U) \rightarrow A$
15	0-17	AN,ANA	Add Negative To A	$(A) - (U) \rightarrow A$
16	0-17	AM,AMA	Add Magnitude To A	$(A) +  (U)  \rightarrow A$
17	0-17	ANM,ANMA	Add Negative Magnitude to A	$(A) -  (U)  \rightarrow A$
20	0-17	AU	Add Upper	$(A) + (U) \rightarrow A+1$
21	0-17	ANU	Add Negative Upper	$(A) - (U) \rightarrow A+1$
22	0-15	BT	Block Transfer	$(Xx) + u \rightarrow Xa + u$ ; repeat k times
23	0-17	L,LR	Load R	$(U) \rightarrow Ra$
24	0-17	A,AX	Add to X	$(Xa) + (U) \rightarrow Xa$
25	0-17	AN,ANX	Add Negative to X	$(Xa) - (U) \rightarrow Xa$
26	0-17	LXM	Load X Modifier	$(U) \rightarrow Xa_{17-0}$ ; $Xa_{35-18}$ unchanged
27	0-17	L,LX	Load X	$(U) \rightarrow Xa$
30	0-17	MI	Multiply Integer	$(A) \times (U) \rightarrow A, A+1$
31	0-17	MSI	Multiply Single Integer	$(A) \times (U) \rightarrow A$
32	0-17	MF	Multiply Fractional	$(A) \times (U) \rightarrow A, A+1$ , left circular one bit
33	00	BM	Byte Move	Transfer LJO bytes from source string to receiving string. Truncate or fill receiving



Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
33	01	BMT	Byte Move With Translate	string as required Translated and transfer LJ0 bytes from source string to receiving string. Truncate or fill receiving string as required
33	03	BTC	Byte Translate and Compare	Translate and compare LJ0 bytes from string SJ0 to LJ1 bytes from string SJ1; terminate instruction on not equal or if both LJ0 and LJ1 are zero, when: (Aa) + ; string SJ0 > SJ1 (Aa) 0 ; string SJ0 = SJ1 (Aa) - ; string SJ0 < SJ1
33	04	BC	Byte Compare	Compare LJ0 bytes from string SJ0 to LJ1 bytes from string SJ1; terminate instruction on not equal or if both LJ0 and LJ1 are zero
33	07	EDIT	Edit	Edit string SJ0 and transfer to string SJ1 under the control of string SJ2
33	10	BI	Byte to Binary Single Integer Convert	Convert LJ0 bytes in string SJ0 to a signed binary integer in register A
33	11	BDI	Byte to Binary Double Integer Convert	Convert LJ0 bytes in string SJ0 to a signed binary integer in registers A and A+1
33	12	IB	Binary Single Integer to Byte Convert	Convert signed binary integer in A to byte format and store in string SJ0
33	13	DIB	Binary Double Integer to Byte Convert	Convert the binary integer in A and A+1 to byte format and store in string SJ0
33	14	BF	Byte to Single Floating Convert	Convert LJ0 bytes in string SJ0 to a single length floating point format in register A
33	15	BDF	Byte to Double Floating Convert	Convert LJ0 bytes in string SJ0 to a double length floating point format in registers A and A+1
33	16	FB	Single Floating to Byte Convert	Convert the single length floating point number in A to byte format and store in string SJ0

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
33	17	DFB	Double Floating to Byte Convert	Convert double length floating point number in A and A+1 to byte format and store in string SJ0
34	0-17	DI	Divide Integer	(A, A+1) divided by (U) → A; REMAINDER → A+1
35	0-17	DSF	Divide Single Fractional	[(A, 36 sign bits) right algebraic shift 1 place] divided by (U) → A+1
36	0-17	DF	Divide Fractional	[(A, A+1) right algebraic shift 1 place] divided by (U) → A; REMAINDER → A+1
37	06	BA	Byte Add	Add the LJ0 bytes in string SJ0 to the LJ1 bytes in string SJ1 and store the results in string SJ2
37	07	BAN	Byte Add Negative	Subtract the LJ0 bytes in string SJ0 from the LJ1 bytes in string SJ1 and store the results in string SJ2
40	0-17	OR	Logical OR	(A) $\boxed{\text{OR}}$ (U) → A+1
41	0-17	XOR	Logical Exclusive OR	(A) $\boxed{\text{XOR}}$ (U) → A+1
42	0-17	AND	Logical AND	(A) $\boxed{\text{AND}}$ (U) → A+1
43	0-17	MLU	Masked Load Upper	[(U) $\boxed{\text{AND}}$ (R2)] $\boxed{\text{OR}}$ [(A) AND NOT (R2)] → A+1
44	0-17	TEP	Test Even Parity	Skip NI if (U) $\boxed{\text{AND}}$ (A) has even parity
45	0-17	TOP	Test Odd Parity	Skip NI if (U) $\boxed{\text{AND}}$ (A) has odd parity
46	0-17	LXI	Load X Increment	(U) → (Xa) <sub>35-18</sub> ; (Xa) <sub>17-0</sub> unchanged
47	0-17	TLEM	Test Less Than or Equal to Modifier	Skip NI if (U) <sub>17-0</sub> ≤ (Xa) <sub>17-0</sub> ; always (Xa) <sub>17-0</sub> + (Xa) <sub>35-18</sub> → Xa <sub>17-0</sub>
		TNGM	Test Not Greater Than Modifier	
50	0-17	TZ	Test Zero	Skip NI if (U) = ± 0

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
51	0-17	TNZ	Test Nonzero	Skip NI if $(U) \neq \pm 0$
52	0-17	TE	Test Equal	Skip NI if $(U) = (A)$
53	0-17	TNE	Test Not Equal	Skip NI if $(U) \neq (A)$
54	0-17	TLE	Test Less Than or Equal	Skip NI if $(U) \leq (A)$
		TNG	Test Not Greater	
55	0-17	TG	Test Greater	Skip NI if $(U) > (A)$
56	0-17	TW	Test Within Range	Skip NI if $(A) < (U) \leq (A+1)$
57	0-17	TNW	Test Not Within Range	Skip NI if $(U) \leq (A)$ or $(U) > (A+1)$
60	0-17	TP	Test Positive	Skip NI if $(U)_{35} = 0$
61	0-17	TN	Test Negative	Skip NI if $(U)_{35} = 1$
62	0-17	SE	Search Equal	Skip NI if $(U) = (A)$ , else repeat
63	0-17	SNE	Search Not Equal	Skip NI if $(U) \neq (A)$ , else repeat
64	0-17	SLE	Search Less Than or Equal	Skip NI if $(U) \leq (A)$ , else repeat
		SNG	Search Not Greater	
65	0-17	SG	Search Greater	Skip NI if $(U) > (A)$ , else repeat
66	0-17	SW	Search Within Range	Skip NI if $(A) < (U) \leq (A+1)$ , else repeat
67	0-17	SNW	Search Not Within Range	Skip NI if $(U) \leq (A)$ or $(U) > (A+1)$ , else repeat
70		JGD	Jump Greater And Decrement	Jump to U if $(\text{Control Register})_{ja} > 0$ ; go to NI if $(\text{Control Register})_{ja} \leq 0$ ; always $(\text{Control Register})_{ja} - 1 \rightarrow \text{Control Register}_{ja}$
71	00	MSE	Mask Search Equal	Skip NI if $(U) \text{ AND } (R2) = (A) \text{ AND } (R2)$ , else repeat

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
71	01	MSNE	Mask Search Not Equal	Skip NI if (U) $\overline{\text{AND}}$ (R2) $\neq$ (A) $\overline{\text{AND}}$ (R2), else repeat
71	02	MSLE	Mask Search Less Than or Equal	Skip NI if (U) $\overline{\text{AND}}$ (R2) $\leq$ (A) $\overline{\text{AND}}$ (R2), else repeat
		MSNG	Mask Search Not Greater	
71	03	MSG	Mask Search Greater	Skip NI if (U) $\overline{\text{AND}}$ (R2) $>$ (A) $\overline{\text{AND}}$ (R2), else repeat
71	04	MSW	Masked Search Within Range	Skip NI if (A) $\overline{\text{AND}}$ (R2) $<$ (U) $\overline{\text{AND}}$ (R2) $\leq$ (A+1) $\overline{\text{AND}}$ (R2), else repeat
71	05	MSNW	Masked Search Not Within Range	Skip NI if (U) $\overline{\text{AND}}$ (R2) $\leq$ (A) $\overline{\text{AND}}$ (R2) or (U) $\overline{\text{AND}}$ (R2) $>$ (A+1) $\overline{\text{AND}}$ (R2), else repeat
71	06	MASL	Masked Alphanumeric Search Less Than or Equal	Skip NI if (U) $\overline{\text{AND}}$ (R2) $\leq$ (A) $\overline{\text{AND}}$ (R2), else repeat
71	07	MASG	Masked Alphanumeric Search Greater	Skip NI if (U) $\overline{\text{AND}}$ (R2) $>$ (A) $\overline{\text{AND}}$ (R2), else repeat
71	10	DA	Double-Precision Fixed-Point Add	(A, A+1) + (U, U+1) $\rightarrow$ A, A+1
71	11	DAN	Double-Precision Fixed-Point Add Negative	(A, A+1) - (U, U+1) $\rightarrow$ A, A+1
71	12	DS	Double Store A	(A, A+1) $\rightarrow$ U, U+1
71	13	DL	Double Load A	(U, U+1) $\rightarrow$ A, A+1
71	14	DLN	Double Load Negative A	- (U, U+1) $\rightarrow$ A, A+1
71	15	DLM	Double Load Magnitude A	(U, U+1)   $\rightarrow$ A, A+1
71	16	DJZ	Double-Precision Jump Zero	Jump to U if (A, A+1) = $\pm$ 0; go to NI if (A, A+1) $\neq$ $\pm$ 0

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
71	17	DTE	Double-Precision Test Equal	Skip NI if $(U < U+1) = (A, A+1)$
72	00	IMI	Initiate Maintenance Interrupt	Send Attention Interrupt to Maintenance Processor. If in Maintenance Mode, otherwise NO-OP
72	01	SLJ	Store Location And Jump	Relative $P+1 \rightarrow U_{17-0}$ ; jump to $U+1$
72	02	JPS	Jump Positive And Shift	Jump to $U$ if $(A)_{35} = 0$ ; go to NI if $(A)_{35} = 1$ ; always shift $(A)$ left circularly one bit position
72	03	JNS	Jump Negative And Shift	Jump to $U$ if $(A)_{35} = 1$ ; go to NI if $(A)_{35} = 0$ ; always shift $(A)$ left circularly one bit position
72	04	AH	Add Halves	$(A)_{35-18} + (U)_{35-18} \rightarrow (A)_{35-18}$ ; $(A)_{17-0} + (U)_{17-0} \rightarrow A_{17-0}$
72	05	ANH	Add Negative Halves	$(A)_{35-18} - (U)_{35-18} \rightarrow (A)_{35-18}$ ; $(A)_{17-0} - (U)_{17-0} \rightarrow A_{17-0}$
72	06	AT	Add Thirds	$(A)_{35-24} + (U)_{35-24} \rightarrow A_{35-24}$ ; $(A)_{23-12} + (U)_{23-12} \rightarrow A_{23-12}$ ; $(A)_{11-0} + (U)_{11-0} \rightarrow A_{11-0}$
72	07	ANT	Add Negative Thirds	$(A)_{35-24} - (U)_{35-24} \rightarrow A_{35-24}$ ; $(A)_{23-12} - (U)_{23-12} \rightarrow A_{23-12}$ ; $(A)_{11-0} - (U)_{11-0} \rightarrow A_{11-0}$
72	10	EX	Execute	Execute the instruction at $U$
72	11	ER	Executive Request	Interrupt to $MSR + 242_8$
72	13	PAIJ	Prevent All I/O Interrupts And Jump	Prevent all I/O interrupts and jump to $U$
72	15	TRA	Test Relative Address	Used to determine whether a relative address is within a given relative addressing range
72	16	SRS	Store Register Set	$A_a$ contains address and count for each of two GRS areas
72	17	LRS	Load Register Set	Move specified storage area to GRS area(s)

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
73	00	SSC	Single Shift Circular	Shift (A) right circularly U places
73	01	DSC	Double Shift Circular	Shift (A, A+1) right circularly U places
73	02	SSL	Single Shift Logical	Shift (A) right U places, zero fill
73	03	DSL	Double Shift Logical	Shift (A, A+1) right U places, zero fill
73	04	SSA	Single Shift Algebraic	Shift (A) right U places, sign fill
73	05	DSA	Double Shift Algebraic	Shift (A, A+1) right U places, sign fill
73	06	LSC	Load Shift And Count	(U) → A; shift (A) left circularly until (A) <sub>35</sub> ≠ (A) <sub>34</sub> ; number of shifts → A+1
73	07	DLSC	Double Load Shift and Count	(U, U+1) → A, A+1; shift (A, A+1) left circularly until (A, A+1) <sub>71</sub> ≠ (A, A+1) <sub>70</sub> ; number of shifts → A+2
73	10	LSSC	Left Single Shift Circular	Shift (A) left circularly U places
73	11	LDSC	Left Double Shift Circular	Shift (A, A+1) left circularly U places
73	12	LSSL	Left Single Shift Logical	Shift (A) left U places, zero fill
73	13	LDL	Left Double Shift Logical	Shift (A, A+1) left U places, zero fill
73	14	EDC a = 11	Enable Day Clock	Enable dayclock in IOAU having channels 0-23
73	14	DDC a = 12	Disable Day Clock	Disable dayclock
73	14	SDC a = 13	Select Day Clock	Select internal dayclock

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
73	15	SIL a = 00	Select Interrupt Locations	(U) <sub>8-0</sub> → MSR
73	15	LBX a = 02	Load Breakpoint Register	Transfer operand to Breakpoint Register
73	15	LQT a = 03	Load Quantum Timer	Place full-word operand in Quantum Timer
73	15	IIIX a = 04	Initiate Interprocessor Interrupt	Interrupt processor specified by operand address value
73	15	SPID a = 05	Store Processor ID	Store: binary serial number in first third; 2-character Fielddata revision level in second third; processor in last sixth of operand
73	15	RAT a = 06	Reset Auto-Recovery Timer	Reset auto-recovery timer in system transition unit
73	15	TAP a = 07	Toggle Auto-Recover Path	Toggle path selection after each auto-recovery attempt
73	15	LB a = 10	Load Base	Place operand bits 0 through 17 in base value field of BDR specified by bits 33 and 34 of Xx
73	15	LL a = 11	Load Limits	Place operand bits 15 through 23 and 24 through 35 in BDR limits fields specified by Xx bits 33 and 34
73	15	LAE a = 12	Load Addressing Environment	Place the double-word operand in GRS location 046 and 047 and the four respective Bank Descriptor Registers
73	15	SQT a = 13	Store Quantum Time	Store Quantum Timer value at GRS location 050 or at the storage location specified by operand address. Executing this instruction has no effect on D29
73	15	LD a = 14	Load Designator Register	Place full-word operand in Designator Register

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
73	15	SD a = 15	Store Designator Register	Store Designator Register contents at location specified by operand address
73	15	UR a = 16	User Return	Provides an orderly return to a user program
73	15	SSS a = 17	Store System Status	Store two system status words at the location specified by operand address
73	16	LCR a = 00	Load Channel Select Register	(U) <sub>5-0</sub> → CSR; if (U) <sub>9</sub> = 1, select back-to-back transfer mode
73	17	TS a = 00	Test And Set	If (U) <sub>30</sub> = 1, interrupt to MSR + 244 <sub>8</sub> ; if (U) <sub>30</sub> = 0, go to NI; always 01 <sub>8</sub> → U <sub>35-0</sub>
73	17	TSS a = 01	Test And Set And Skip	if (U) <sub>30</sub> = 0, skip NI; if (U) <sub>30</sub> = 1, go to NI; always 01 <sub>8</sub> → U <sub>35-30</sub>
73	17	TCS a = 02	Test and Clear And Skip	If (U) <sub>30</sub> = 0, go to NI; if (U) <sub>30</sub> = 1, skip NI; always clear (U) <sub>35-30</sub>
73	17	TSA	Test and Set Alternate	Test bit position 14; if (U) <sub>14</sub> = 1, interrupt; if (U) <sub>14</sub> = 0, take next instruction and set bits 00 through 14 to one
73	17	TSSA	Test and Set and Skip Alternate	If (U) <sub>14</sub> = 1, take next instruction; if (U) <sub>14</sub> = 0, skip the next instruction and set bits 00 through 14 to one
74	00	JZ	Jump Zero	Jump to U if (A) = ± 0 go to NI if (A) ≠ ± 0
74	01	JNZ	Jump Nonzero	Jump to U if (A) ≠ ± 0; go to NI if (A) = ± 0
74	02	JP	Jump Positive	Jump to U if (A) <sub>35</sub> = 0; go to NI if (A) <sub>35</sub> = 1
74	03	JN	Jump Negative	Jump to U if (A) <sub>35</sub> = 1; go to NI if (A) <sub>35</sub> = 0



Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
74	04	J JK	Jump Jump Key	Jump to U if a = 0 or if a = set SELECT JUMPS control circuit n go to NI if neither is true
74	05	HJ HKJ	Halt Jump Halt Keys And Jump	Stop if a = 0 or if [a field <b>AND</b> set SELECT STOPS control circuits] ≠ 0; on restart or continuation jump to U
74	06	NOP	No Operation	Proceed to next Instruction
74	07	AAIJ	All All I/O Interrupts And Jump	Allow all I/O interrupts and jump to U
74	10	JNB	Jump No Low Bit	Jump to U if (A) <sub>0</sub> = 0; go to NI if (A) <sub>0</sub> = 1
74	11	JB	Jump Low Bit	Jump to U if (A) <sub>0</sub> = 1; go to NI if (A) <sub>0</sub> = 0
74	12	JMGI	Jump Modifier Greater and Increment	Jump to U if (Xa) <sub>17-0</sub> > 0; go to NI if (Xa) <sub>17-0</sub> ≤ 0; always (Xa) <sub>17-0</sub> + (Xa) <sub>35-18</sub> → Xa <sub>17-0</sub>
74	13	LMJ	Load Modifier and Jump	Relative P + 1 → (Xa) <sub>17-0</sub> ; jump to U
74	14	JO a = 00	Jump Overflow	Jump to U if D1 = 1; go to NI if D1 = 0
74	14	JFU a = 01	Jump Floating Underflow	Jump to U if D21 = 1, clear D21; go to NI if D21 = 0
74	14	JFO a = 02	Jump Floating Overflow	Jump to U if D22 = 1, clear D22; go to NI if D22 = 0
74	14	JDF a = 03	Jump Divide Fault	Jump to U if D23 = 1, clear D23; go to NI if D23 = 0
74	15	JNO a = 00	Jump No Overflow	Jump to U if D1 = 0; go to NI if D1 = 1
74	15	JNFU a = 01	Jump No Floating Underflow	Jump to U if D21 = 0; go to NI if D21 = 1; clear D21

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
74	15	JNFO a = 02	Jump No Floating Overflow	Jump to U if D22 = 0; go to NI if D22 = 1; clear D22
74	15	JNDF a = 03	Jump No Divide Fault	Jump to U if D23 = 0; go to NI if D23 = 1; clear D23
74	16	JC	Jump Carry	Jump to U if D0 = 1; go to NI if D0 = 0
74	17	JNC	Jump No Carry	Jump to U if D0 = 0; go to NI if D0 = 1
75	00	SRL	Select Release	Initiates the execution of a CCW list
75	01	SIOF	Start I/O Fast Release	Initiates operation specified by bit 00 through 15 of CAW
75	02	TIO	Test I/O	Interrogates the channel, subchannel and device
75	03	TSC	Test Subchannel	Interrogates the channel and subchannel
75	04	HDV	Halt Device	Terminates current operation on channel and subchannel
75	05	HCH	Halt Channel	Terminates current operation on channel
75	10	LCR	Load Channel Register	Load the interrupt mask register
75	11	LTCW	Load Control Words	Loads the status table subchannel
76	00	FA	Floating Add	(A) + (U) → A; RESIDUE → A+1 if D17 = 1
76	01	FAN	Floating Add Negative	(A) - (U) → A; RESIDUE → A+1 if D17 = 1
76	02	FM	Floating Multiply	(A) x (U) → A (and A+1 if D17 = 1)
76	03	FD	Floating Divide	(A) divided by (U) → A; REMAINDER → A+1 if D17 = 1
76	04	LUF	Load and Unpack Floating	(U) <sub>34-27</sub> → A <sub>7-0</sub> , zero fill (U) <sub>26-00</sub> → A+1 <sub>26-00</sub> , sign fill (U) <sub>35</sub> → A+1 <sub>35</sub>

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
76	05	LCF	Load and Convert To Floating	$(U)_{35} \rightarrow A+1_{35}$ ; [NORMALIZED $(U)_{26-0} \rightarrow A+1_{26-0}$ ; if $(U)_{35} = 0$ , $(A)_{7-0} \pm$ NORMALIZING COUNT $\rightarrow A+1_{34-27}$ ; if $(U)_{35} = 1$ , ones complement of $[(A)_{7-0} \pm$ NORMALIZING COUNT] $\rightarrow A+1_{34-27}$
76	06	MCDU	Magnitude of Characteristic Difference To Upper	$ (A)_{35-27} - (U)_{35-27}  \rightarrow A+1_{8-0}$ ; ZEROS $\rightarrow A+1_{35-9}$
76	07	CDU	Characteristic Difference To Upper	$ (A)_{35-27} - (U)_{35-27}  \rightarrow A+1_{8-0}$ SIGN BITS $\rightarrow A+1_{35-9}$
76	10	DFA	Double-Precision Floating Add	$(A, A+1) + (U, U+1) \rightarrow A, A+1$
76	11	DFAN	Double-Precision Floating Add Negative	$(A, A+1) - (U, U+1) \rightarrow A, A+1$
76	12	DFM	Double-Precision Floating Multiply	$(A, A+1) \times (U, U+1) \rightarrow A, A+1$
76	13	DFD	Double-Precision Floating Divide	$(A, A+1)$ divided by $(U, U+1) \rightarrow A, A+1$
76	14	DFU	Double Load and Unpack Floating	$(U, U+1)_{70-60} \rightarrow A_{10-0}$ , zero fill; $(U, U+1)_{59-36} \rightarrow A+1_{23-0}$ , sign fill; $(U, U+1)_{35-0} \rightarrow A+2$
76	15	DLCF, DFP	Double Load and Convert To Floating	$(U)_{35} \rightarrow A+1_{35}$ ; [NORMALIZED $(U < U+1)_{59-0} \rightarrow A+1_{23-0}$ and $A+2$ ; if $(U)_{35}$ , $(A)_{10-0} \pm$ NORMALIZING COUNT $\rightarrow A+1_{34-24}$ ; if $(U)_{35} = 1$ , ones complement of $[(A)_{10-0} \pm$ NORMALIZING COUNT] $\rightarrow A+1_{34-24}$
76	16	FEL	Floating Expand and Load	If $(U)_{35} = 0$ ; $(U)_{35-27} + 1600_8 \rightarrow A_{35-24}$  If $(U)_{35} = 1$ ; $(U)_{35-27} - 1600_8 \rightarrow A_{35-24}$ $(U)_{26-3} \rightarrow A_{23-0}$ ; $(U)_{2-0} \rightarrow A+1_{35-33}$ ; $(U)_{35} \rightarrow A+1_{32-0}$

Table C-2. Instruction Repertoire (continued)

Function Code (Octal)		Mnemonic	Instruction	Description
f	j			
76	17	FCL	Floating Compress and Load	If $(U)_{35} = 0$ ; $(U)_{35-24} - 1600_8 \rightarrow A_{35-27}$ ; if $(U)_{35} = 1$ ; $(U)_{35-24} + 1600_8 \rightarrow A_{35-27}$ $(U)_{23-0} \rightarrow A_{26-3}$ ; $(U+1)_{35-33} \rightarrow A_{2-0}$

## Appendix D. Character Codes

Table D-1. Fielddata To ASCII Code Conversion

Fielddata Code (Octal)	80-Column Card Code	High Speed Printer Symbol	ASCII	
			Octal Code	Symbol
00	7-8	@	100	@
01	12-5-8	[	133	[
02	11-5-8	]	135	]
03	12-7-8	#	43	#
04	11-7-8	Δ	136	^
05	(blank)	(space)	40	(space)
06	12-1	A	101	A
07	12-2	B	102	B
10	12-3	C	103	C
11	12-4	D	104	D
12	12-5	E	105	E
13	12-6	F	106	F
14	12-7	G	107	G
15	12-8	H	110	H
16	12-9	I	111	I
17	11-1	J	112	J
20	11-2	K	113	K
21	11-3	L	114	L
22	11-4	M	115	M
23	11-5	N	116	N
24	11-6	O	117	O
25	11-7	P	120	P
26	11-8	Q	121	Q
27	11-9	R	122	R

Table D-1. Fielddata To ASCII Code Conversion (continued)

Fielddata Code (Octal)	80-Column Card Code	High Speed Printer Symbol	ASCII	
			Octal Code	Symbol
30	0-2	S	123	S
31	0-3	T	124	T
32	0-4	U	125	U
33	0-5	V	126	V
34	0-6	W	127	W
35	0-7	X	130	X
36	0-8	Y	131	Y
37	0-9	Z	132	Z
40	12-4-8	)	51	)
41	11	-	55	-
42	12	+	53	+
43	12-6-8	<	74	<
44	3-8	=	75	=
45	6-8	>	76	>
46	2-8	&	46	&
47	11-3-8	¢	44	¢
50	11-4-8	*	52	*
51	0-4-8	(	50	(
52	0-5-8	%	45	%
53	5-8	:	72	:
54	12-0	?	77	?
55	11-0		41	
56	0-3-8	,(comma)	54	,(comma)
57	0-6-8	\	134	\
60	0	0	60	0
61	1	1	61	1
62	2	2	62	2
63	3	3	63	3
64	4	4	64	4
65	5	5	65	5
66	6	6	66	6
67	7	7	67	7
70	8	8	70	8
71	9	9	71	9
60	0	0	60	0
61	1	1	61	1
62	2	2	62	2
63	3	3	63	3
64	4	4	64	4

Table D-1. Fieldata To ASCII Code Conversion (continued)

Fieldata Code (Octal)	80-Column Card Code	High Speed Printer Symbol	ASCII	
			Octal Code	Symbol
65	5	5	65	5
66	6	6	66	6
67	7	7	67	7
70	8	8	70	8
71	9	9	71	9
72	4-8	'(apostrophe)	47	'(apostrophe)
73	11-6-8	;	73	;
74	0-1	/	57	/
75	12-3-8	.(period)	56	.(period)
76	0-7-8	□	42	"
77	0-2-8	z or stop	137	—
100		@	00	@
101		A	06	A
102		B	07	B
103		C	10	C
104		D	11	D
105		E	12	E
106		F	13	F
107		G	14	G
110		H	15	H
111		I	16	I
112		J	17	J
113		K	20	K
114		L	21	L
115		M	22	M
116		N	23	N
117		O	24	O
120		P	25	P
121		Q	26	Q
122		R	27	R
123		S	30	S
124		T	31	T
125		U	32	U
126		V	33	V
127		W	34	W
130		X	35	X
131		Y	36	Y
132		Z	37	Z
133		[	01	[
134		\	57	\
135		]	60	]
136		-	04	-
137		.	77	.
140			00	@
141		a*	06	A**
through		through	through	through
172		z*	37	Z**

Table D-1. Fielddata To ASCII Code Conversion (continued)

Fielddata Code (Octal)	80-Column Card Code	High Speed Printer Symbol	ASCII	
			Octal Code	Symbol
173			54	
174			57	
175			55	
176			04	
177			77	

- \* Lower case alphabet
- \*\* Upper case alphabet

Codes, which also represent collating sequence, are given in octal.

ASCII codes from  $00_8$  to  $37_8$  are for communications, format, and separator control characters. These are not converted into Fielddata.

The ASCII symbols represented by codes  $40_8$  to  $137_8$  are converted into the identical Fielddata symbols, except that the quotation marks symbol ( $42_8$ ) is converted into a lozenge ( $76_8$ ), the circumflex ( $136_8$ ) is converted into a delta ( $04_8$ ), underscore ( $137_8$ ) is converted into a not equal sign ( $77_8$ ).

There are no remaining unique Fielddata symbols into which to convert the balance of the ASCII symbols, represented by codes  $140_8$  to  $177_8$ , so most of these codes are "folded" over codes  $100_8$  to  $137_8$  (by clearing bit 5, which amounts to subtracting  $40_8$ ). This means that ASCII codes  $101_8$  (A) and  $141_8$  (a), for example, are both translated as if they were code  $101_8$  (converted to Fielddata  $06_8$  for A). Two exceptions to this general rule are the ASCII opening brace ( $173_8$ ) and closing brace ( $175_8$ ) which are converted to Fielddata question mark ( $54_8$ ) and exclamation point ( $55_8$ ), respectively, to satisfy overpunch sign considerations. This is the conversion provided by the Fielddata/ASCII translator (F1325-00), as explained in the UNIVAC 1100 Series Multi-Subsystem Adapter Programmer Reference, UP-7890 (current version).

#### The Special Characters In ASCII

- SP designates space, which is normally nonprinting.
- DEL designates delete, and has a code of all 1 bits. This code obliterates any unwanted previous character – even on paper tape or other nonerasable medium.

#### Definitions of the 32 ASCII control characters, codes $00_8$ to $37_8$ :

- 00 NUL Null – all zero character which may serve as time fill
- 01 SOH Start of heading
- 02 STX Start of text
- 03 ETX End of text
- 04 EOT End of transmission
- 05 ENQ Enquire – "Who Are You?"
- 06 ACK Acknowledge – "Yes"
- 07 BEL Bell – human attention required



- |    |     |   |   |  |
|----|-----|---|---|--|
| 10 | BS  | Backspace   | } | format effectors for<br>printing or punching   |
| 11 | HT  | Horizontal tabulation   |   |  |
| 12 | LF  | Line feed   |   |  |
| 13 | VT  | Vertical tabulation   |   |  |
| 14 | FF  | Form feed   |   |  |
| 15 | CR  | Carriage return   | } |  |
| 16 | SO  | Shift out – nonstandard code follows                                |   |  |
| 17 | SI  | Shift in – return to standard code                                  |   |  |
| 20 | DLE | Data link escape – change limited data communication control        |   |  |
| 21 | DC1 | } Device control for turning on or off ancillary devices            | } |  |
| 22 | DC2 |   |   |  |
| 23 | DC3 |   |   |  |
| 24 | DC4 |   |   |  |
| 25 | NAK | Negative acknowledge – "No"   |   |  |
| 26 | SYN | Synchronous idle – from which to achieve synchronism                |   |  |
| 27 | EBT | End of transmission block – relate to physical communication block  |   |  |
| 30 | CAN | Cancel previous data  |   |  |
| 31 | EM  | End of medium – end of used, or wanted portion of information       |   |  |
| 32 | SUB | Substitute character for one in error                               |   |  |
| 33 | ESC | Escape – for code extension – change some character interpretations |   |  |
| 34 | FS  | File separator  | } | These information separators are ordered in<br>descending hierarchy. They are followed by<br>ASCII 40 <sub>8</sub> (space), which can also be thought<br>of as a word separator. |
| 35 | GS  | Group separator   |   |  |
| 36 | RS  | Record separator  |   |  |
| 37 | US  | Unit separator  |   |  |

