

# SPECTRA 70

RADIO CORPORATION OF AMERICA • ELECTRONIC DATA PROCESSING



PROCESSORS

70 3455

REFERENCE MANUAL



RADIO CORPORATION OF AMERICA

70-35-601  
March 1966

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

First Printing: June, 1965

Revised: March, 1966

# CONTENTS

|   | Page |
|---|------|
| <b>INTRODUCTION</b>                       |      |
| RCA Model 70/35 Processor .....           | 1    |
| RCA Model 70/45 Processor .....           | 1    |
| RCA Model 70/55 Processor .....           | 2    |
| Organization of Data .....                | 2    |
| Data Formats .....                        | 2    |
| Numbering System .....                    | 3    |
| <b>SYSTEMS STRUCTURE</b>                  |      |
| Introduction .....                        | 4    |
| Main Memory .....                         | 4    |
| Non-Addressable Main Memory .....         | 5    |
| Scratch-Pad Memory .....                  | 5    |
| Program Control and Arithmetic Unit ..... | 6    |
| Input/Output Control .....                | 6    |
| Read-Only Memory .....                    | 7    |
| <b>INSTRUCTION FORMATS</b>                |      |
| RR Format .....                           | 8    |
| RX Format .....                           | 8    |
| RS Format .....                           | 8    |
| SI Format .....                           | 8    |
| SS Format .....                           | 8    |
| <b>ADDRESSING</b>                         | 10   |
| <b>PROGRAM INTERRUPT</b>                  |      |
| Introduction .....                        | 10   |
| Processor States .....                    | 10   |
| Processing State $P_1$ .....              | 10   |
| Interrupt Response State $P_2$ .....      | 10   |
| Interrupt Control State $P_3$ .....       | 11   |
| Machine Condition State $P_4$ .....       | 11   |
| Processor State Registers .....           | 11   |
| Program Counter .....                     | 11   |
| General Registers .....                   | 13   |
| Floating-Point Registers .....            | 13   |
| Interrupt Status Registers .....          | 13   |
| Interrupt Mask Registers .....            | 14   |
| Program Mask Registers .....              | 15   |
| Register Addressing .....                 | 15   |
| Interrupt Flag Register .....             | 16   |
| Interrupt Conditions .....                | 17   |
| Interrupt Mechanization .....             | 23   |
| Automatic Interrupt .....                 | 23   |
| Program Controlled Interrupt .....        | 26   |
| <b>INPUT/OUTPUT OPERATION</b>             |      |
| Introduction .....                        | 30   |
| Input/Output Channels .....               | 30   |
| Selector Channels .....                   | 30   |
| Multiplexor Channel .....                 | 31   |

## CONTENTS (Cont'd)

|   | Page |
|---|------|
| <b>INPUT/OUTPUT OPERATION</b><br>(Cont'd)                     |      |
| Input/Output Operational Control .....                        | 32   |
| Programming Considerations prior to Input/Output Initiation.. | 32   |
| Input/Output Initiation .....                                 | 32   |
| Channel Servicing .....                                       | 32   |
| Channel Address Word .....                                    | 34   |
| Channel Command Word .....                                    | 34   |
| Input/Output Channel Registers .....                          | 38   |
| Channel Address Register, CAR .....                           | 39   |
| Channel Command Register-I, CCR-I .....                       | 39   |
| Channel Command Register-II, CCR-II .....                     | 40   |
| Assembly/Status Register .....                                | 40   |
| Input/Output Instructions .....                               | 41   |
| Start Device Instruction .....                                | 41   |
| Halt Device Instruction .....                                 | 46   |
| Test Device Instruction .....                                 | 51   |
| Check Channel Instruction .....                               | 53   |
| Input/Output Status Indicators .....                          | 55   |
| Condition Code .....  | 55   |
| Channel Status Byte .....                                     | 58   |
| Standard Device Byte .....                                    | 60   |
| Sense Bytes .....   | 61   |
| Channel Servicing .....                                       | 61   |
| Servicing a Data Transfer .....                               | 61   |
| End and Chaining Servicing .....                              | 67   |
| Interrupt Servicing .....                                     | 72   |
| <br><b>MULTI-PROCESSOR INSTALLATION</b>                       |      |
| Introduction .....  | 76   |
| Operational Characteristics .....                             | 76   |
| Direct Control Interface .....                                | 77   |
| Static Out Lines .....  | 77   |
| Static In Lines .....   | 77   |
| Signal Out Line .....   | 77   |
| External Signal Line .....                                    | 77   |
| Power Failure Line (PFND) .....                               | 77   |
| Power Failure Inhibit In Line (PFIR) .....                    | 77   |
| Dual Processor Complex .....                                  | 78   |
| Master/Satellite Complex .....                                | 79   |
| Maximum Multi-Processor Complex .....                         | 80   |
| Operational Procedures .....                                  | 81   |
| Transmission Procedure .....                                  | 81   |
| Response Procedure .....                                      | 81   |
| <br><b>PRIVILEGED INSTRUCTIONS</b>                            |      |
| Introduction .....  | 83   |
| Instruction Formats .....                                     | 83   |
| Interrupt Action .....  | 83   |
| Special Considerations .....                                  | 84   |
| Load Scratch-Pad (LSP) .....                                  | 86   |
| Store Scratch-Pad (SSP) .....                                 | 87   |
| Program Control (PC) .....                                    | 88   |
| Idle (IDL) .....  | 90   |
| Diagnose (DIG) .....  | 91   |



## CONTENTS (Cont'd)

|   | <b>Page</b> |
|---|-------------|
| <b>PRIVILEGED INSTRUCTIONS</b><br>(Cont'd)      |             |
| Start Device (SDV) .....                        | 92          |
| Halt Device (HDV) .....                         | 95          |
| Test Device (TDV) .....                         | 97          |
| Check Channel (CKC) .....                       | 99          |
| Insert Storage Key (ISK) .....                  | 100         |
| Set Storage Key (SSK) .....                     | 101         |
| Write Direct (WRD) .....                        | 102         |
| Read Direct (RDD) .....                         | 103         |
| <br><b>PROCESSOR STATE CONTROL INSTRUCTIONS</b> |             |
| Introduction .....                              | 104         |
| Instruction Format .....                        | 104         |
| Condition Code Utilization .....                | 104         |
| Interrupt Action .....                          | 104         |
| Supervisor Call (SVC) .....                     | 105         |
| Set Program Mask (SPM) .....                    | 106         |
| <br><b>FIXED-POINT INSTRUCTIONS</b>             |             |
| Introduction .....                              | 107         |
| Data Format .....                               | 107         |
| Representation of Numbers .....                 | 107         |
| Instruction Formats .....                       | 108         |
| Condition Code Utilization .....                | 109         |
| Interrupt Action .....                          | 110         |
| Load Word (LR) (L) .....                        | 111         |
| Load Halfword (LH) .....                        | 112         |
| Load and Test (LTR) .....                       | 113         |
| Load Complement (LCR) .....                     | 114         |
| Load Positive (LPR) .....                       | 115         |
| Load Negative (LNR) .....                       | 116         |
| Load Multiple (LM) .....                        | 117         |
| Add Word (AR) (A) .....                         | 118         |
| Add Halfword (AH) .....                         | 119         |
| Add Logical (ALR) (AL) .....                    | 120         |
| Subtract Word (SR) (S) .....                    | 121         |
| Subtract Halfword (SH) .....                    | 122         |
| Subtract Logical (SLR) (SL) .....               | 123         |
| Compare Word (CR) (C) .....                     | 124         |
| Compare Halfword (CH) .....                     | 125         |
| Multiply Word (MR) (M) .....                    | 126         |
| Multiply Halfword (MH) .....                    | 127         |
| Divide (DR) (D) .....                           | 128         |
| Convert to Binary (CVB) .....                   | 129         |
| Convert to Decimal (CVD) .....                  | 130         |
| Store Word (ST) .....                           | 131         |
| Store Halfword (STH) .....                      | 132         |
| Store Multiple (STM) .....                      | 133         |
| Shift Left Single (SLA) .....                   | 134         |
| Shift Right Single (SRA) .....                  | 135         |
| Shift Left Double (SLDA) .....                  | 136         |
| Shift Right Double (SRDA) .....                 | 137         |

## CONTENTS (Cont'd)

|  | Page   |
|--|--|
| <b>DECIMAL<br/>ARITHMETIC<br/>INSTRUCTIONS</b> | Introduction ..... 138                           |
|  | Data Formats ..... 138                           |
|  | Representation of Numbers ..... 139              |
|  | Instruction Format ..... 139                     |
|  | Condition Code Utilization ..... 140             |
|  | Interrupt Action ..... 140                       |
|  | Add Decimal (AP) ..... 142                       |
|  | Subtract Decimal (SP) ..... 143                  |
|  | Zero and Add (ZAP) ..... 144                     |
|  | Compare Decimal (CP) ..... 145                   |
|  | Multiply Decimal (MP) ..... 146                  |
|  | Divide Decimal (DP) ..... 147                    |
|  | Pack (PACK) ..... 148                            |
|  | Unpack (UNPK) ..... 149                          |
|  | Move with Offset (MVO) ..... 150                 |
| <b>LOGICAL<br/>INSTRUCTIONS</b>                | Introduction ..... 151                           |
|  | Data Format ..... 151                            |
|  | Instruction Formats ..... 152                    |
|  | Condition Code Utilization ..... 153             |
|  | Interrupt Action ..... 153                       |
|  | Move (MVI) (MVC) ..... 154                       |
|  | Move Numerics (MVN) ..... 155                    |
|  | Move Zones (MVZ) ..... 156                       |
|  | Compare Logical (CLR) (CL) (CLI) (CLC) ..... 157 |
|  | AND (NR) (N) (NI) (NC) ..... 158                 |
|  | OR (OR) (O) (OI) (OC) ..... 159                  |
|  | Exclusive OR (XR) (X) (XI) (XC) ..... 160        |
|  | Test Under Mask (TM) ..... 161                   |
|  | Insert Character (IC) ..... 162                  |
|  | Store Character (STC) ..... 163                  |
|  | Load Address (LA) ..... 164                      |
|  | Translate (TR) ..... 165                         |
|  | Translate and Test (TRT) ..... 166               |
|  | Edit (ED) ..... 167                              |
|  | Edit and Mark (EDMK) ..... 170                   |
|  | Shift Left Single Logical (SLL) ..... 172        |
|  | Shift Right Single Logical (SRL) ..... 173       |
|  | Shift Left Double Logical (SLDL) ..... 174       |
|  | Shift Right Double Logical (SRDL) ..... 175      |
|  | <b>BRANCHING<br/>INSTRUCTIONS</b>                |
| Sequential Execution ..... 176                 |  |
| Instruction Formats ..... 176                  |  |
| Interrupt Action ..... 177                     |  |
| Branch on Condition (BCR) (BC) ..... 178       |  |
| Branch and Link (BALR) (BAL) ..... 179         |  |
| Branch on Count (BCTR) (BCT) ..... 180         |  |
| Branch on Index High (BXH) ..... 181           |  |
| Branch on Index Low or Equal (BXLE) ..... 182  |  |
| Execute (EX) ..... 183                         |  |

## CONTENTS (Cont'd)

|  | Page |
|--|------|
| <b>FLOATING-POINT INSTRUCTIONS</b>                             |      |
| Introduction .....   | 184  |
| Data Formats .....   | 184  |
| Representation of Numbers .....                                | 185  |
| Normalization .....  | 185  |
| Instruction Formats .....                                      | 185  |
| Condition Code Utilization .....                               | 186  |
| Interrupt Action .....   | 187  |
| Load (LER) (LE) (LDR) (LD) .....                               | 188  |
| Load and Test (LTER) (LTDR) .....                              | 189  |
| Load Complement (LCER) (LCDR) .....                            | 190  |
| Load Positive (LPER) (LPDR) .....                              | 191  |
| Load Negative (LNER) (LNDR) .....                              | 192  |
| Add Normalized (AER) (AE) (ADR) (AD) .....                     | 193  |
| Add Unnormalized (AUR) (AU) (AWR) (AW) .....                   | 195  |
| Subtract Normalized (SER) (SE) (SDR) (SD) .....                | 196  |
| Subtract Unnormalized (SUR) (SU) (SWR) (SW) .....              | 197  |
| Compare (CER) (CE) (CDR) (CD) .....                            | 198  |
| Halve (HER) (HDR) .....  | 199  |
| Store (STE) (STD) .....  | 200  |
| Multiply (MER) (ME) (MDR) (MD) .....                           | 201  |
| Divide (DER) (DE) (DDR) (DD) .....                             | 202  |
| <b>OPTIONAL FEATURES</b>                                       |      |
| Feature 5001 — Memory Protect .....                            | 203  |
| Feature 5002 — Elapsed Time Clock .....                        | 203  |
| Feature 5003 — Direct Control .....                            | 204  |
| Feature 5015 — Selector Channel .....                          | 204  |
| Feature 5016 — Selector Channel .....                          | 204  |
| Feature 5020 — Selector Channel .....                          | 204  |
| Feature 5022 — Selector Channel .....                          | 204  |
| Feature 5024 — Selector Channel .....                          | 204  |
| Feature 5030 — Selector Channel .....                          | 204  |
| Feature 5031 — Selector Channel .....                          | 204  |
| Emulator Options .....   | 204  |
| <b>APPENDICES</b>  |      |
| A — Summary of Instructions .....                              | 208  |
| B — Program Interrupts .....                                   | 228  |
| C — Input/Output Service Request .....                         | 230  |
| D — Extended Binary-Coded-Decimal Interchange Code .....       | 231  |
| E — American Standard Code for Information Interchange .....   | 232  |
| F — Character Codes .....                                      | 233  |
| G — Powers of Two Table .....                                  | 238  |
| H — Hexadecimal-Decimal Number Conversion .....                | 239  |
| I — Scratch-Pad Memory Layout and Register Assignments .....   | 247  |
| <b>LIST OF TABLES</b>  |      |
| Table 1. Basic Hexadecimal Marking System .....                | 4    |
| Table 2. 70/35-45-55 Memory Capacities .....                   | 5    |
| Table 3. Main Memory Capacity and Multiplexor Sets/Devices ... | 5    |
| Table 4. Use of General Registers .....                        | 9    |
| Table 5. Processor State Registers .....                       | 11   |
| Table 6. Instruction Length Codes .....                        | 11   |

## CONTENTS (Cont'd)

|  | <b>Page</b> |
|--|-------------|
| <b>LIST OF TABLES</b>  |             |
| <b>(Cont'd)</b>  |             |
| Table 7. Interrupt State Identifier Codes .....                | 13          |
| Table 8. Program Indicator Codes .....                         | 13          |
| Table 9. Register Addressing in Processor States .....         | 15          |
| Table 10. Interrupt Conditions and Priority .....              | 16          |
| Table 11. Interrupt Conditions .....                           | 18          |
| Table 12. Command Code Operations .....                        | 34          |
| Table 13. Input/Output Channel Registers .....                 | 39          |
| <br><b>LIST OF ILLUSTRATIONS</b>                               |             |
| Figure 1. Data Formats .....                                   | 2           |
| Figure 2. Functional Logic of Automatic Interrupt .....        | 24          |
| Figure 3. Functional Logic of Program Control Instruction .... | 27          |
| Figure 4. Functional Logic of Start Device Instruction .....   | 42          |
| Figure 5. Functional Logic of Halt Device Instruction .....    | 48          |
| Figure 6. Functional Logic of Test Device Instruction .....    | 52          |
| Figure 7. Functional Logic of Check Channel Instruction .....  | 54          |
| Figure 8. Functional Logic of Servicing a Data Transfer .....  | 62          |
| Figure 9. Functional Logic of End and Chaining Servicing ....  | 68          |
| Figure 10. Functional Logic of Interrupt Servicing .....       | 74          |
| Figure 11. Dual-Processor Complex .....                        | 78          |
| Figure 12. Master/Satellite Complex .....                      | 79          |
| Figure 13. Maximum Multi-Processor Complex .....               | 80          |

**Privileged Instructions**

**Processor State Control Instructions**

**Fixed-Point Instructions**

## **INSTRUCTION INDEX**

The index marks at the right edge of this page line up with similar index marks in the text. By merely examining the page edges, the reader can quickly locate a category of instructions.

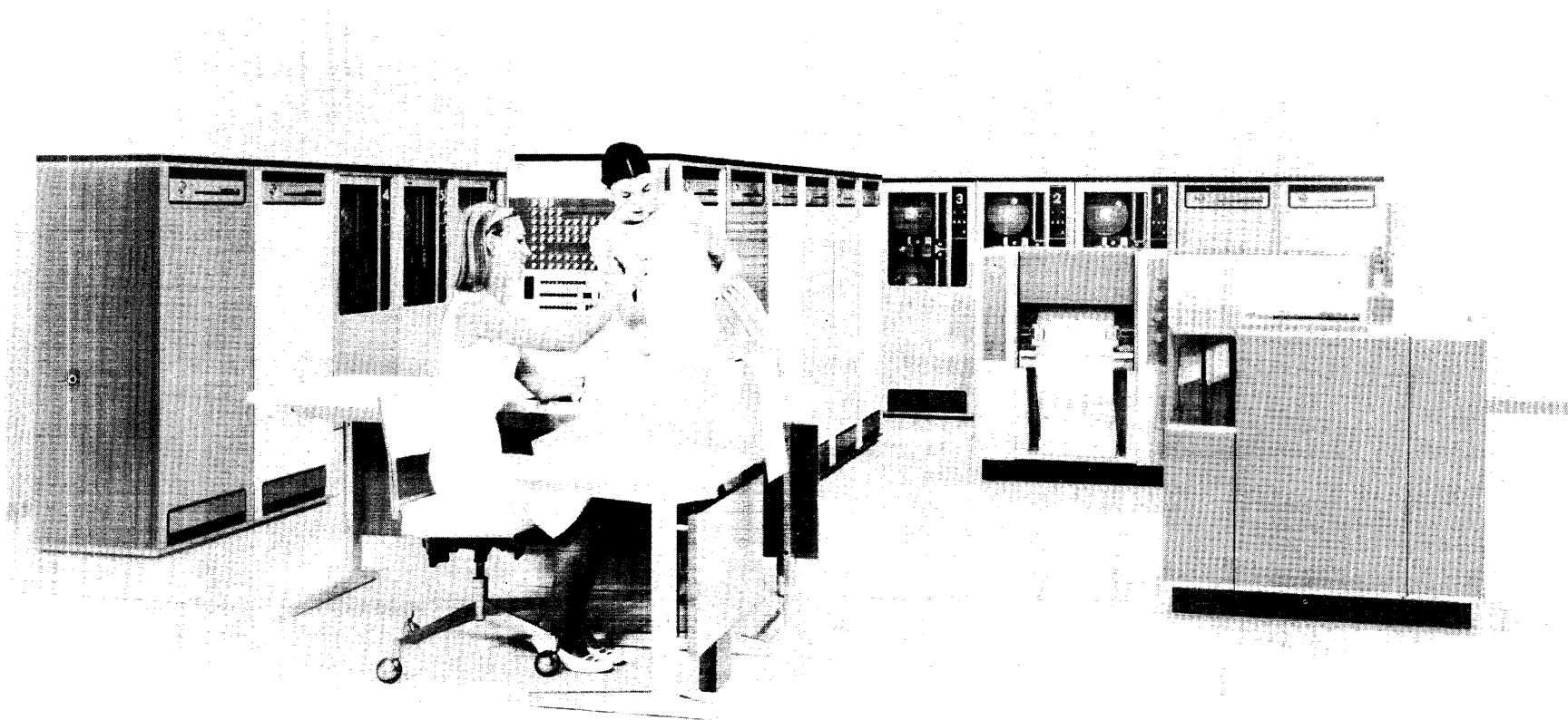
Appendix A summarizes the instruction set for the 70/45-55 Processors, including timing, formats and condition codes.

**Decimal Arithmetic Instructions**

**Logical Instructions**

**Branching Instructions**

**Floating-Point Instructions**



**Frontispiece**

## INTRODUCTION

### RCA MODEL 70/35 PROCESSOR

◆ The RCA Model 70/35 Processor is the small-scale member of the 70/45, 70/55 product line. It is a powerful, solid-state, general-purpose, digital processor. It is the main element of a system handling small to medium-large data processing applications. This processor is capable of handling commercial, scientific, and communications applications. The internal logic is controlled by microinstructions stored in a read-only control memory.

All instructions, character codes, formats, interrupt facilities, and programming features are functionally the same as corresponding features on the Model 70/45 and 70/55 Processors. Programs may be interchanged between processors provided:

1. Systems features are equivalent.
2. Programs are written to be independent of strict timing considerations.
3. Programs are restricted to specified functions and do not utilize unspecified characteristics peculiar to the hardware of any one of the processors.

The 70/35 is a variable-format processor consisting of main memory, read only control memory, non-addressable memory, program control, and input/output control. Internal logic processes one byte at a time. However, internal transmission paths are two bytes such that addresses and sometimes data are transferred two bytes at a time.

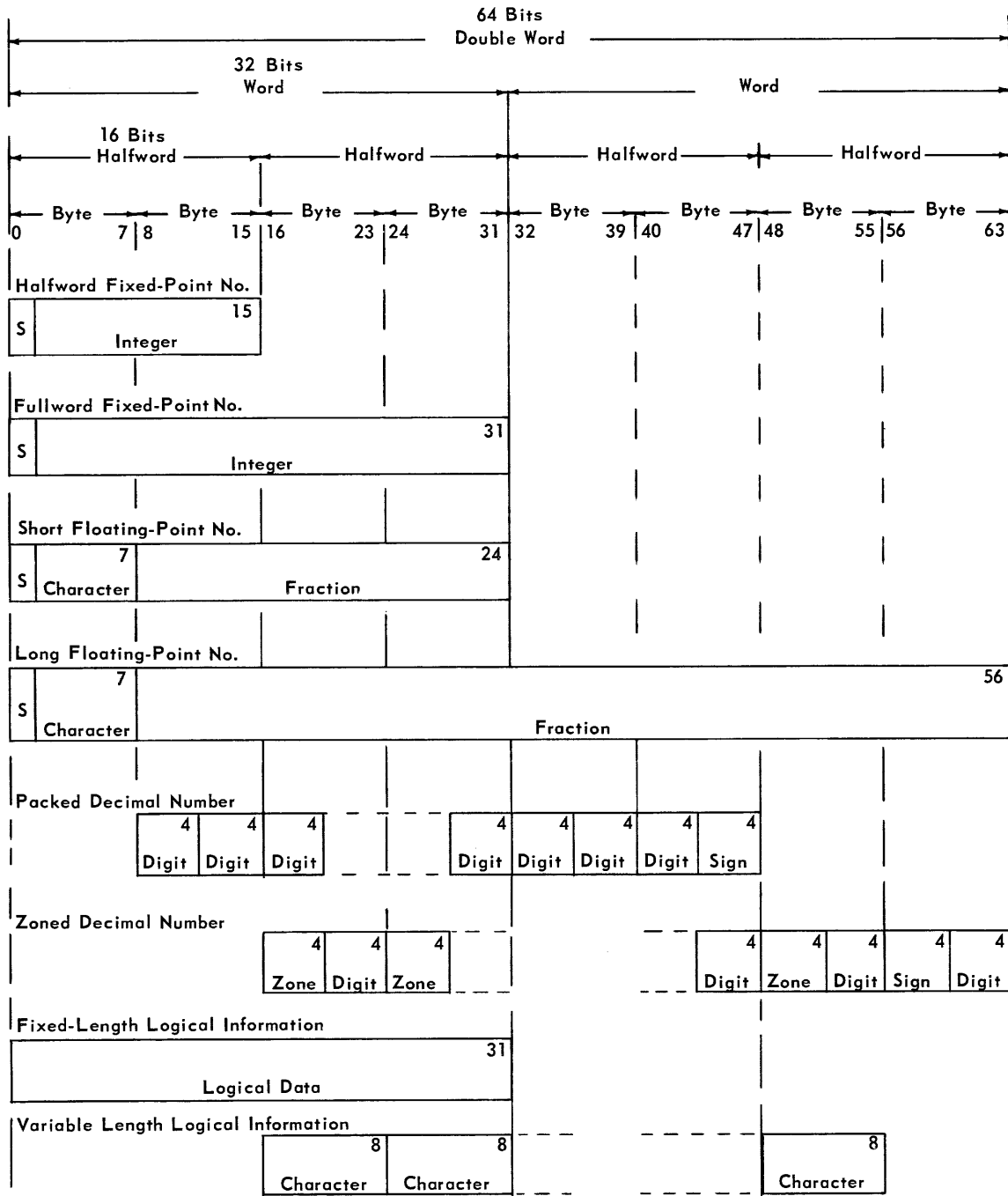
### RCA MODEL 70/45 PROCESSOR

◆ The RCA Model 70/45 Processor, a member of the Spectra 70 Series, is a powerful, solid-state, general-purpose, digital processor. It is the main element of a system that handles medium-large data processing applications. Because of its large storage capacity, fast data transmission, computation rates and communications capabilities, this processor is highly efficient as a data processor, a scientific problem solver, or a communications control processor. The internal logic is controlled by microinstructions stored in a read-only control memory.

All instructions, character codes, interrupt facilities, and programming features are functionally the same as corresponding features on the Model 70/35 and 70/55 Processors. Programs can be interchanged between processors provided that:

1. Systems features are equivalent.
2. Programs are written to be independent of strict timing considerations.
3. Programs are restricted to specified functions and do not use unspecified characteristics peculiar to the hardware of either processor.

The 70/45 is a halfword-organized, variable-format processor consisting of *main memory, non-addressable main memory, scratch-pad memory, read-only memory, program control and arithmetic unit, and input/output control.*



NOTE: Numbers in upper right corners of blocks indicate number of bits used.

Figure 1. Data Formats



**RCA MODEL 70/55  
PROCESSOR**

◆ The RCA Model 70/55 Processor, largest of the open-ended Spectra 70 Series, satisfies the most sophisticated data processing, scientific problem solving, or communications systems requirements. Its order code is implemented by processor logic, resulting in extremely fast data transmission and instruction execution rates.

All instructions, character codes, interrupt facilities, and programming features are functionally the same as corresponding features on the Model 70/35 and 70/45 Processors. Programs can be interchanged between the processors provided that:

1. Systems features are equivalent.
2. Programs are written to be independent of strict timing considerations.
3. Programs are restricted to specified functions and do not use unspecified characteristics peculiar to the hardware of either processor.

The 70/55 is a word-organized, variable-format processor consisting of *main memory, non-addressable main memory, scratch-pad memory, program control and arithmetic unit, and input/output control.*

**ORGANIZATION  
OF DATA**

|                   |   |
|-------------------|---|
| <b>Bit</b>        | ◆ A bit is a single binary digit having the value of either zero or one.  |
| <b>Byte</b>       | ◆ A byte consists of eight information bits. It represents two decimal digits, one alphabetic character, or one special symbol.                                     |
| <b>Halfword</b>   | ◆ A halfword consists of two consecutive bytes beginning on a main memory location that is a multiple of two.   |
| <b>Word</b>       | ◆ A word consists of four consecutive bytes beginning on a main memory location that is a multiple of four.   |
| <b>Doubleword</b> | ◆ A doubleword consists of eight consecutive bytes beginning on a main memory location that is a multiple of eight.   |
| <b>Item/Field</b> | ◆ An item/field consists of any number of bytes that specify a particular unit of information (numeric field, alphabetic name, street address, stock number, etc.). |
| <b>Record</b>     | ◆ A record consists of one or more related items.   |

**DATA FORMATS**

◆ The basic unit of information in the 70/35, 70/45 and 70/55 Processors is a byte, which is the smallest addressable unit. A byte consists of eight information bits. The parity bit ensures the accuracy of all bytes accessed by the processor. Odd parity is used in all processors.

The internal code representation in the 70/35, 70/45 and 70/55 is either the Extended Binary-Coded-Decimal Interchange Code (EBCDIC) or the American Standard Code for Information Interchange (ASCII) as specified by program. (See Appendices D and E.)

There are eight distinct formats for data in main memory (see figure 1). Further explanation of each format appears in the instruction sections of this manual.

## NUMBERING SYSTEM

◆ Since binary addresses are cumbersome to work with, the hexadecimal numbering system has been adopted to represent characters and addresses in the 70/35-45-55 Processors. The hexadecimal system has a base of 16. The first ten marks are represented by decimal numbers zero (0) through nine (9); the last six marks are represented by the letters A through F.

The basic hexadecimal marking system and its binary and decimal equivalent are specified in table 1. (See also Appendix H.)

**Table 1. Basic Hexadecimal Marking System**

| Hexadecimal<br>(Base 16) | Binary<br>(Base 2) | Decimal<br>(Base 10) | Hexadecimal<br>(Base 16) | Binary<br>(Base 2) | Decimal<br>(Base 10) |
|--------------------------|--------------------|----------------------|--------------------------|--------------------|----------------------|
| 0                        | 0000               | 0                    | 8                        | 1000               | 8                    |
| 1                        | 0001               | 1                    | 9                        | 1001               | 9                    |
| 2                        | 0010               | 2                    | A                        | 1010               | 10                   |
| 3                        | 0011               | 3                    | B                        | 1011               | 11                   |
| 4                        | 0100               | 4                    | C                        | 1100               | 12                   |
| 5                        | 0101               | 5                    | D                        | 1101               | 13                   |
| 6                        | 0110               | 6                    | E                        | 1110               | 14                   |
| 7                        | 0111               | 7                    | F                        | 1111               | 15                   |

## SYSTEMS STRUCTURE

### INTRODUCTION

◆ The RCA 70/35-45-55 Processors consist of main memory, non-addressable main memory, scratch-pad memory, (or equivalent scratch-pad memory in the 70/35), program control and arithmetic unit, and input/output control. In addition, the 70/35-45 Processors contain a read-only memory.

### MAIN MEMORY

◆ The main memory of the RCA 70/35-45-55 Processors is central storage for both data to be processed and the controlling instructions. Main memory consists of planes of magnetic cores, with each core representing one binary digit. The smallest addressable unit of information in main memory is one byte (eight bits).

The basic cycle time of these processors is the time required to access and transfer a halfword (70/35-45) or a full word (70/55) from main memory to the memory register and regenerate the information in main memory. For the 70/35-45 Processors, the memory cycle time is 1.44 microseconds; for the 70/55 Processor, the memory cycle time is 0.84 microseconds.

Table 2 indicates the various main memory capacities and corresponding model number for the three Processors.

# **MAIN MEMORY** *(Cont'd)*

**Table 2. 70/35-45-55 Memory Capacities**

| Model Number | Capacity (in Bytes) | Model Number | Capacity (in Bytes) | Model Number | Capacity (in Bytes) |
|--------------|---------------------|--------------|---------------------|--------------|---------------------|
| 70/35C       | 16,384              | 70/45D       | 32,768              | 70/55E       | 65,536              |
| 70/35D       | 32,768              | 70/45E       | 65,536              | 70/55F       | 131,072             |
| 70/35E       | 65,536              | 70/45F       | 131,072             | 70/55G       | 262,144             |
| 70/45C       | 16,384              | 70/45G       | 262,144             | 70/55H       | 524,288             |

The first 128 locations of main memory are reserved for processor use and must not be used by the program.

# **NON-ADDRESSABLE MAIN MEMORY**

◆ A non-addressable main memory, is in addition to main memory and cannot be addressed by programming. It contains the subchannel registers that control the operation of input/output devices on the multiplexor channel. A set of three 32-bit registers services each device on the multiplexor channel. The number of subchannel register sets and the number of devices that can be connected to the multiplexor channel are determined by the capacity of main memory, which is given in table 3.

**Table 3. Main Memory Capacity and Multiplexor Sets/Devices**

| Capacity of Main Memory (Bytes) | No. of Multiplexor Subchannel Register Sets/Devices |                |                |
|---------------------------------|---|----------------|----------------|
|                                 | 70/35   | 70/45          | 70/55          |
| 16,384                          | 64  | 64             | Not Applicable |
| 32,768                          | 192   | 128            | Not Applicable |
| 65,536                          | 192   | 256            | 256            |
| 131,072                         | Not Applicable                                      | 256            | 256            |
| 262,144                         | Not Applicable                                      | 256            | 256            |
| 524,288                         | Not Applicable                                      | Not Applicable | 256            |

# **SCRATCH-PAD MEMORY**

◆ The scratch-pad memory is a micromagnetic storage device consisting of 128 four-byte words, the access time of which is 300 nanoseconds. Each word in scratch-pad memory is uniquely addressed.

The following registers are contained in scratch-pad memory. (See also Appendix H.) :

1. *Processor Utility Registers* — All locations designated as processor utility registers are used by the processor for program control and cannot be used by the program.
2. *General Registers* — These locations are the general registers for each processor state. These registers are used by the program for base addressing, for indexing, or for storing operands.

*Note:* The RCA/35-45-55 Processors have four processor states that pertain to system and program interrupts (see page 9).

3. *Interrupt Mask Registers* — An Interrupt Mask register for each processor state permits or inhibits 32 interrupt conditions.

## SCRATCH-PAD MEMORY (Cont'd)

4. *Interrupt Status Registers* — An Interrupt Status register for each processor state stores interrupt identification information and operational control information. This register contains indications of the last state interrupted, the protection key, the decimal mode (ASCII or EBCDIC), the privileged mode bit, and the supervisor call identification.
5. *Program Counter* — A Program Counter for each processor state contains the main memory address of the next instruction to be executed, the condition code, the instruction length code, and the program mask.
6. *Input/Output Channel Registers* — A set of six registers for each selector channel controls input/output operation. A set of four registers for the multiplexor channel controls initiation and termination of input/output operations on the multiplexor channel.
7. *Floating-Point Registers* — Four floating-point registers (each is two words long) are used in floating-point arithmetic.
8. *Interrupt Flag Register* — One Interrupt Flag register is provided. When an interrupt condition occurs, a bit associated with this condition is set in the Interrupt Flag register.

*Note:* On the 70/35, the Scratch-Pad Memory is contained in non-addressable main memory.

## PROGRAM CONTROL AND ARITHMETIC UNIT

◆ The program control and arithmetic unit in the Model 70/45 and 70/55 Processors interprets and executes the instructions stored in main memory. Registers and indicators monitor the sequence of operations, perform automatic accuracy checks, and communicate with the RCA standard interface in the control of input/output devices.

## INPUT/OUTPUT CONTROL

◆ The RCA 70/35, 70/45 and 70/55 Processors communicate with all input/output devices through the RCA standard interface.

The 70/35 Processor can have up to two selector channels (optional). Each selector channel contains two standard interface trunks. Each standard interface trunk controls one device subsystem (from 1 to 16 devices). All selector channels can operate simultaneously.

The 70/45 Processor can have up to three selector channels (optional). Each selector channel contains two standard interface trunks. Each standard interface trunk controls one device subsystem (from 1 to 16 devices). All selector channels can operate simultaneously.

The 70/55 Processor can have up to six selector channels (optional). Each selector channel contains four standard interface trunks. Each standard interface trunk controls one device subsystem (from 1 to 16 devices). All selector channels can operate simultaneously.

In addition to the selector channels, a multiplexor channel is standard equipment on the 70/35, 70/45 and 70/55 Processors. The multiplexor channel on the 70/35 contains seven standard interface trunks. Each trunk controls one device subsystem. An eighth trunk is provided on the multiplexor for exclusive use of the \*Model 70/97 Console.

## INPUT/OUTPUT CONTROL (Cont'd)

The multiplexor channel on the 70/45 and 70/55 contains eight standard interface trunks. Each trunk controls one device subsystem. A ninth trunk is provided on the multiplexor for exclusive use of the \*Model 70/97 Console. All trunks on the multiplexor channel can operate simultaneously. Also, the multiplexor channel and all selector channels can operate simultaneously.

## READ-ONLY MEMORY

◆ Read-Only Memory is a standard feature of both the Spectra 70/35 and 70/45 Processors. The 70/35 ROM consists of 1,024 54-bit words (each containing two microinstructions of 27-bit length); and the 70/45 ROM consists of 2,048 54-bit words (each containing one microinstruction of 53-bit length). In addition both the 70/35 and 70/45 ROM each contain a 12-bit address register and a 54-bit memory register.

The wired-in microprogram logic contained in these read-only memory banks control the elementary operations of the 70/35 and 70/45. The effective cycle time of both ROM banks is 480 nanoseconds with a 54-bit access.

The 70/35 Processor can be ordered with *one* additional ROM bank containing the microinstructions for either the 1401 or the RCA 301 Emulator feature (but not both). The 70/45 Processor can be ordered with *two* additional ROM banks containing the microinstructions for any combination of the available Emulator features.

Although the Read-Only Memory is a standard feature in the 70/35 and 70/45, it is not accessible by programming and the programmer need not be familiar with the detailed method of operation of the ROM.

## INSTRUCTION FORMATS

◆ The five basic instruction formats express, in general terms, the operation to be performed as follows:

RR = register-to-register

RX = register-to-indexed main memory

RS = register-to-main memory

SI = main memory and immediate operand operation

SS = main memory to main memory

The instruction subfields are defined as follows:

R<sub>1</sub>, R<sub>2</sub>, R<sub>3</sub> — four-bit general register designation used for an operand

X<sub>2</sub> — four-bit general register designation used for indexing

B<sub>1</sub>, B<sub>2</sub> — four-bit general register designation used for base addressing

D<sub>1</sub>, D<sub>2</sub> — 12-bit displacement

I<sub>2</sub> — eight-bit immediate operand

L<sub>1</sub>, L<sub>2</sub> — four-bit operand length specification

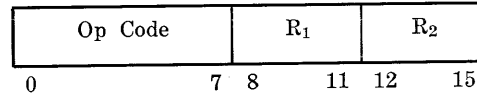
L — eight-bit operand length specification

M — eight-bit mask

\* The Model 70/97 Console has been assigned the permanent device address of "0" (zero) as a standard.

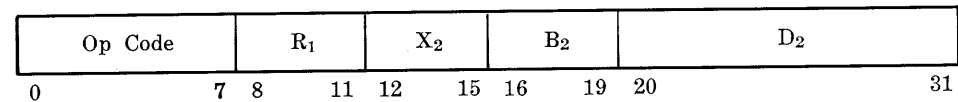
### RR FORMAT

- ◆ The contents of the general register specified by  $R_1$  is the first operand. The contents of the general register specified by  $R_2$  is the second operand. In floating-point operations,  $R_1$  designates the address of the floating-point register that contains the first operand.  $R_2$  designates the floating-point register that contains the second operand. The first and second operands can be the same and are designated by identical  $R_1$  and  $R_2$  addresses.



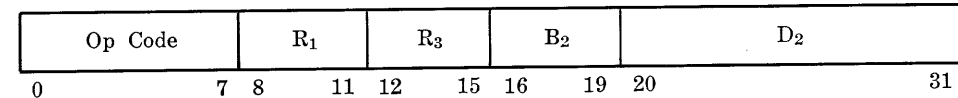
### RX FORMAT

- ◆ The contents of the general register specified by  $R_1$  is the first operand. To obtain the address of the second operand, the contents of the general registers specified by  $X_2$  and  $B_2$  are added to the  $D_2$  field. In floating-point operations,  $R_1$  designates the floating-point register that contains the first operand.



### RS FORMAT

- ◆ The RS format is used by shift instructions, branching instructions, and load/store multiple instructions.



### Shift Instructions

- ◆ The contents of the general register specified by  $R_1$  is the first operand. The contents of the general register specified by  $B_2$  are added to the  $D_2$  field. The sum specifies the number of bits of shifting to be done by the shift operation. The  $R_3$  field is ignored.

### Branching Instructions

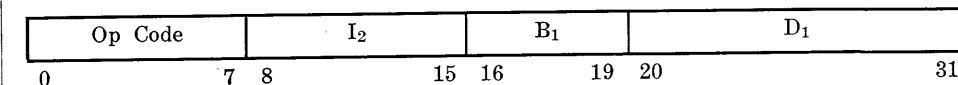
- ◆ The contents of the general register specified by  $R_1$  is the first operand. The contents of the general register specified by  $B_2$  are added to the  $D_2$  field to obtain the branch address. The contents of the general register specified by  $R_3$  is the third operand.

### Load/Store Multiple Instructions

- ◆ The  $R_1$  and  $R_3$  fields specify the general register boundaries. The contents of the general register specified by  $B_2$  are added to the  $D_2$  field to obtain the main memory address of the second operand.

### SI FORMAT

- ◆ The contents of the general register specified by  $B_1$  are added to the contents of the  $D_1$  field to obtain the address of the first operand. The second operand is the immediate eight-bit byte in the  $I_2$  field of instruction.



### SS FORMAT

- ◆ The contents of the general register specified by  $B_1$  are added to the contents of the  $D_1$  field to obtain the address of the leftmost byte of the first operand. The  $L_1$  field specifies the number of additional bytes in the operand that are to the right of the first operand address. To obtain the second operand address, the contents of the general register specified by

**SS FORMAT**  
(Cont'd)

B<sub>2</sub> are added to the contents of the D<sub>2</sub> field. The L<sub>2</sub> field specifies the number of additional bytes in the operand that are to the right of the second operand address. The L field specifies the number of additional bytes that are to the right of the first and the second operand address.

| Op Code | L              |                | B <sub>1</sub> | D <sub>1</sub> | B <sub>2</sub> | D <sub>2</sub> |    |    |    |    |    |    |    |
|---------|----------------|----------------|----------------|----------------|----------------|----------------|----|----|----|----|----|----|----|
|         | L <sub>1</sub> | L <sub>2</sub> |                |                |                |                |    |    |    |    |    |    |    |
| 0       | 7              | 8              | 11             | 12             | 15             | 16             | 19 | 20 | 31 | 32 | 35 | 36 | 47 |

**Notes**

- ◆ 1. A zero appearing in the X<sub>2</sub>, B<sub>1</sub> or B<sub>2</sub> fields indicates an absence of the corresponding address or shift-amount component. An instruction can specify the same general register both for address modification and for operand location.
- 2. Address modification is completed before the execution of an operation.
- 3. The results replace the first operand (except in Store Character instruction), where the result replaces the second operand.
- 4. A variable-length result is never stored outside the field specified by the address and length.
- 5. The contents of all registers and main memory locations not specified by an instruction remain unchanged except for the Edit and Mark instruction and the Translate and Test instructions. These instructions automatically use certain general registers as given in table 4.

**Table 4. Use of General Registers**

| Processor State* | Edit and Mark | Translate and Test |
|------------------|---------------|--------------------|
| P <sub>1</sub>   | GR 1          | GR 1 and 2         |
| P <sub>2</sub>   | GR 1          | GR 1 and 2         |
| P <sub>3</sub>   | GR 13         | GR 13 and 14       |
| P <sub>4</sub>   | GR 9          | GR 9 and 10        |

\* Processor States are discussed on page 9.

## ADDRESSING

◆ Locations in main memory are consecutively numbered starting with zero. In forming an address, the base address ( $B_1 B_2$ ) and the index ( $X_2$ ) are treated as unsigned 24-bit positive binary numbers. The displacement ( $D_1 D_2$ ) is treated as a 12-bit positive binary number. The three are added together as absolute binary numbers and overflow is ignored. The results of these additions is an effective address of up to 24-bits depending on the processor model as follows:

70/35 — yields a 16-bit effective address

70/45 — yields an 18-bit effective address

70/55 — yields a 24-bit effective address

Any address which is within the effective address as shown above, but specifies memory not available in the particular installation, causes an interrupt to occur. Any address which is outside the effective address as shown above is ignored. However, to maintain program compatibility on all processors, all addressing should assume a 24-bit effective address. Negative indexing may be achieved by address wrap-around since overflow bits over the 24-bit address are ignored.

## PROGRAM INTERRUPT

### INTRODUCTION

◆ Program interrupts occur as a result of errors in data or instruction specifications, input/output operations, external signals, equipment malfunctions or arithmetic errors. The instruction being executed at the time of the interrupt can be completed, suppressed, or terminated depending on the cause of the interrupt.

An interrupt can be inhibited or permitted in any state through programming. If an interrupt occurs and is permitted, conditions existing in the interrupted state are automatically stored. Control is then passed to the Interrupt Control State  $P_3$  or Machine Condition State  $P_4$ , depending on the cause of the interrupt. (See Processor States below.) The priority of the interrupt is established and an analysis is made to determine the proper linkage to the Interrupt Response State  $P_2$  so that the interrupt may be processed. After interrupt processing is completed, control is returned to the state which was last interrupted, and normal processing is resumed.

If several interrupts occur at the same time, the one having the highest priority is processed. The remaining interrupts are processed in turn, depending on their priority.

## PROCESSOR STATES

◆ The RCA 70/35, 70/45 and 70/55 Processors have four processor states that provide control of system and program interrupts. Programs can be executed in any one of the states, because each state is completely independent and has its own set of registers. The processor states and their functions are as follows:

### Processing State $P_1$

◆ The Processing State  $P_1$  interprets and executes the user's program. This processing state is the problem-oriented state.

### Interrupt Response State $P_2$

◆ The Interrupt Response State  $P_2$  performs specific program tasks as dictated by the Interrupt Control State  $P_3$ .



**Interrupt Control  
State P<sub>3</sub>**

◆ The Interrupt Control State P<sub>3</sub> is automatically entered when an interrupt is recognized that is other than one caused by a machine check or power failure. In this state, programming is responsible for performing a detailed analysis of the cause of the interrupt and establishing its priority. After these functions are performed, linkage is provided to the related interrupt processing routine in the Interrupt Response State P<sub>2</sub>.

**Machine Condition  
State P<sub>4</sub>**

◆ The Machine Condition State P<sub>4</sub> is entered whenever a machine check or power failure occurs. In this state, programming analyzes the cause of a machine interrupt and establishes its priority. Control is then transferred to the Interrupt Response State P<sub>2</sub>, so that an indication of the cause of interrupt can be given to the operator.

**PROCESSOR STATE  
REGISTERS**

◆ Registers are provided in scratch-pad memory, for each processor state as given in table 5.

**Table 5. Processor State Registers**

| Register                  | State          |                |                |                |
|---------------------------|----------------|----------------|----------------|----------------|
|                           | P <sub>1</sub> | P <sub>2</sub> | P <sub>3</sub> | P <sub>4</sub> |
| Program Counter           | 1              | 1              | 1              | 1              |
| General Registers         | 16             | 16             | 6              | 5              |
| Floating-Point Registers  | 4              | *              | *              | *              |
| Interrupt Status Register | 1              | 1              | 1              | 1              |
| Interrupt Mask Register   | 1              | 1              | 1              | 1              |

\* Floating-point instructions executed in any of the processor states use the floating-point registers assigned to P<sub>1</sub>.

Because each processor state has its own general registers, Interrupt Status Register and Interrupt Mask Register, storing and reloading these registers is not necessary during interrupt processing.

**Program Counter**

◆ The Program Counter (P counter) is a 32-bit register that is located in scratch-pad memory. A separate P counter is provided for each of the four processor states.

The format of the P counter is as follows:

| ILC       | CC | Program Mask | Next Instruction Address |
|-----------|----|--------------|--------------------------|
| 0 1 2 3 4 |    | 7 8          | 31                       |

*Bit Positions 0 and 1* contain the instruction length code. When an interrupt occurs and is taken, or a Program Control instruction is executed, the length of the last instruction executed in the terminated state, before the interrupt condition occurred, is stored in bit positions 0 and 1 as given in table 6. The instruction length code is always generated from the operation code of the instruction.

**Table 6. Instruction Length Codes**

| ILC | Length in Bytes        |
|-----|------------------------|
| 01  | Two-byte instruction.  |
| 10  | Four-byte instruction. |
| 11  | Six-byte instruction.  |

Program Counter  
(Cont'd)

## Notes:

1. If the interrupt condition is an operation code trap, the length of the instruction causing the interrupt is generated from the operation code and is stored in bit positions 0 and 1 as given in table 6.
2. The instruction length code is unpredictable if the interrupt was caused by one of the following:

Power Failure

Machine Check

Address Error (only if the address error was caused  
by an invalid instruction address)

*Bit Positions 2 and 3* contain the condition code. When an interrupt occurs or a Program Control instruction is executed, the condition code is moved from a machine register, where it is maintained for instruction execution, and stored in this field of the P counter of the state being terminated. The condition code in this field of the P counter of the state being initiated is moved into a machine register where it is maintained for possible future use.

*Bit Positions 4 through 7* contain the program mask. When an interrupt occurs or a Program Control instruction is executed, the program mask is moved from the machine register, where it is maintained for instruction execution, and stored in bits 4 through 7 of the P counter of the state being terminated. The program mask in this field of the P counter of the state being initiated is moved into the machine register where it is maintained for possible future use.

*Note:* On the 70/35 Processor, there is *no* machine register used to maintain the program mask during instruction execution. The program mask is always maintained in the P counter of each state. Consequently, when an interrupt occurs or a Program Control instruction is executed on the 70/35 the program mask is not affected.

*Bit Positions 8 through 31* contain the next instruction address. This field stores the address of the next instruction in main memory to be staticized by the appropriate processor state. Each time an instruction is staticized, the P counter is updated to the next instruction. This field is left intact whenever an interrupt requires switching to a new processor state.

*Note:* Because the scratch-pad memory on the 70/35 is a portion of non-addressable main memory, a special machine register is incorporated into this processor to speed up staticizing time. This machine register contains the next instruction address and is updated each time an instruction is staticized. When an interrupt occurs or a Program Control instruction is executed, the next instruction address is moved from the machine register where it is maintained and is stored in bits 8 through 31 of the P counter of the state being terminated. The next instruction address in this field of the state being initiated is moved into the machine register where it is maintained for the initiated state.

## Floating-Point Registers

## Interrupt Status Registers

The format of each Interrupt Status register is as follows:

*Bit Positions 0 through 2* contain the interrupt state identifier. When an interrupt occurs, the number of the processor state being interrupted is stored in this field of the processor state being initiated as given in table 7.

### Table 7. Interrupt State Identifier Codes

| ISI | Definition             |
|-----|------------------------|
| 000 | $P_4$ was interrupted. |
| 001 | $P_3$ was interrupted. |
| 010 | $P_2$ was interrupted. |
| 011 | $P_1$ was interrupted. |

*Bit Positions 3 through 5 are not used and must be zeros.*

*Bit Positions 6 and 7* contain the program indicators. When an interrupt occurs due to a parity error in Main Memory or Scratch Pad Memory, the program indicators are stored in this field in P<sub>4</sub> as given in Table 8.

### Table 8. Program Indicator Codes

| Program Indicators | Definition  |
|--------------------|---|
| 00                 | Neither error has occurred.   |
| 01                 | Scratch Pad Memory parity error has occurred.                               |
| 10                 | Main Memory parity error has occurred.                                      |
| 11                 | Scratch Pad Memory parity error and Main Memory parity error have occurred. |

*Note:* On the 70/35 Processor, the program indicators are always zeros since Scratch Pad Memory is a part of non-addressable main memory.

# Interrupt Status Registers (Cont'd)

*Bit Positions 8 through 11* contain the memory protection key. This field is set by the program to indicate the desired protection key. When an interrupt occurs or a Program Control instruction is executed, the memory protection key is extracted from this field of the processor state being initiated and placed in a machine register where it performs the memory protect function. The four-bit key provides a possible 15 keys ranging from  $(1)_{16}$  to  $(F)_{16}$ . Each 2,048-byte block of main memory has its individual machine register for the protection key. When the key related to the current processor state and the key related to the main memory block are equal, or either is zero, the main memory block accepts a data store. Conversely, if the keys do not match, and neither is zero, an address error (protection) interrupt occurs.

*Note:* If the memory protect feature is not installed, this field must be zero.

*Bit Position 12* designates the internal decimal code. When an interrupt occurs or a Program Control instruction is executed, the decimal code (either ASCII or EBCDIC) for the processor state being initiated is established by the setting of this bit. If the bit is 1, ASCII Code is established; if the bit is 0, EBCDIC is established.

*Note:* The setting of this Decimal Code does not affect any automatic translation of data read into or written from the processor. The Decimal Code is used to determine what zone configuration (ASCII or EBCDIC) is to be established internally when executing the decimal arithmetic instruction set, the Edit instruction, and the Edit and Mask instruction.

*Bit Positions 13 and 14* are used when an Emulator feature is included in the system. If an Emulator feature is not installed, this field must be zero or an address error interrupt occurs (for further details, refer to the Emulator Reference Manual).

*Bit Position 15* is the non-privileged mode bit. This field is set by the program to indicate the privileged status of the processor state being initiated. If  $N = 0$ , the initiated processor state runs in the privileged mode, allowing execution of the privileged instructions; if  $N = 1$ , the processor state runs in the non-privileged mode, inhibiting the execution of the privileged instructions.

*Bit Positions 16 through 23* are not used and must be zeros.

*Bit Positions 24 through 31* is the call field. This field is set during the execution of a Supervisor Call instruction. The  $R_1$  and  $R_2$  field of this instruction provide a code which is placed into the call field of the Interrupt Status register of the processor state in which the Supervisor Call instruction is issued. This code provides linkage to the program required to accomplish the purpose of the Supervisor Call instruction.

# Interrupt Mask Registers

◆ The Interrupt Mask register is a 32-bit register. A separate register is provided for each of the four processor states. Each bit in the Interrupt Mask register is associated with an interrupt condition. A 0 bit in any bit position in this register inhibits the associated interrupt condition; a 1 bit in any bit position in this register permits the associated interrupt condition.

### Interrupt Mask Registers (Cont'd)

#### Important:

1. The Power Failure and Machine Check interrupts must be inhibited in the Machine Condition State  $P_4$ . The mask bits in the Interrupt Mask register for these interrupt conditions must always be zero. This is a program restriction.

2. The Address Error interrupt must be inhibited in the Interrupt Control State  $P_3$ . The mask bit in the Interrupt Mask register for this interrupt condition must always be zero. This is a programming restriction.

### Program Mask Registers

◆ In addition to the Interrupt Mask register, a Program Mask register is also provided for each state. The Program Mask register is not contained in main memory or scratch-pad memory. It is a separate machine register which is set by the non-privileged instruction, Set Program Mask, and it applies to the following interrupt conditions:

Significance error.  
Exponent underflow.  
Decimal overflow.  
Fixed-point overflow.

The program mask bit settings have priority over the bit settings in the Interrupt Mask register for the above four program interrupts. A 0 bit in any bit position in this register cancels the interrupt condition if it occurs. A 1 bit in any bit position in this register indicates that the Interrupt Mask register is to be examined. If an interrupt condition occurs and is inhibited by the Interrupt Mask register, it remains pending until it is serviced (permitted).

### Register Addressing

◆ Register addressing in each of the processor states is given in table 9.

Table 9. Register Addressing in the Processor States

| Register Number | Processor States |       |                         |                        |
|-----------------|------------------|-------|-------------------------|------------------------|
|                 | $P_1$            | $P_2$ | $P_3$                   | $P_4$                  |
| 0               | GR               | GR    | IMR, $P_1$ State        | Processor Utility      |
| 1               | GR               | GR    | ISR, $P_1$ State        | Processor Utility      |
| 2               | GR               | GR    | P counter, $P_1$ State  | Processor Utility      |
| 3               | GR               | GR    | Interrupt Flag Register | Processor Utility      |
| 4               | GR               | GR    | IMR, $P_2$ State        | Processor Utility      |
| 5               | GR               | GR    | ISR, $P_2$ State        | Processor Utility      |
| 6               | GR               | GR    | P counter, $P_2$ State  | Processor Utility      |
| 7               | GR               | GR    | GR                      | Processor Utility      |
| 8               | GR               | GR    | IMR, $P_3$ State        | GR                     |
| 9               | GR               | GR    | ISR, $P_3$ State        | GR                     |
| 10              | GR               | GR    | P counter, $P_3$ State  | GR                     |
| 11              | GR               | GR    | GR                      | GR                     |
| 12              | GR               | GR    | GR                      | IMR, $P_4$ State       |
| 13              | GR               | GR    | GR                      | ISR, $P_4$ State       |
| 14              | GR               | GR    | GR                      | P counter, $P_4$ State |
| 15              | GR               | GR    | GR/Weight               | GR/Weight              |

GR = General Register  
IMR = Interrupt Mask Register  
ISR = Interrupt Status Register

**Register Addressing**  
*(Cont'd)*
**Notes:**

1. The P counter, Interrupt Status register, and Interrupt Mask register for processor state  $P_1$ ,  $P_2$  and  $P_3$  can be addressed by register notation ( $R_1$ ,  $R_2$  or  $R_3$  field of an instruction) in processor state  $P_3$  only. The P counter, ISR and IMR for processor state  $P_4$  can be addressed by register notation in processor state  $P_4$  only. Because the P counter, the ISR's and the IMR's are contained in scratch-pad memory, they can be addressed in any of the processor states by using the Load Scratch Pad instruction and the Store Scratch Pad instruction. However, these instructions are privileged instructions and the processor state in which they are executed must be running in the privileged mode. (Bit position 15 of the appropriate Interrupt Status register must be set to zero.)
2. Floating-Point registers may be addressed by floating-point instructions only, and are addressed as 0, 2, 4 and 6 in all processor states.

**Interrupt Flag Register**

◆ The Interrupt Flag register is a 32-bit register. There is only one Interrupt Flag register. When an interrupt condition occurs, a bit associated with the specific interrupt is set in the Interrupt Flag register. If the corresponding bit in the Interrupt Mask register for the current state is set, an interrupt occurs.

*Note:* If the interrupt condition is one of the four program interrupts, the corresponding bit in the Program Mask register must also be set to cause an interrupt.

The Interrupt Flag register is scanned on a priority basis and the highest priority interrupts are serviced first. Each interrupt condition is assigned a specific weight which is put into the rightmost eight bits of General register No. 15 of the initiated state ( $P_3$  or  $P_4$ ). This weight can be used by the program to enter the proper interrupt routine.

*Note:* General register No. 15 in  $P_3$  or  $P_4$  is cleared and reloaded each time an interrupt occurs.

Table 10 lists the priority, the Interrupt Flag register position, the program state initiated, and the weight of each of the interrupt conditions.

**Table 10. Interrupt Conditions and Priority**

| Priority | Interrupt Condition    | Flag *Bit      | State Initiated | Weight |
|----------|------------------------|----------------|-----------------|--------|
| 1        | Power Failure          | 2 <sup>0</sup> | $P_4$           | 0      |
| 2        | Machine Check          | 2 <sup>1</sup> | $P_4$           | 4      |
| 3        | External Signal No. 1  | 2 <sup>2</sup> | $P_3$           | 8      |
| 4        | External Signal No. 2  | 2 <sup>3</sup> | $P_3$           | 12     |
| 5        | External Signal No. 3  | 2 <sup>4</sup> | $P_3$           | 16     |
| 6        | External Signal No. 4  | 2 <sup>5</sup> | $P_3$           | 20     |
| 7        | External Signal No. 5  | 2 <sup>6</sup> | $P_3$           | 24     |
| 8        | External Signal No. 6  | 2 <sup>7</sup> | $P_3$           | 28     |
| 9        | Not Specified          | 2 <sup>8</sup> | $P_3$           | 32     |
| 10       | Selector Channel No. 1 | 2 <sup>9</sup> | $P_3$           | 36     |

(Cont.)

Interrupt Flag Register  
(Cont'd)

Table 10. Interrupt Conditions and Priority (Cont'd)

| Priority | Interrupt Condition                                   | Flag<br>*Bit | State<br>Initiated | Weight |
|----------|---|--------------|--------------------|--------|
| 11       | Selector Channel No. 2                                | $2^{10}$     | $P_3$              | 40     |
| 12       | Selector Channel No. 3                                | $2^{11}$     | $P_3$              | 44     |
| 13       | Selector Channel No. 4                                | $2^{12}$     | $P_3$              | 48     |
| 14       | Selector Channel No. 5                                | $2^{13}$     | $P_3$              | 52     |
| 15       | Selector Channel No. 6                                | $2^{14}$     | $P_3$              | 56     |
| 16       | Multiplexor Channel                                   | $2^{15}$     | $P_3$              | 60     |
| 17       | Elapsed Time Clock                                    | $2^{16}$     | $P_3$              | 64     |
| 18       | Console Interrupt Request                             | $2^{17}$     | $P_3$              | 68     |
| 19       | Not Specified   | $2^{18}$     | $P_3$              | 72     |
| 20       | Not Specified   | $2^{19}$     | $P_3$              | 76     |
| 21       | Supervisor Call Instruction                           | $2^{20}$     | $P_3$              | 80     |
| 22       | Privileged Operation                                  | $2^{21}$     | $P_3$              | 84     |
| 23       | Op-Code Trap  | $2^{22}$     | $P_3$              | 88     |
| 24       | Address Error (Protect,<br>Addressing, Specification) | $2^{23}$     | $P_3$              | 92     |
| 25       | Data Error  | $2^{24}$     | $P_3$              | 96     |
| 26       | Exponent Overflow                                     | $2^{25}$     | $P_3$              | 100    |
| 27       | Divide Error  | $2^{26}$     | $P_3$              | 104    |
| 28       | Significant Error**                                   | $2^{27}$     | $P_3$              | 108    |
| 29       | Exponent Underflow**                                  | $2^{28}$     | $P_3$              | 112    |
| 30       | Decimal Overflow**                                    | $2^{29}$     | $P_3$              | 116    |
| 31       | Fixed Point Overflow**                                | $2^{30}$     | $P_3$              | 120    |
| 32       | Test Mode   | $2^{31}$     | $P_3$              | 124    |

\*  $2^0$  = The rightmost bit in the Interrupt Flag register.

\*\* *Note:* These interrupt conditions can be masked by two separate masks. The first, the program mask, is a four-bit, non-privileged, program settable mask, that can be used to cancel the interrupt condition when it occurs. The second mask is composed of bits  $2^{30}$  through  $2^{27}$  of the 32-bit Interrupt Mask register associated with the state in which the processor is operating. If the Program Mask prohibits the interrupt it is cancelled. If the Program Mask permits the interrupt, the Interrupt Mask register is scanned. Like all the other interrupt conditions, the masks of the 32-bit Interrupt Mask register leave these four interrupt conditions pending if the associated mask bits are zeros.

**INTERRUPT  
CONDITIONS**

◆ A description of the individual interrupt conditions is given in table 11. More detailed information concerning the interrupt conditions is given in the instruction descriptions. Some interrupt conditions arise from input/output channel operations, and these conditions are further discussed in the Input/Output Operational Control section.

*Note:* When an interrupt condition occurs, the current instruction can be suppressed or it can be terminated. When an instruction is suppressed, the condition code setting that existed before the instruction was attempted remains unchanged. Data in main memory and the general registers specified by the instruction also remain unchanged. When an instruction is terminated, the condition code setting and data in the general registers and/or main memory are unpredictable.

Table 11. Interrupt Conditions

| Priority No.                           | Condition   | Flag Bit   | Explanation  |
|--|---|--|--|
| 1                                      | Power Failure   | 2 <sup>0</sup>   | A power failure interrupt occurs when there is a power failure in the processor or main memory caused by a line failure or by pressing the MASTER pushbutton indicator on the 70/97 Console. Any instruction being executed at the time of interrupt is terminated. It is a program restriction that the mask bit in processor state P <sub>4</sub> for this interrupt condition must always be zero when this interrupt occurs. This permits the program to operate in processor state P <sub>4</sub> for the purpose of closing down the machine during a one-millisecond interval between power failure and actual power loss to the system.  |
| 2                                      | Machine Check   | 2 <sup>1</sup>   | <p>The machine check interrupt occurs when a machine fault or malfunction is detected. Any instruction being executed at the time of interrupt is terminated. It is a program restriction that the mask bit in processor state P<sub>4</sub> for this interrupt condition must always be zero when this interrupt occurs. The following conditions can cause a machine check interrupt to occur:</p> <p><i>Scratch-Pad Memory Parity Error</i>—This error is detectable on the 70/45 and 70/55 Processors only and can occur when data is read from the Scratch-Pad Memory.</p> <p><i>Main Memory or Non-Addressable Main Memory Parity Error</i>—This error is detectable on the 70/35, 70/45 and 70/55 Processors and can occur when data or instructions are read from Main Memory or Non-Addressable Main Memory (70/35 only). If a main memory parity error occurs during an I/O data transfer, this interrupt condition does not occur. A channel interrupt occurs and the program is notified of the condition via the channel status byte.</p> |
| 3<br>4<br>5<br>6<br>7<br>8             | External Signal No. 1<br>External Signal No. 2<br>External Signal No. 3<br>External Signal No. 4<br>External Signal No. 5<br>External Signal No. 6                              | 2 <sup>2</sup><br>2 <sup>3</sup><br>2 <sup>4</sup><br>2 <sup>5</sup><br>2 <sup>6</sup><br>2 <sup>7</sup>                         | The external signal interrupt occurs when a signal is received on an external line (1-6) associated with the Direct Control option. Any instruction being executed at the time of interrupt goes to completion.  |
| 9                                      | Not Used  | 2 <sup>8</sup>   |  |
| 10<br>11<br>12<br>13<br>14<br>15<br>16 | Selector Channel No. 1<br>Selector Channel No. 2<br>Selector Channel No. 3<br>Selector Channel No. 4<br>Selector Channel No. 5<br>Selector Channel No. 6<br>Multiplexor Channel | 2 <sup>9</sup><br>2 <sup>10</sup><br>2 <sup>11</sup><br>2 <sup>12</sup><br>2 <sup>13</sup><br>2 <sup>14</sup><br>2 <sup>15</sup> | <p>This interrupt provides the means by which the processor can receive and act upon signals from input/output devices connected to a Selector Channel (1-6) or the Multiplexor Channel. This interrupt can occur as a result of the termination (normal or abnormal) of an input/output operation or at the request of an input/output device. It can also occur as the result of a program controlled interrupt. Any instruction being executed at the time of interrupt goes to completion. (Selector Channels are optional.)</p> <p><i>Note:</i> Selector Channel No. 3 is applicable to the 70/45 and 70/55 only; Selector Channels No. 4, 5, and 6 are applicable to the 70/55 only.</p>   |



**Table 11. Interrupt Conditions (Cont'd)**

| Priority No. | Condition                 | Flag Bit        | Explanation   |     |                 |    |                      |    |                       |    |                      |
|--------------|---------------------------|-----------------|---|-----|-----------------|----|----------------------|----|-----------------------|----|----------------------|
| 17           | Elapsed Time Clock        | 2 <sup>16</sup> | This interrupt occurs when the Elapsed Time Clock counts downward from positive to negative, indicating that its maximum range has been reached. Any instruction being executed at the time of interrupt goes to completion. (The Elapsed Time Clock is an option.)   |     |                 |    |                      |    |                       |    |                      |
| 18           | Console Interrupt Request | 2 <sup>17</sup> | This interrupt is controlled by the Console Interrupt key on the operator's console. Any instruction being executed at the time of interrupt goes to completion.  |     |                 |    |                      |    |                       |    |                      |
| 19           | Not Used                  | 2 <sup>18</sup> |   |     |                 |    |                      |    |                       |    |                      |
| 20           | Not Used                  | 2 <sup>19</sup> |   |     |                 |    |                      |    |                       |    |                      |
| 21           | Supervisor Call           | 2 <sup>20</sup> | This interrupt results from the execution of the Supervisor Call instruction. The P counter and the Interrupt Status register of the interrupted state are updated normally. The rightmost eight bits of the Interrupt Status register of the state in which the instruction is executed receives the R <sub>1</sub> , R <sub>2</sub> field of the Supervisor Call instruction.   |     |                 |    |                      |    |                       |    |                      |
| 22           | Privileged Operation      | 2 <sup>21</sup> | <p>This interrupt occurs when a privileged instruction is attempted and the current processor state is in non-privileged mode. (Bit position 15 of the Interrupt Status register is set.) The instruction is suppressed. The privileged instructions in the 70/35, 70/45 and 70/55 Processors are:</p> <div><div>Diagnose<br/>Start Device<br/>Test Device<br/>Halt Device<br/>Check Channel<br/>Program Control<br/>Load Scratch Pad<br/>Store Scratch Pad<br/>Idle<br/>Set Storage Key<br/>Insert Storage Key<br/>Write Direct<br/>Read Direct</div><div><div>}</div><div>If the Memory Protect option is installed.</div><div>}</div><div>If the Direct Control option is installed.</div></div></div> |     |                 |    |                      |    |                       |    |                      |
| 23           | Operation Code Trap       | 2 <sup>22</sup> | <p>This interrupt occurs when an operation code that is either not assigned or not available on the particular processor is attempted. No operation is performed. The length of the instruction upon which the trap occurred is determined by the instruction length code field of the P counter of the terminated state as follows:</p> <table><tr><th>ILC</th><th>Length in Bytes</th></tr><tr><td>01</td><td>Two-byte instruction</td></tr><tr><td>10</td><td>Four-byte instruction</td></tr><tr><td>11</td><td>Six-byte instruction</td></tr></table> <p><i>Note:</i> The ILC is always generated from the operation code of the instruction.</p>   | ILC | Length in Bytes | 01 | Two-byte instruction | 10 | Four-byte instruction | 11 | Six-byte instruction |
| ILC          | Length in Bytes           |                 |   |     |                 |    |                      |    |                       |    |                      |
| 01           | Two-byte instruction      |                 |   |     |                 |    |                      |    |                       |    |                      |
| 10           | Four-byte instruction     |                 |   |     |                 |    |                      |    |                       |    |                      |
| 11           | Six-byte instruction      |                 |   |     |                 |    |                      |    |                       |    |                      |

Table 11. Interrupt Conditions (Cont'd)

| Priority No. | Condition     | Flag Bit        | Explanation   |
|--------------|---------------|-----------------|---|
| 24           | Address Error | 2 <sup>23</sup> | <p>Three conditions cause an address error interrupt to occur. They are: address error, specification error, and protection error.</p> <p><i>Addressing</i> — An address error (addressing) interrupt occurs when:</p> <ol style="list-style-type: none"> <li>1. An address specifies any part of data, an instruction or control word outside the available main memory for the particular installation. The instruction operation is terminated for an invalid data address, and the results of the instruction are unpredictable. The instruction operation is suppressed for an invalid instruction address.</li> <li>2. An Execute instruction specifies another Execute instruction to be performed. The operation is suppressed.</li> <li>3. The first operand address field of an instruction designates an odd register address for a pair of general registers that contain a double word operand. The operation is suppressed.</li> <li>4. A floating-point instruction addresses a floating-point register other than 0, 2, 4, 6. The operation is suppressed.</li> </ol> <p><i>Specification</i> — An address error (specification) interrupt occurs when:</p> <ol style="list-style-type: none"> <li>1. A data, instruction, or control word address does not specify a doubleword, word, halfword, or byte boundary as required by the particular instruction concerned. The operation is suppressed.</li> <li>2. The multiplier or divisor in decimal arithmetic exceeds 15 digits and sign. The operation is suppressed.</li> <li>3. The first operand field is not longer than the second operand field in decimal division or multiplication. The operation is suppressed.</li> <li>4. Bit positions 28 through 31 of the second operand of a Set Storage Key or Insert Storage Key instruction are not zero. The operation is suppressed.</li> <li>5. The Memory Protect option is not installed and the protection key in the Interrupt Status register is not zero. The operation is suppressed.</li> <li>6. The Program Control instruction specifies an instruction address which is not on a halfword boundary. The operation is suppressed.</li> </ol> <p><i>Protection</i> — An address error (protection) interrupt occurs when the storage key and the protection key of the result main memory location do not match, and neither is zero. The operation is suppressed if the first main memory location specified that the instruction is in a protected area. The operation is terminated with unpredictable results if the instruction is in progress when the protected area is addressed. (This interrupt can only occur if the Memory Protect option is installed.)</p> |
|              | (Cont'd)      |                 |   |

Table 11. Interrupt Conditions (Cont'd)

| Priority No. | Condition                 | Flag Bit        | Explanation   |
|--------------|---------------------------|-----------------|---|
| 24           | Address Error<br>(Cont'd) | 2 <sup>23</sup> | <p><i>Notes:</i></p> <ol style="list-style-type: none"> <li>1. If an address error type interrupt occurs during an input/output operation (after initiation), an address error interrupt does not occur. Instead, a channel interrupt occurs for the appropriate channel.</li> <li>2. It is a program restriction that the mask bit in processor state P<sub>3</sub> for this interrupt condition must always be zero when this interrupt occurs.</li> </ol>  |
| 25           | Data Error                | 2 <sup>24</sup> | <p>This interrupt occurs when any of the following conditions occur:</p> <ol style="list-style-type: none"> <li>1. The sign or digit codes of operands in decimal arithmetic, editing, or Convert To Binary instructions are incorrect.</li> <li>2. Fields overlap incorrectly in decimal arithmetic.</li> <li>3. A decimal multiplicand has too many high-order, significant digits.</li> </ol> <p>The operation is terminated (suppressed if the operation is a Convert To Binary instruction) upon detection of any of the above.</p>  |
| 26           | Exponent Overflow         | 2 <sup>25</sup> | <p>The exponent overflow interrupt occurs when the result exponent of floating-point addition, subtraction, multiplication, or division is greater than 127. The operation is terminated.</p>   |
| 27           | Divide Error              | 2 <sup>26</sup> | <p>The divide error interrupt occurs when any of the following occur:</p> <ol style="list-style-type: none"> <li>1. A quotient exceeds the general register size in fixed-point division, including division by zero. The division is suppressed.</li> <li>2. The result of a Convert To Binary instruction exceeds one word. The conversion is completed by ignoring information which is outside the general register size.</li> <li>3. A quotient exceeds the specified data field size in decimal divide. The division is suppressed.</li> <li>4. Floating-point division is attempted with a divisor whose mantissa is zero. The operation is suppressed.</li> </ol>   |
| 28           | Significance Error        | 2 <sup>27</sup> | <p>This interrupt occurs when the result mantissa of a floating-point add or subtract instruction is zero. If the interrupt is permitted (by the program mask and the the interrupt mask) the operation is completed, the exponent is unaltered, and the interrupt is taken. If the interrupt is inhibited by the program mask, the interrupt condition is cancelled and the operation is completed by setting the result to true zero (zero sign, zero exponent and zero mantissa). If the interrupt is permitted by the program mask but inhibited by the interrupt mask, the interrupt remains pending and the operation is completed by setting the result to true zero (zero sign, zero exponent and zero mantissa).</p> |

Table 11. Interrupt Conditions (Cont'd)

| Priority No. | Condition            | Flag Bit        | Explanation   |
|--------------|----------------------|-----------------|---|
| 29           | Exponent Underflow   | 2 <sup>28</sup> | This interrupt occurs when the result exponent of a floating-point addition, subtraction, multiplication, or division is less than zero. The operation is completed by making the result true zero (zero sign, zero exponent, and zero mantissa). If the interrupt is inhibited by the program mask, the interrupt condition is cancelled. If the interrupt is permitted by the program mask, but inhibited by the interrupt mask, the interrupt remains pending. |
| 30           | Decimal Overflow     | 2 <sup>29</sup> | This interrupt occurs when the result field is too small to contain the result of a decimal operation. The operation is completed by ignoring the overflow data. If the interrupt is inhibited by the program mask, the interrupt condition is cancelled. If the interrupt is permitted by the program mask, but inhibited by the interrupt mask, the interrupt remains pending.  |
| 31           | Fixed-Point Overflow | 2 <sup>30</sup> | This interrupt occurs when a high-order carry occurs or high-order significant bits are lost in fixed-point addition, subtraction, shifting, or sign control operations. The operation is completed by ignoring the overflow data. If the interrupt is inhibited by the program mask, the interrupt condition is cancelled. If the interrupt is permitted by the program mask, but inhibited by the interrupt mask, the interrupt remains pending.                |
| 32           | Test Mode            | 2 <sup>31</sup> | This interrupt provides program control over the processor during program testing. The program test interrupt flag is set by the Program Control instruction. When the interrupt flag bit and the related interrupt mask bit in the state to be initiated are both set, an interrupt occurs after the first instruction that is executed in the initiated processor state.  |

## INTERRUPT MECHANIZATION

- ◆ There are two ways of causing a change of processor state. They are:
  1. *Automatic Interrupt*: effected when any interrupt condition described in table 11 occurs, and is permitted.
  2. *Program Controlled Interrupt*: effected when a Program Control instruction is executed.

Whenever the processor state is changed, either by automatic interrupt or by the execution of a Program Control instruction, some machine conditions must be stored in the P counter and the Interrupt Status register of the terminated state for possible use when the state is initiated again. In addition, certain machine conditions associated with the state being initiated must be extracted from the P counter and the Interrupt Status register of the new state.

All the storing and extracting required when processor status are changed is accomplished by hardware.

### Automatic Interrupt

- ◆ When an automatic interrupt condition occurs, the following events occur: (See figure 2.)

#### Block 1

- ◆ A check is made to see if the interrupt condition is one of the following four:

Significance Error

Exponent Underflow

Decimal Overflow

Fixed-Point Overflow

#### Block 2

- ◆ If the interrupt condition is one of the above, the program mask (machine register) for the current program state is checked to see if the interrupt is permitted. If the program mask indicates that the interrupt is inhibited (mask = 0), the interrupt condition is cancelled and the next instruction in the current processor state is executed.

#### Block 3

- ◆ If the interrupt condition is not one of the four program interrupts, or is one of the four program interrupts but the program mask indicates that the interrupt is to be permitted (mask = 1), the specific bit associated with the interrupt condition is set in the Interrupt Flag register.

#### Block 4

- ◆ The bit in the Interrupt Flag register is compared with the corresponding bit in the Interrupt Mask register for the current state. If the bit in the Interrupt Mask register is reset (0), the interrupt condition remains pending and the next instruction in the current processor state is executed. The interrupt remains pending until the mask is changed to a permit status and the interrupt is serviced.

#### Block 5

- ◆ If the bit in the Interrupt Mask register is set, the interrupt is taken and information (ILC, CC, program mask) is stored in the P counter of the state being terminated.

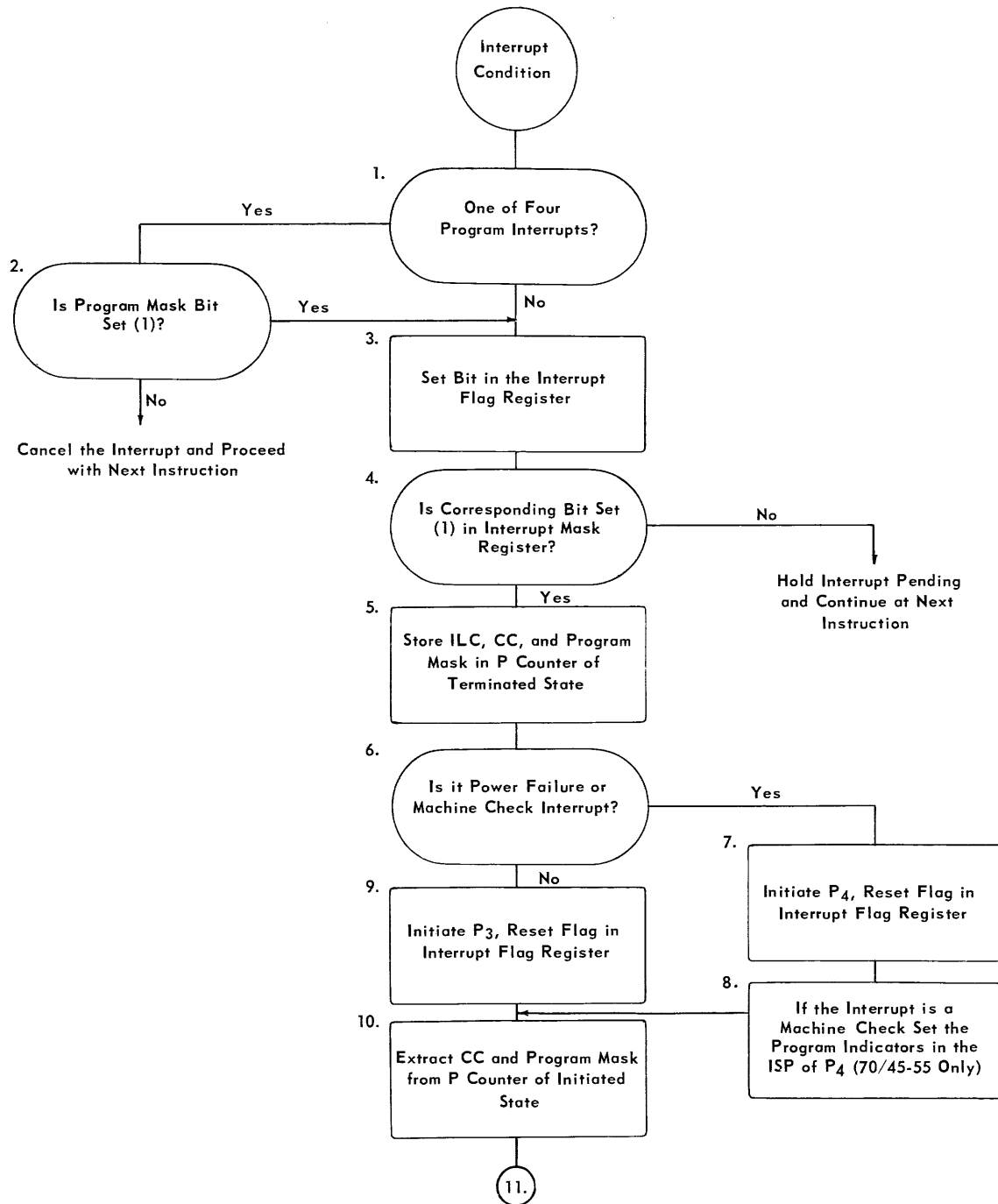


Figure 2. Functional Logic of Automatic Interrupt

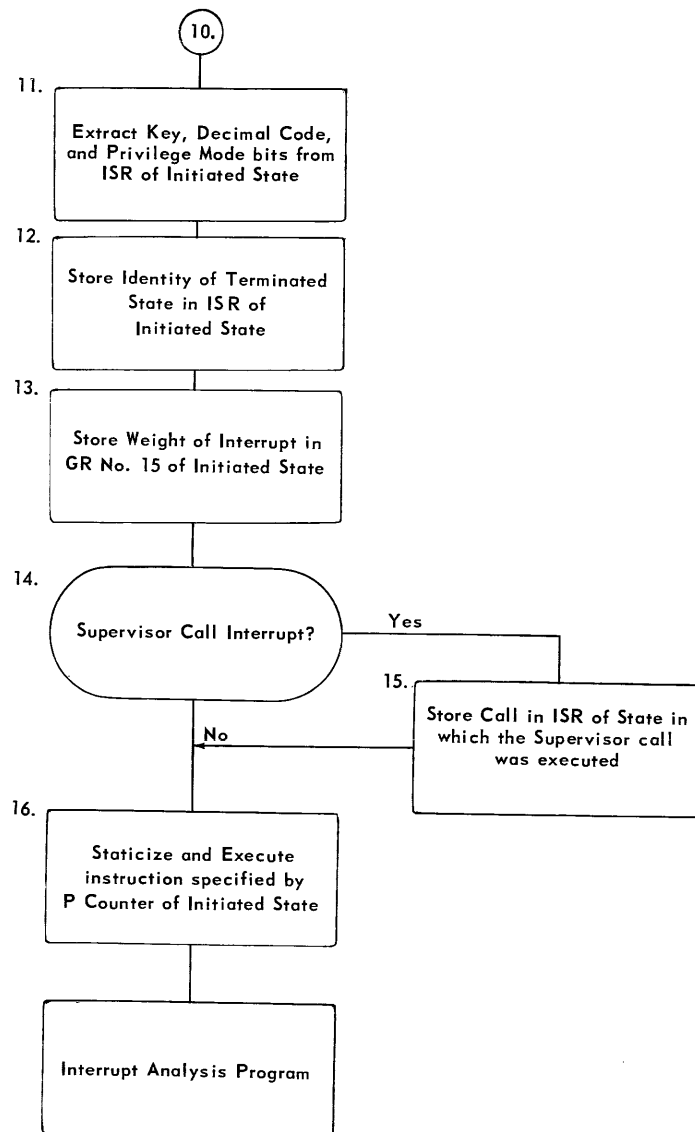


Figure 2. Functional Logic of Automatic Interrupt (Cont'd)

**Automatic Interrupt**  
(Cont'd)

*Blocks 6 and 7*

◆ If the interrupt condition is a power failure or a machine check, the Machine Condition State  $P_4$  is initiated. The flag in the Interrupt Flag register is reset.

*Block 8*

◆ If the interrupt is a Machine Check, the Program Indicators are stored in the Interrupt Status register of  $P_4$ . (The Program Indicators are applicable only on the 70/45 and 70/55 Processors.)

*Block 9*

◆ If the interrupt condition is not a power failure or machine check, the Interrupt Control State  $P_3$  is initiated. The flag in the Interrupt Flag register is reset.

*Block 10*

◆ The condition code setting and the program mask are extracted from the P counter of the initiated state and stored in the appropriate hardware registers.

*Block 11*

◆ The memory protection key, the decimal code and the privileged mode bits are extracted from the Interrupt Status register of the initiated state and stored in the appropriate registers.

*Block 12*

◆ The state being terminated is identified to the state being initiated by setting an interrupted state identifier code in the Interrupt Status register of the initiated state.

*Block 13*

◆ The weight of the condition causing the interrupt is stored in general register No. 15 of the initiated state ( $P_3$  or  $P_4$ ).

*Blocks 14 and 15*

◆ If the interrupt condition is a Supervisor Call, the  $R_1$  and  $R_2$  fields of the Supervisor Call instruction are stored in the rightmost eight-bits of the Interrupt Status register of the state in which the instruction is executed.

*Block 16*

◆ The instruction at the address specified in the P counter of the initiated state is staticized and executed.

**Program Controlled  
Interrupt**

The Program Control instruction transfers the program from one processor state to another. This instruction is a privileged operation and can be executed only if the state in which the processor is operating is in the privileged mode (bit position 15 of the Interrupt Status register = 0). When a Program Control instruction is executed, the following events occur. (See Figure 3.)

*Block 1*

◆ The address ( $B_1/D_1$ ) specified in the Program Control instruction is stored in the P counter of the terminated state. The length of the last instruction executed in the terminated state, the condition code setting, and the program mask is stored in the P counter of the terminated state.

*Block 2*

◆ A check is made to see if the program test bit in the Program Control instruction is set.



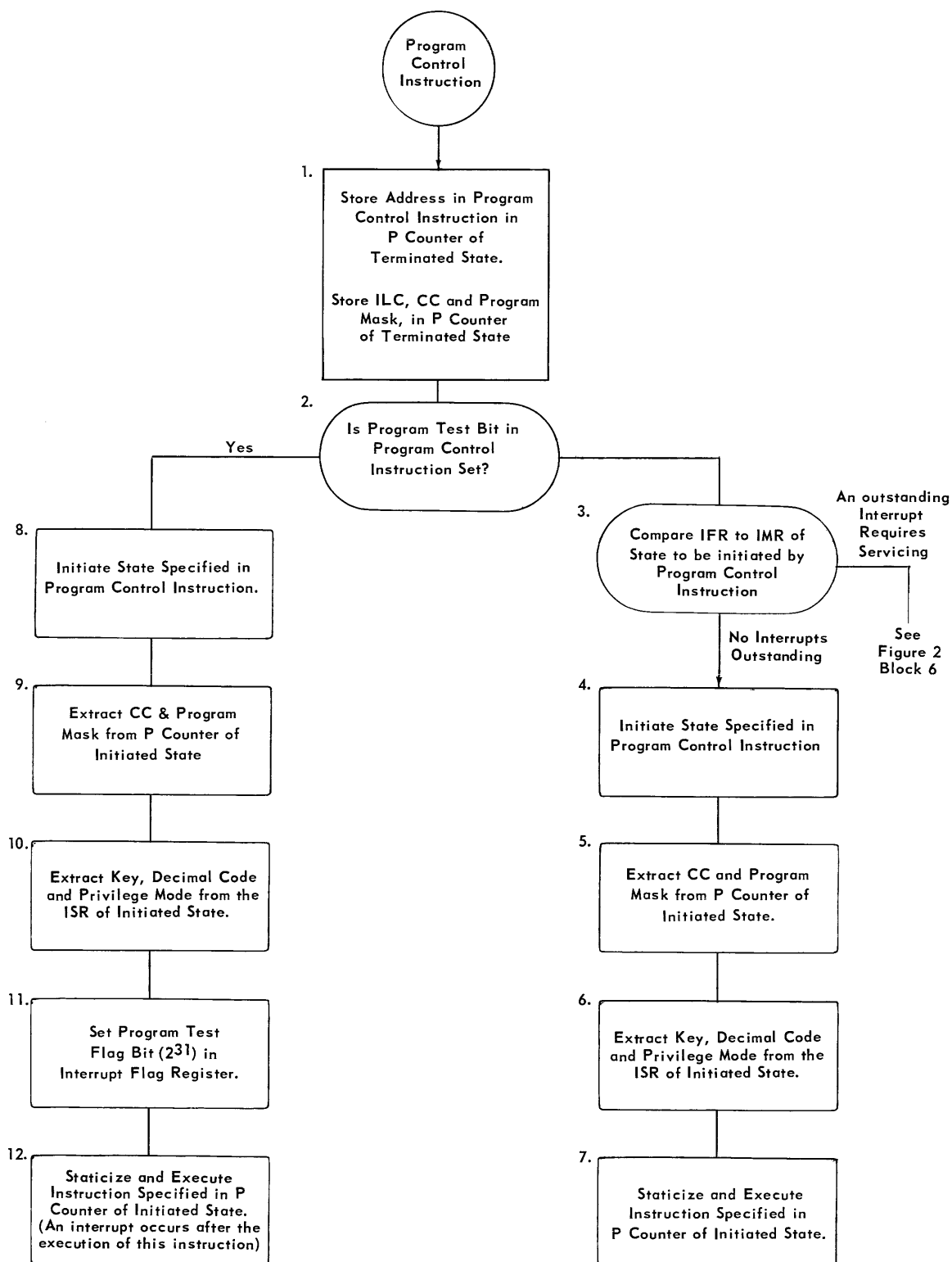


Figure 3. Functional Logic of Program Control Instruction

**Program Controlled  
Interrupt  
(Cont'd)**

*Block 3*

◆ If the program test bit is not set, the Interrupt Mask register for the state to be initiated by the Program Control instruction is compared to the Interrupt Flag register. If an interrupt condition has occurred, the events described under automatic interrupt take place (see figure 2, block 3).

*Important:* If an interrupt is outstanding in the state to be initiated by the Program Control instruction, the number of the *initiated state* specified by the Program Control instruction is stored in the interrupt status identifier field of the Interrupt Status register of the initiated state ( $P_3$  or  $P_4$ ).

*Block 4*

◆ If an interrupt condition is not outstanding in the state to be initiated by the Program Control, instruction control is transferred to the state specified by the Program Control instruction (directly or indirectly — See Program Control instruction).

*Block 5*

◆ The condition code setting and the program mask are extracted from the P counter of the initiated state and stored in the appropriate machine registers.

*Block 6*

◆ The memory protection key, the decimal code and the privileged mode bits are extracted from the Interrupt Status register of the initiated state and stored in the appropriate machine registers.

*Block 7*

◆ The instruction at the address specified in the P counter of the initiated state is staticized and executed.

*Block 8*

◆ If the program test bit is set, control is transferred to the state specified by the Program Control instruction (directly or indirectly — see Program Control instruction).

*Block 9*

◆ The condition code setting and the program mask are extracted from the P counter of the initiated state and stored in the appropriate registers.

*Block 10*

◆ The memory protection key, the decimal code, and the privileged mode bits are extracted from the Interrupt Status register of the initiated state and stored in the appropriate registers.

*Block 11*

◆ The program test flag bit ( $2^{31}$ ) in the Interrupt Flag register is set.

*Block 12*

◆ The instruction at the address specified in the P counter of the initiated state is staticized and executed.

*Notes:*

1. When a Program Control instruction has the program test bit set, the first instruction of the initiated state is always executed before any interrupt is taken.

**Program Controlled  
Interrupt**  
*Block 12*  
*(Cont'd)*

2. If the initiated state permits the program test interrupt (via the Interrupt Mask register), a program test interrupt occurs after the first instruction in the initiated state is executed.
3. An interrupt condition can occur while executing the first instruction of the initiated state. If it does, and is permitted, it is serviced before the program test interrupt.

*General Notes for Program Interrupt:*

1. The decimal mode in the 70/45 and 70/55 Processors is either ASCII or EBCDIC as specified by bit 12 in the Interrupt Status register. When an automatic interrupt occurs or a Program Control instruction is executed, the decimal mode is not stored in the Interrupt Status register of the terminated state. The mode of the state being initiated is determined by the mode bit in its own Interrupt Status register.

Consequently, to change mode, the mode bit of the Interrupt Status register associated with the appropriate state must be altered by the program, and that state must be initiated either by an interrupt condition or a Program Control instruction. This is the method available to the program for changing the mode.

2. The interrupt flags are scanned to determine whether or not an interrupt shall occur if the Interrupt Mask register associated with the current state or the Interrupt Flag register are written into by the program.
3. Changing the protection key, decimal mode, or privileged mode fields in the Interrupt Status register does not change the protection key, machine mode, or privileged mode bits of the associated processor state. To change the status of the processor, the state concerned must be initiated by an interrupt condition or a Program Control instruction.

## INPUT/OUTPUT OPERATION

### INTRODUCTION

◆ The RCA Model 70/35-45-55 Processors can control a variety of input/output devices. All the input/output devices function independently of normal processor operation. This simultaneous operation is achieved by processor input/output channels that control input/output operations. The control electronics of each peripheral device is connected to an input/output channel via the RCA Standard Interface. This interface permits all peripheral equipment (with the exception of remote communications and random access devices) to be attached to any channel in the 70/35-45-55 Processors. Remote communication devices must be connected to the multiplexor channel. Random access devices must be connected to a selector channel.

After an input/output operation is initiated by the program, data is transferred, byte-by-byte, between the processor and the peripheral device. This data transfer over the standard interface is controlled by the applicable input/output channel, freeing the processor to continue the program. Each of the channels on the 70/35-45-55 Processors can interrupt normal process or operations.

### INPUT/OUTPUT CHANNELS

#### Selector Channels

◆ The 70/35-45-55 Processors have two types of input/output channels, selector channels and a multiplexor channel.

◆ Up to two selector channels (optional) can be attached to a 70/35 Processor; up to three selector channels (optional) can be attached to a 70/45 Processor; and up to six selector channels (optional) can be attached to a 70/55 Processor. Each selector channel can address up to 256 peripheral devices.

On the 70/35 and 70/45 Processors, each selector channel has two standard interface trunks; on the 70/55 Processor each selector channel has four standard interface trunks. Each standard interface trunk can be connected to the control electronics of an input/output device. A device control electronics controls one device (i.e., card reader, printer), or a number of devices (i.e., tape controller: up to 16 tape stations).

Only one device can operate on a selector channel at one time. However, all selector channels can operate simultaneously with, and independently of, normal processor operation.

The multiplexor channel operates simultaneously with selector channels and independently of normal processor operation.

Control and operation of each input/output device connected to the multiplexor channel is done through a set of subchannel registers contained in non-addressable main memory.

In addition to the subchannel registers, four 32-bit registers, called multiplexor registers, are provided in scratch-pad memory. These registers are used for subchannel initiation and termination. Upon servicing a termination interrupt of a device connected to the multiplexor channel, the information which pertains to the completed operation is transferred from the non-addressable main memory to the scratch-pad memory.

**Selector Channels**  
(Cont'd)

The multiplexor registers in scratch-pad memory are called:

Channel Address Register (CAR)  
Channel Command Register-II (CCR-II)  
Channel Command Register-I (CCR-I)  
Assembly/Status Register

Each selector channel is controlled and operated via four 32-bit registers. These registers are located in scratch-pad memory and are called:

Channel Address Register (CAR)  
Channel Command Register-II (CCR-II)  
Channel Command Register-I (CCR-I)  
Assembly/Status Register

All the information that is required to control selector channel operation is contained in these registers. Data is transferred between the selector channel and the peripheral device one byte at a time.

*Note:* Because the scratch-pad memory is part of non-addressable main memory in the 70/35 Processor, machine registers are used to control selector channel operation and thereby provide a higher throughput rate. The registers in equivalent scratch-pad memory are used only during initiation and termination of input/output operations.

**Multiplexor Channel**

◆ The multiplexor channel is standard on the 70/35-45-55 Processors, and can address up to 256 devices.

The multiplexor channel has seven standard interface trunks (70/35) or eight standard interface trunks (70/45, 55) each of which can be connected to a device control electronics. This permits the multiplexor channel to operate devices on all seven or eight trunks simultaneously. The limit as to the number of input/output devices that can be connected is determined by the device control electronics. An eighth trunk (70/35) or a ninth trunk (70/45-55) is provided on the multiplexor channel for exclusive use by the Model 70/97 Console.

Although the multiplexor channel can handle slow-speed devices on a time-sharing basis, it can accommodate fast devices through a burst mode. Burst mode operation is specified by the program, and causes a transfer of data to occur between a specific device and main memory without time-sharing the multiplexor channel with other input/output devices. If a program is to specify burst mode, a program check is made that other devices on the multiplexor channel have completed operation. This ensures that data is not lost.

Data is transferred between the multiplexor channel and each peripheral device one byte at a time.

*Note:* When a burst mode operation is executed the subchannel registers are not utilized. The input/output operation is similar to a selector channel operation and is controlled entirely by the multiplexor registers in scratch-pad memory.

## **INPUT/OUTPUT OPERATIONAL CONTROL**

### **Programming Considerations Prior to Input/Output Initiation**

◆ All input/output operations are executed by the selected channel and are independent of normal processor operation. Prior to initiation of an input/output operation, the program must supply information concerning the operation. The program must store information in main memory, such as the type of operation (read, write, etc.), the data area address in main memory at which to begin the operation, and the number of bytes to be transferred by the channel. This information is called the Channel Command Word (CCW).

After the channel command word is stored in main memory, the address of this CCW must be stored in a standard main memory location. This standard location is called the Channel Address Word (CAW) and is main memory locations 72 through 75.

Once the channel address word and the channel command word have been assembled, the input/output operation can be initiated.

### **Input/Output Initiation**

◆ All input/output operations are initiated by executing a Start Device instruction or by manually depressing the LOAD pushbutton/indicator on the Model 70/97 Console. Execution of the Start Device instruction causes the information contained in the Channel Address Word (CAW) and the Channel Command Word (CCW) to be transferred to the input/output channel registers in scratch-pad memory for the specified selector channel. If the specified channel is the multiplexor channel, this information is transferred to the subchannel registers in non-addressable main memory for the specified device. Once this has been accomplished, the Start Device instruction terminates and the input/output operation has been initiated. Completion of the input/output operation is under control of the channel, and normal processor operation can proceed.

### **Channel Servicing**

#### *Servicing a Data Transfer*

◆ When an input/output operation has been initiated and the input/output device control electronics is ready to send or receive a data byte, the channel asks the processor for a service request. When the processor permits the service request, a data transfer occurs. This servicing permits the transfer of a data byte between main memory and the input/output device to occur. It also updates the information in the input/output channel registers or the subchannel registers (multiplexor) to prepare for the next data byte.

#### *End and Chaining Servicing*

◆ When an input/output operation has been completed, the channel asks the processor for another service request. This service request is required so that the channel can (1) tell the device control electronics to set a channel interrupt condition, or (2) check the current command to see if chaining is specified, and if it is to initiate the next command.

#### *Interrupt Servicing*

◆ If an input/output operation has been completed and chaining has not been specified, the input/output device control electronics causes the appropriate channel interrupt flag to be set in the Interrupt Flag register. If the Interrupt Mask register for the current processor state permits the

### Interrupt Servicing (Cont'd)

interrupt, it is taken. At this time the channel asks the processor for another service request. This service request is required so that the channel can transfer information concerning the status of the device and the channel to the input/output channel registers in scratch-pad memory. If the interrupt is caused by a device on the multiplexor channel, the appropriate subchannel registers are transferred from non-addressable main memory to scratch-pad memory.

Because all input/output servicing (servicing a data transfer, end and chaining servicing, and interrupt servicing) requires that the channel utilize main memory, scratch-pad memory and non-addressable main memory (multiplexor devices), normal processor operation is *held-off* until the servicing has been completed. Servicing is time-shared with normal mode processing.

### Servicing Priority

◆ Because input/output operations on all selector channels and the multiplexor channel proceed simultaneously, the processor must constantly scan the channels to determine their servicing status. If servicing is required by a channel, scanning is stopped and the input/output device is serviced. After a device is serviced, scanning is resumed.

Each selector channel and the multiplexor channel has a scanning priority. If servicing is required by devices on more than one channel, the channel with the highest priority is serviced first. The priority is as follows:

Selector Channel No. 1

Selector Channel No. 2

Selector Channel No. 3 (70/45 and 70/55 only)

Selector Channel No. 4 (70/55 only)

Selector Channel No. 5 (70/55 only)

Selector Channel No. 6 (70/55 only)

Multiplexor Channel

The devices on the multiplexor have a priority depending upon the standard interface trunk to which they are connected; the lower the standard interface trunk in the scanning sequence, the higher the priority.

After a device has been serviced, scanning always resumes with Selector Channel No. 1. With this scanning technique, the devices with the shortest holding time (high-speed devices) must be connected to the channel with the highest scan priority. Servicing of a device connected to the multiplexor channel may be temporarily interrupted by a selector channel service request. If this occurs, all selector channels requiring service are served before multiplexor channel servicing resumes.

The most optimum connection of device control electronics to selector channels and the multiplexor channel depends on the requirements of each installation. However, a general rule is to connect the device control electronics which control devices with the highest data transfer requirements to the channels with the highest priority. The remaining device control electronics are connected in descending order of data transfer requirements to descending priority sequence of channels.

**Channel Address  
Word  
(CAW)**

◆ The Channel Address Word (CAW) is used by the Start Device instruction (see Privileged Instructions section), and specifies the address of the first Channel Command Word (CCW) used to control the operation of the input/output device. If the Memory Protect option is installed, the memory protection key must also be stored in the CAW before a Start Device instruction is issued.

The CAW must be stored in main memory locations 72 through 75 before executing a Start Device instruction and has the following format:

| Key | 0000 | Address of CCW |
|-----|------|----------------|
| 0   | 3 4  | 7 8 31         |

*Bit Positions 0 through 3* contain the memory protection key. It is used to ensure that data is not being transferred to a protected memory area. If the Memory Protect option is not installed, these bits must be zero.

*Bit Positions 4 through 7* are reserved for future expansion.

*Bit Positions 8 through 31* contain the main memory address of the initial channel command word.

**Channel  
Command  
Word (CCW)**

◆ The Channel Command Word (CCW) supplies the information for controlling the operation of the input/output device. This information must be stored in main memory by the program before a Start Device instruction is issued. The CCW consists of two 32-bit words in main memory that must be aligned on a double word boundary. The CCW has the following format:

| Command Code | Address of First Data Byte or Address of Next CCW if Command is a Transfer in Channel |
|--------------|---|
| 0            | 7 8 31  |

| Flags    | Reserved for Future Expansion | Byte Count |
|----------|-------------------------------|------------|
| 32 36 37 | 47 48                         | 63         |

*Bit Positions 0 through 7* contain the command code, which specifies the operation to be performed by the I/O device. (See table 12.)

**Table 12. Command Code Operations**

| Command Code |   |   |     |     |   |   |   | Operation           |
|--------------|---|---|-----|-----|---|---|---|---------------------|
| 0            | 1 | 2 | 3   | 4   | 5 | 6 | 7 | Bit Position        |
| M            | M | M | M   | 0   | 0 | 0 | 1 | Sense               |
| M            | M | M | M   | M/0 | 0 | 1 | 0 | Read Reverse        |
| M            | M | M | M/B | M/0 | 0 | 1 | 1 | Write               |
| M            | M | M | M/B | M/0 | 1 | 0 | 0 | Write Erase         |
| M            | M | M | M/B | M/0 | 1 | 0 | 1 | Read                |
| M            | M | M | M   | 0   | 1 | 1 | 1 | Write Control       |
| M            | M | M | M   | 1   | 0 | 0 | 1 | Transfer in Channel |



**Channel Command  
Word (CCW)**  
(Cont'd)

*Notes:*

1. Any command code other than the ones shown in table 12 is illegal and must not be programmed. If this rule is violated, the resulting effect on the channel and device is unpredictable. If one of the legal commands is issued to a device which is not capable of accepting the operation (i.e. a Write command is issued to a card reader), the command, after being accepted, is terminated by the device control electronics. A channel interrupt occurs and the sense byte(s) indicate the illegal operation.
2. The bit position designated as "B" indicates that the specified device is connected to the multiplexor channel and the multiplexor is to be operated in the burst mode. If this position is a 1 bit, the multiplexor channel is *locked-on* to the selected device, and the servicing of other devices connected to the multiplexor channel is inhibited. A burst mode can only be initiated when it is specified in the first command of a chain. Subsequent commands, linked by chaining, cannot initiate a burst mode. However, if the first command in a chain specifies burst mode, all commands in the chain are executed under burst mode conditions.
3. Bit positions designated as M (modifier) indicate variations of the operation and are unique to the specific input/output device. Definition of these M bits is provided in the applicable input/output device reference manuals.

An explanation of the commands shown in table 12 is as follows:

*Sense (0001)* — Information is transferred from the specified input/output device control electronics to main memory. The information transferred indicates unusual conditions that occurred as a result of the last operation performed by the device. (The information received is defined in the individual input/output device reference manuals.) The address specified by the CCW is the leftmost main memory location of the input area.

*Note:* Parity is not checked on data transferred to main memory by this command.

*Read Reverse (0010)* — Information is transferred from the specified input/output device to main memory in descending order. The address specified by the CCW is the rightmost main memory location of the input area.

*Write (0011)* — Information is transferred from main memory to the specified input/output device. The address specified by the CCW is the leftmost main memory location of the output area.

*Write Erase (0100)* — Information is transferred from main memory to the specified input/output device control electronics. Data is not written to tape and the tape is erased in accordance with the byte count (applicable to magnetic tapes only). The address specified by the CCW is the leftmost main memory location of the output area.

**Channel Command  
Word (CCW)**  
(Cont'd)

*Read (0101)* — Information is transferred from the specified input/output device to main memory in ascending order. The address specified by the CCW is the leftmost main memory location of the input area.

*Write Control (0111)* — Information is transferred from main memory to the specified input/output device control electronics. The device control electronics interprets this information as control information and initiates a function not involving a data transfer. The address specified by the CCW is the leftmost main memory location of the output area.

*Transfer in Channel (1001)* — This command provides chaining of CCW's that are not located in adjacent double word main memory. An actual branch to the address of the next CCW is taken. The branch address (specified in bits 8 through 31 of the channel command word) must specify a double word location. (Bits 29 through 31 must be zero.) This command cannot be the first command in a chain. A Transfer in Channel command may address another Transfer in Channel command.

*Note:* The flag bits are ignored if a Transfer in Channel command is specified. The flag bits of the preceding command remain effective.

*Bit Positions 8 through 31* (see CCW format) contain the address of the first byte in main memory at which the input/output operation begins, or if the command is a transfer in channel, the main memory address of the next CCW to be executed. The address of the first byte of the next data segment can also be specified if data chaining.

*Bit Positions 32 through 36* are the flag bits and have the following significance:

1. Bit position 32 is the Chain Data flag (CD). In addition to transferring data to and from a single main memory area, the 70/35-45-55 Processors can read into, or write from, many non-contiguous areas of main memory by executing one Start Device instruction. When data chaining is specified by setting this bit, a chain (series of channel command words in sequence) is used and each channel command word designates an area of main memory at which to continue the current operation. When one channel command word has a lapsed byte count, the next channel command word in sequence is automatically fetched. The current operation is continued at the main memory area specified by the new channel command word. The command code of the new CCW is ignored unless it specifies a Transfer in Channel. If any of the following channel status byte conditions occur, data chaining is suppressed (see Channel Status Byte for further definition):

Program Check

Protection Check

Channel Control Check

Channel Data Check (if the operation is a write)

Incorrect Length Condition

**Channel Command  
Word (CCW)**  
(Cont'd)

When data chaining, the chain data flag in the last channel command word must be reset. This causes the data chain to be terminated upon completion of the operation specified by this CCW.

2. Bit position 33 is the Chain Command flag (CC). The 70/35-45-55 Processors can perform several operations to an input/output device by executing one Start Device instruction. When command chaining is specified by setting this bit, a chain (series of channel command words in sequence) is used and each channel command word specifies the operation to be performed. When the operation specified by one channel command word is completed, the next channel command word in sequence is automatically fetched and the operation specified is initiated. If any of the following conditions occur, command chaining is suppressed:

- a. Channel status byte conditions (see channel status byte for further definition).

Incorrect Length Condition and suppress length flag is zero.

Program Check

Protection Check

Channel Control Check

- b. Standard device byte conditions (see standard device byte for further definition).

Secondary Indicator is set

Device Inoperable is set

Device End is not set

When command chaining, the chain command flag in the last channel command word must be reset. This causes the command chain to be terminated upon completion of the operation specified by this CCW.

3. Bit position 34 is the Suppress Length Indication flag (SLI). Incorrect length occurs in the 70/35-45-55 Processors when the number of bytes specified in the channel command word is not equal to the number of bytes sought by, or sent from, the input/output device. (When a command or chain of commands terminates, the data byte count has not lapsed.) An example of an incorrect length condition is a tape read which terminates on a gap before the byte count has lapsed. If the SLI bit is set, the program does not receive an indication of an incorrect length upon termination of the input/output operation. If the SLI bit is reset, the program receives an indication of an incorrect length upon termination of the input/output operation. This indication is contained in the channel status byte.

**Channel Command  
Word (CCW)**  
(Cont'd)

*Notes:*

1. If the SLI bit is set and the chain data flag of the final CCW in a chain is reset, the incorrect length indication is suppressed, if it occurs.
2. If the chain data flag of a CCW is set and an incorrect length condition occurs, the program is notified of the condition regardless of the setting of the SLI flag.
4. Bit position 35 is the Skip flag (SKIP). In conjunction with data chaining, portions of a block of information can be suppressed during an input operation. If this bit is set, the transfer of data to main memory specified by this command is suppressed. This bit can be used only with Read, Read Reverse or Sense commands.
5. Bit position 36 is the Program Controlled Interrupt flag (PCI). During data and command chaining, the 70/35-45-55 Processors have the ability to notify the program of the progress of chaining through an interrupt when a channel command word is fetched. When this bit is set, a channel interrupt occurs when the channel command word is fetched from main memory and the first data byte has been transferred. This flag is ineffective if the channel is the multiplexor operating in burst mode.
6. Bit positions 37 through 47 are reserved for future expansion and must be set to all zeros by the program.
7. Bit positions 48 through 63 contain the count of the number of bytes to be transferred to or from main memory during the input/output operation (from 0 to 65,536 bytes). An initial count of zero specifies the maximum number of bytes to be transferred.

**INPUT/OUTPUT  
CHANNEL  
REGISTERS**

◆ The Channel Address Word (CAW) and the Channel Command Word(s) (CCW) are stored by the program in main memory. However, when an input/output operation is initiated, the information contained in the CAW and the first CCW is transferred to the scratch-pad input/output channel registers for the channel specified by the Start Device instruction. (See table 13.) Because the access speed in scratch-pad memory is faster than main memory, faster servicing of input/output devices is possible. These registers also eliminate the need for the program to reset channel command words, because incrementing and decrementing addresses and byte count is done in scratch-pad memory. These registers allow the input/output operation to proceed under control of the specified channel, thereby permitting normal mode processing to continue.

*Note:* Because the scratch-pad memory is part of non-addressable main memory in the 70/35 Processor, machine registers are used to control selector channel and multiplexor channel operation and thereby provide a higher throughput rate. The registers in equivalent scratch-pad memory are used only during initiation and termination of input/output operation.

# **INPUT/OUTPUT CHANNEL REGISTERS** (Cont'd)

Table 13. Input/Output Channel Registers

| Register                             | Selector Channel       | Multiplexor Channel       |                             |
|--------------------------------------|------------------------|---------------------------|-----------------------------|
|                                      | Scratch-Pad Memory     | Scratch-Pad Memory        | Non-Addressable Main Memory |
| Channel Address Register (CAR)       | 1 per selector channel | 1 per multiplexor channel | 1 per device                |
| Channel Command Register-I (CCR-I)   | 1 per selector channel | 1 per multiplexor channel | 1 per device                |
| Channel Command Register-II (CCR-II) | 1 per selector command | 1 per multiplexor channel | 1 per device                |
| Assembly/Status Register             | 1 per selector channel | 1 per multiplexor channel | None                        |

The format for each of these four 32-bit registers is as follows:

## **Channel Address Register (CAR)**

| Device No. | Address of next CCW |
|------------|---------------------|
| 0 7 8      | 31                  |

*Bit Positions 0 through 7* contain the device number specified in the input/output operation. This number is obtained from the B<sub>1</sub>/D<sub>1</sub> Address in the Start Device instruction.

*Bit Positions 8 through 31* contain the address of the next channel command word if chaining is specified.\* This information is obtained by incrementing the address of the first CCW by eight. The address of the first CCW is obtained from the Channel Address Word (CAW).

## **Channel Command Register-I (CCR-I)**

| 0000      | Command Code | Data Address of First Byte or Location of new CCW if Command is Transfer in Channel |
|-----------|--------------|---|
| 0 3 4 7 8 |              | 31  |

*Bit Positions 0 through 3* are used by the processor. It should be noted that these bits are used in the channel command word as modifier bits. Once the command has been initiated and the entire 8-bit command code has been sent to the specified device control electronics, these bits are used by the processor. They no longer contain the modifier bits.

*Bit Positions 4 through 7* contain the command code. This code is obtained from the channel command word. The commands are defined as follows:

Read (0101)  
 Write (0011)  
 Write Control (0111)  
 Sense (0001)  
 Read Reverse (0010)  
 Write Erase (0100)  
 Transfer in Channel (1001)

\* If a program check occurs as a result of a Transfer in Channel, the low order 3 bits of the CAR must be ignored in the 70/35 Processor. These 3 bits are cleared to zero in the 70/35 system.

**Channel Command  
Register-I (CCR-I)**  
(Cont'd)

*Bit Positions 8 through 31* contain the address of the initial byte in main memory at which the operation begins; or contains the branch address if the command is a Transfer in Channel. This information is obtained from the Channel Command Word.

**Channel Command  
Register-II (CCR-II)**

| Flags | 000     | Channel Status Byte | Byte Count |
|-------|---------|---------------------|------------|
| 0     | 4 5 7 8 | 15 16               | 31         |

*Bit Positions 0 through 4* contain the flags. The flags are obtained from the channel command word. The flag bits are defined as follows:

- Bit 0 — Chain data flag (CD)
- Bit 1 — Chain command flag (CC)
- Bit 2 — Suppress length indicator flag (SLI)
- Bit 3 — Skip flag (SKIP)
- Bit 4 — Program controlled interrupt flag (PCI)

*Bit Positions 5 through 7* are reserved for future use.

*Bit Positions 8 through 15* contain the channel status byte. The bits of the channel status byte are generated as a result of the input/output operation and are defined as follows:

- Bit 8 — Program Controlled Interrupt
- Bit 9 — Incorrect Length
- Bit 10 — Program Check
- Bit 11 — Protection Check
- Bit 12 — Channel Data Check
- Bit 13 — Channel Control Check
- Bit 14 — Reserved for use by the processor
- Bit 15 — Termination Interrupt

(For a detailed description of the above see Channel Status Byte section, below.)

*Bit Positions 16 through 31* contain the number of bytes of main memory to or from which data is transferred. This information is obtained from the Channel Command Word. The count can range from 0 bytes to 65,536 bytes.

**Assembly/Status  
Register**

| Data Bytes | Standard Device Byte |
|------------|----------------------|
| 0          | 23 24 31             |

*Bit Positions 0 through 31* are used as an intermediate storage area during the transfer of data between an input/output device connected to a selector channel and 70/55 Processor main memory. Data is transferred one byte at a time across the channel and the information is stored in these scratch-pad memory locations until a word (4 bytes) is accumulated. Then, the word is transferred to main memory, thus requiring memory access on a word basis rather than byte-by-byte. In the 70/35-45 Processors, intermediate storage is not used and data is transferred one byte at a time directly to main memory.

**Assembly/Status  
Register  
(Cont'd)**

When the device status is stored as a result of an input/output operation, bit positions 24 through 31 of the assembly/status register are used to store the standard device byte. The bits of the standard device byte supply status information pertaining to the device control electronics and the input/output device and are defined as follows:

- Bit 24 — External Device Request Interrupt Pending
- Bit 25 — Terminating Interrupt Pending
- Bit 26 — Device Busy
- Bit 27 — Control Busy (not applicable)
- Bit 28 — Device End
- Bit 29 — Secondary Indicator
- Bit 30 — Device Inoperable
- Bit 31 — Status Modifier

(For a detailed description of the above, see Standard Device Byte section, below.)

**INPUT/OUTPUT  
INSTRUCTIONS**

◆ There are four processor instructions which are concerned with input/output operations. They are Start Device, Halt Device, Check Channel and Test Device. These instructions are executed by the processor and the results, in the form of condition codes, are available upon instruction completion. It should be noted that the condition code settings indicate the results of the instruction and not the results of the input/output operation that the instruction may be initiating. The channel continues off-line to accomplish the input/output operation as specified by the instruction. However, during this time the processor continues executing subsequent instructions.

**Start Device Instruction**

◆ The Start Device instruction is a privileged operation and, therefore, can be executed only if the mode bit (bit position 15 of the Interrupt Status register for the current state) is set to zero. This instruction is executed in the normal mode. Continuation of program execution is delayed until the Start Device instruction has been terminated.

Upon execution of a Start Device instruction, the following events occur. (See figure 4).

- Block 1* ◆ If the privileged mode bit (bit position 15 of the Interrupt Status register) for the current state is not set to zero, the privileged operation bit is set in the Interrupt Flag register and an interrupt occurs (if permitted).
- Block 2* ◆ If the specified channel is a selector channel that is not available on the system, the condition code is set to 3, the Start Device instruction is terminated and program control is transferred to the next instruction. The input/output operation is not initiated.
- Block 3* ◆ If the specified channel is a selector channel that is busy or has an interrupt pending (termination or external device request) or if the specified channel is the multiplexor that is operating in the burst mode, the condition code is set to 2, the Start Device instruction is terminated and program control is transferred to the next instruction. The input/output operation is not initiated.

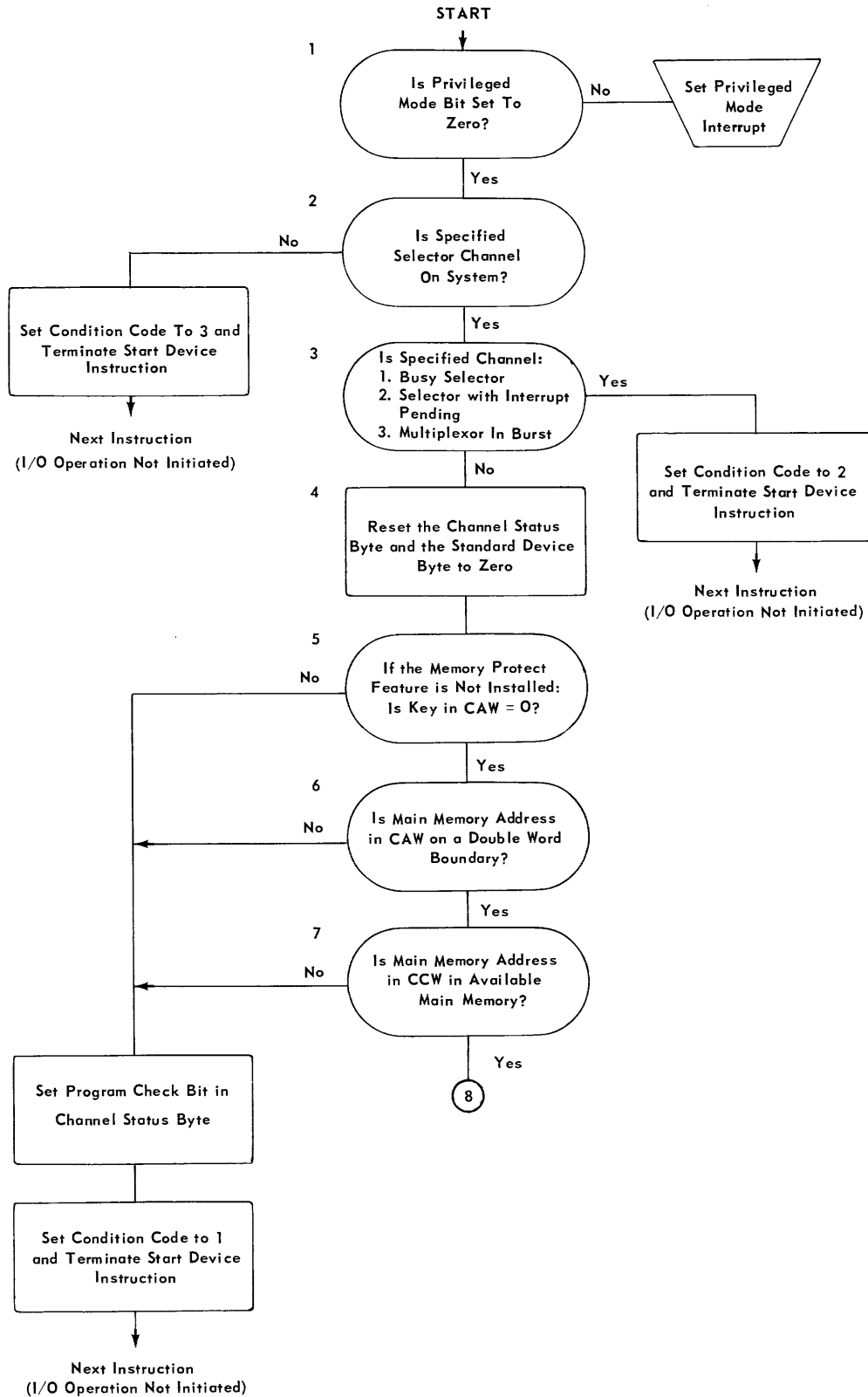


Figure 4. Functional Logic of Start Device Instruction



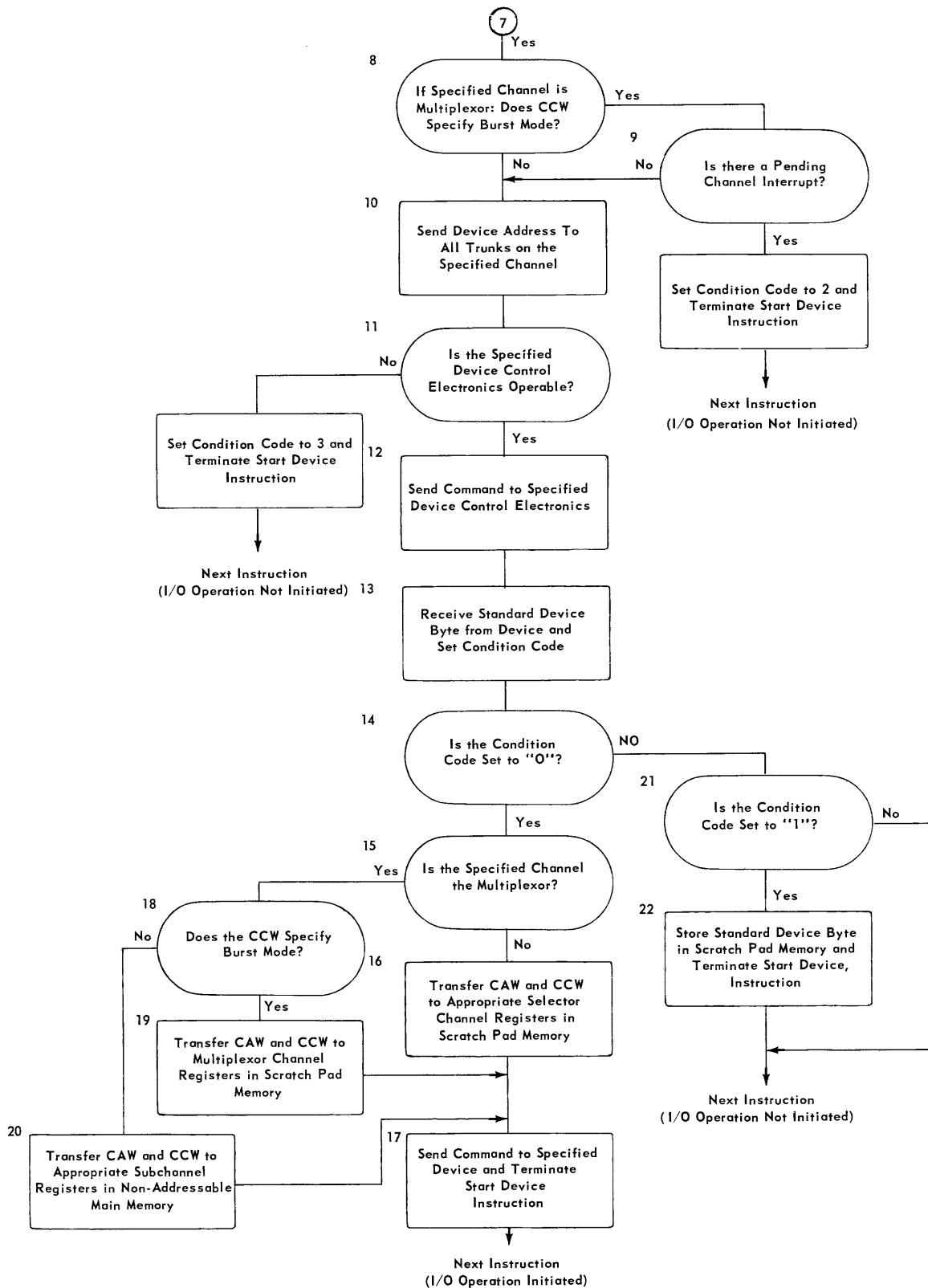


Figure 4. Functional Logic of Start Device Instruction (Cont'd)

- Block 4*     ♦ The channel status byte and the standard device byte for the specified channel are reset to zeros in the appropriate channel registers.
- Block 5*     ♦ If the Memory Protect feature is not installed, the key in the Channel Address Word (CAW) is tested to see if it is equal to zeros. If it is not zeros, the program check bit in the channel status byte is set, the condition code is set to 1, the Start Device instruction is terminated, and program control is transferred to the next instruction. The input/output operation is not initiated.
- Block 6*     ♦ The main memory address in the Channel Address Word is tested to see if it is on a double word boundary. If it is not, the program check bit in the channel status byte is set, the condition code is set to 1, the Start Device instruction is terminated and program control is transferred to the next instruction. The input/output operation is not initiated.
- Block 7*     ♦ The main memory address in the Channel Command Word (CCW) is tested to see if it is within the available main memory for the system. If it is not, the program check bit in the channel status byte is set, the condition code is set to 1, the Start Device instruction is terminated and program control is transferred to the next instruction. The input/output operation is not initiated.
- Block 8*     ♦ If the specified channel is the multiplexor channel, the command code in the Channel Command Word is tested to see if a burst mode operation has been specified.
- Block 9*     ♦ If a burst mode operation has been specified, a test is made to see if there is a terminating interrupt pending on any of the trunks on the multiplexor. If a terminating interrupt is pending, the condition code is set to 2, the Start Device instruction is terminated and program control is transferred to the next instruction. The input/output operation is not initiated.
- Block 10*    ♦ The device address as specified in the Start Device instruction is sent to all trunks on the addressed channel.
- Block 11*    ♦ A test is made to see if the specified device control electronics is operable. The device control electronics has 50 microseconds to signal the processor that it is operable. If it does not, the condition code is set to 3, the Start Device instruction is terminated and program control is transferred to the next instruction. The input/output operation is not initiated.
- Block 12*    ♦ If the specified device control electronics is operable, the command code from the Channel Command Word is sent to the specified device control electronics.

- Block 13** ♦ After receiving the command code, the device control electronics sends the standard device byte to the processor. This standard device byte is *not* stored in the channel registers in scratch-pad memory. It is used to set the proper condition code as follows:

| Condition Code | Definition  |
|----------------|---|
| 3              | Device control electronics is inoperable.   |
| 2              | A termination interrupt pending condition exists in the device control electronics on the multiplexor channel.              |
| 2              | The device control electronics is busy working with the specified device.   |
| 2              | The device control electronics is busy working with a device other than the one specified.                                  |
| 1              | An external device request interrupt pending condition exists in the device control electronics on the multiplexor channel. |
| 1              | The specified device is busy but the device control electronics is not busy (i.e. tape rewinding, off-line seek).           |
| *1             | The specified device is inoperable.   |
| 0              | The specified device and control electronics is available.  |

\* If the command is a Sense, the condition code is set to 0 permitting the operation to be initiated.

- Block 14** ♦ A test is made to see if the condition code is set to 0 (input/output operation can be initiated).
- Block 15** ♦ If the condition code is zero, a test is made to see if the specified channel is the multiplexor channel.
- Block 16** ♦ If the specified channel is a selector channel, the channel address word is fetched from main memory locations 72 through 75 and stored in the appropriate channel address register. Using the main memory address specified in the CAW, the Channel Command Word is fetched from main memory and stored in the appropriate channel command registers.
- Block 17** ♦ The command is sent to the specified device control electronics and the Start Device is terminated (with the condition code set to 0). The input/output operation is initiated and proceeds under control of the appropriate channel and registers in scratch-pad memory and non-addressable main memory (multiplexor devices). Normal program execution of the next instruction continues simultaneously with the input/output operation.
- Block 18** ♦ If the specified channel is the multiplexor channel, the command code in the Channel Command Word is tested to see if a burst mode operation has been specified.
- Block 19** ♦ If a burst mode operation has been specified, the Channel Address Word is fetched main memory locations 72 through 75 and stored in the channel address register for the multiplexor channel. Using the main memory address specified in the CAW, the Channel Command Word is fetched and stored in the channel command registers for the multiplexor channel.

- Block 20* ♦ If a burst mode operation has not been specified, the Channel Address Word and the Channel Command Word are fetched from main memory and stored in the subchannel registers in non-addressable main memory for the device specified.
- Block 21* ♦ If the condition code is not set to 0, a test is made to see if the condition code is set to 1.
- Block 22* ♦ If the condition code is set to 1, the standard device byte is transferred to the channel registers for the channel specified, the Start Device instruction is terminated and program control is transferred to the next instruction. The input/output operation is not initiated.

*Notes on Start Device Instruction*

1. The channel status byte and the standard device byte are not stored if the condition codes are 0, 2, 3.
2. If the specified channel and device can be initiated ( $CC = 0$ ) the contents of the Channel Address Word and Channel Command Word are loaded into the appropriate channel registers and the command is sent to the device. The legality of the command is not determined at initiation time. If the device gets an illegal command, the operation is terminated and a channel interrupt occurs. The standard device byte (stored in the appropriate channel registers when the interrupt is taken) indicates a secondary indicator. A Sense command must be issued to bring the Sense byte(s) into main memory. The Sense byte(s) indicate the illegal operation.
3. If execution of this instruction causes the channel status byte or the standard device byte to be stored, the program must inhibit interrupts on this channel until the status byte has been analysed or moved from the channel registers. If interrupts are permitted and one occurs the standard device byte and the channel status byte are destroyed.

**Halt Device Instruction**

- ♦ The Halt Device instruction is a privileged operation and can be executed only if the mode bit (bit position 15 of the Interrupt Status register) for the current state is set to 0. This instruction is executed in the normal mode. Continuation of program execution is delayed until termination is accepted by the device control electronics. When the device control electronics receives the termination, it causes a channel interrupt to occur. Both the channel number and the device number must be specified in the instruction. Because the Channel Address Word is not referred to by the Halt Device instruction, both the Channel Address Word and a Channel Command Word are not required.

Upon execution of a Halt Device instruction, the following events occur (see figure 5).

- Block 1*     ♦ If the privileged mode bit (bit position 15 of the Interrupt Status register) for the current state is not set to zero, the privileged operation bit is set in the Interrupt Flag register and an interrupt occurs (if permitted).
- Block 2*     ♦ If the specified channel is a selector channel which is not available on the system, the condition code is set to 3, the Halt Device instruction is terminated and program control is transferred to the next instruction.
- Block 3*     ♦ If the specified channel is a selector channel that is busy or if the specified channel is the multiplexor that is operating in the burst mode, the Chain Command (CC) flag in CCR-II is reset, the device control electronics is told to set an end condition, the condition code is set to 2, the Halt Device instruction is terminated, and program control is transferred to the next instruction.
- Notes:*
1. Setting an end condition causes the device to be halted on servicing the next data transfer (see Servicing a Data Transfer).
  2. The Chain Command flag must be reset to suppress chaining during termination (see Chaining and End Servicing section, below).
- Block 4*     ♦ If the specified channel is not the multiplexor channel, the condition code is set to 0, the Halt Device instruction is terminated and program control is transferred to the next instruction.
- Block 5*     ♦ If the specified channel is the multiplexor channel, the channel status byte and the standard device byte are reset to zeros in the multiplexor channel registers.
- Block 6*     ♦ The device address as specified in the Start Device instruction is sent to all trunks on the multiplexor channel.
- Block 7*     ♦ A test is made to see if the specified device control electronics is operable. The device control electronics has 50 microseconds to signal the processor that it is operable. If it does not the condition code is set to 3, the Halt Device instruction is terminated and program control is transferred to the next instruction.
- Block 8*     ♦ If the specified device control electronics is operable, it sends the standard device byte to the processor. This standard device byte is *not* stored in the channel registers. It is used to set the proper condition code as follows:

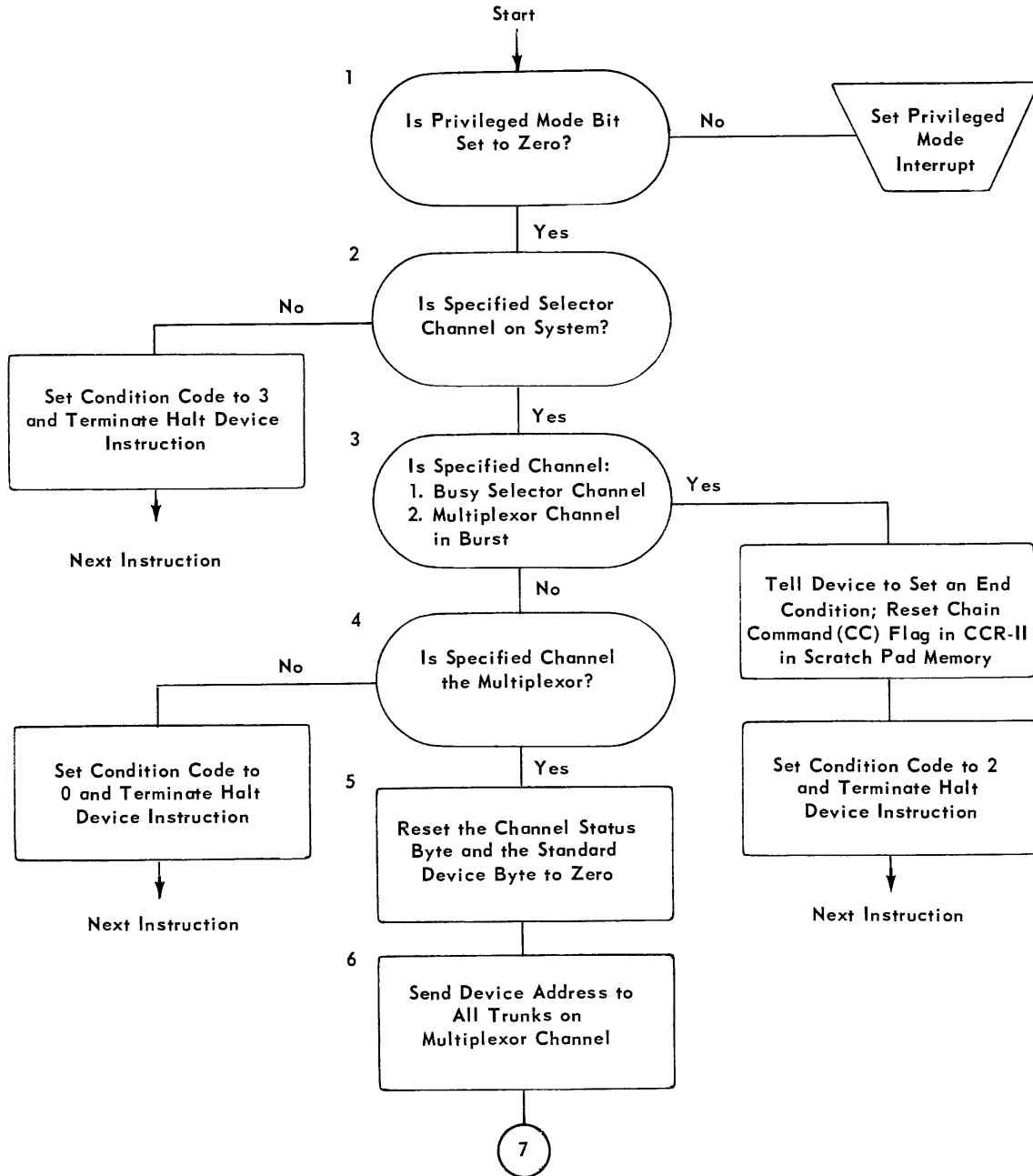


Figure 5. Functional Logic of Halt Device Instruction

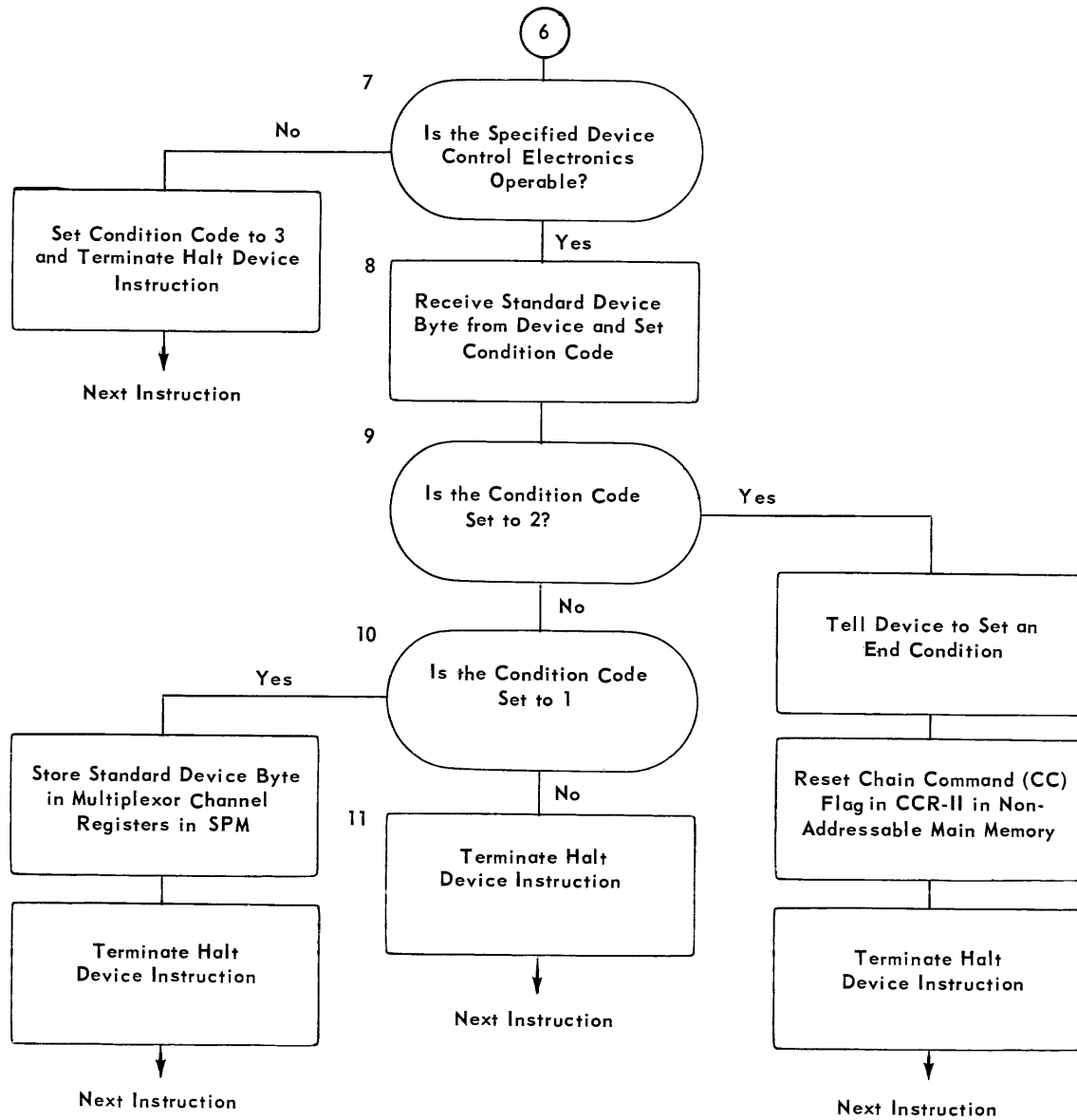


Figure 5. Functional Logic of Halt Device Instruction (Cont'd)

Block 8  
(Cont'd)

| Condition Code | Definition  |
|----------------|---|
| 3              | Device control electronics is inoperable.   |
| 0              | A termination interrupt pending condition exists in the device control electronics.                               |
| 2              | The device control electronics is busy working with the specified device.   |
| 0              | The device control electronics is busy working with a device other than the one specified.                        |
| 1              | An external device request interrupt pending condition exists in the device control electronics.                  |
| 1              | The specified device is busy but the device control electronics is not busy (i.e. tape rewinding, off-line seek). |
| 1              | The specified device is inoperable.   |
| 0              | The specified device and control electronics is available.  |

## Block 9

◆ A test is made to see if the condition code is set to 2 (input/output operation can be terminated). If it is, the device control electronics is told to set an end condition, the Chain Command (CC) flag in CCR-II in the appropriate sub-channel register is reset and control is transferred to the next instruction.

## Notes:

1. Setting an end condition causes the device to be halted on servicing next data transfer (see Servicing a Data Transfer).
2. The Chain Command flag must be reset to suppress chaining during termination (see Chaining and End Servicing section, below).

## Block 10

◆ If the condition code is set to 1, the standard device byte is transferred to the assembly/status registers for the multiplexor channel, the Halt Device instruction is terminated and program control is transferred to the next instruction.

## Block 11

◆ If the condition code is not set to 1 (it is 0, 3) the Halt Device instruction is terminated and program control is transferred to the next instruction.

## Notes on Halt Device instruction:

1. The channel status byte is not stored as a result of this operation. However, the incorrect length bit in the channel status byte may be set.
2. The standard device byte is not stored if the condition codes are 0, 2, 3.
3. If an interrupt pending (termination or external device request) condition exists on a specified selector channel, the condition code is set to zero.



Block 11  
(Cont'd)

4. The channel and device are terminated at the next data service request (see Servicing a Data Transfer).
5. The Channel Address Word (CAW) and Channel Command Word (CCW) are not used by this instruction.
6. If execution of this instruction causes the standard device byte to be stored in the multiplexor channel registers, the program must inhibit interrupts from the multiplexor channel until the standard device byte has been analysed or moved from the channel registers. If interrupts are permitted and one occurs, the standard device byte is destroyed.

**Test Device  
Instruction**

◆ The status of an input/output device can be tested by executing a Test Device instruction. The Test Device instruction is a privileged operation and can be executed only if the mode bit (bit position 15 of the Interrupt Status register for the current state) is set to 0. This instruction is executed in the normal mode. Continuation of program execution is delayed until the instruction is terminated.

Both the channel number and the device number must be specified in the instruction. Because the Channel Address Word is not referred to by the Test Device instruction, the Channel Address Word and a Channel Command Word are not required.

Upon execution of a Test Device instruction, the following events occur (see figure 6).

Block 1

◆ If the privileged mode bit (bit position 15 of the Interrupt Status register) for the current state is not set to 0, the privileged operation bit is set in the Interrupt Flag register and an interrupt occurs, if permitted.

Block 2

◆ If the specified channel is a selector channel that is not available on the system, the condition code is set to 3, the Test Device instruction is terminated and program control is transferred to the next instruction.

Block 3

◆ If the specified channel is a selector channel that is busy or has on interrupt pending (termination or external device request) ; or if the specified channel is the multiplexor that is operating in the burst mode, the condition code is set to 2, the Test Device instruction is terminated and program control is transferred to the next instruction.

Block 4

◆ The channel status byte and the standard device byte for the specified channel are reset to zeros in the appropriate channel registers.

Block 5

◆ The device address as specified in the Test Device instruction is sent to all trunks on the addressed channel.

Block 6

◆ A test is made to see if the specified device control electronics is operable. The device control electronics has 50 microseconds to signal the processor that it is operable. If it does not, the condition code is set to 3, the Test Device instruction is terminated and program control is transferred to the next instruction.

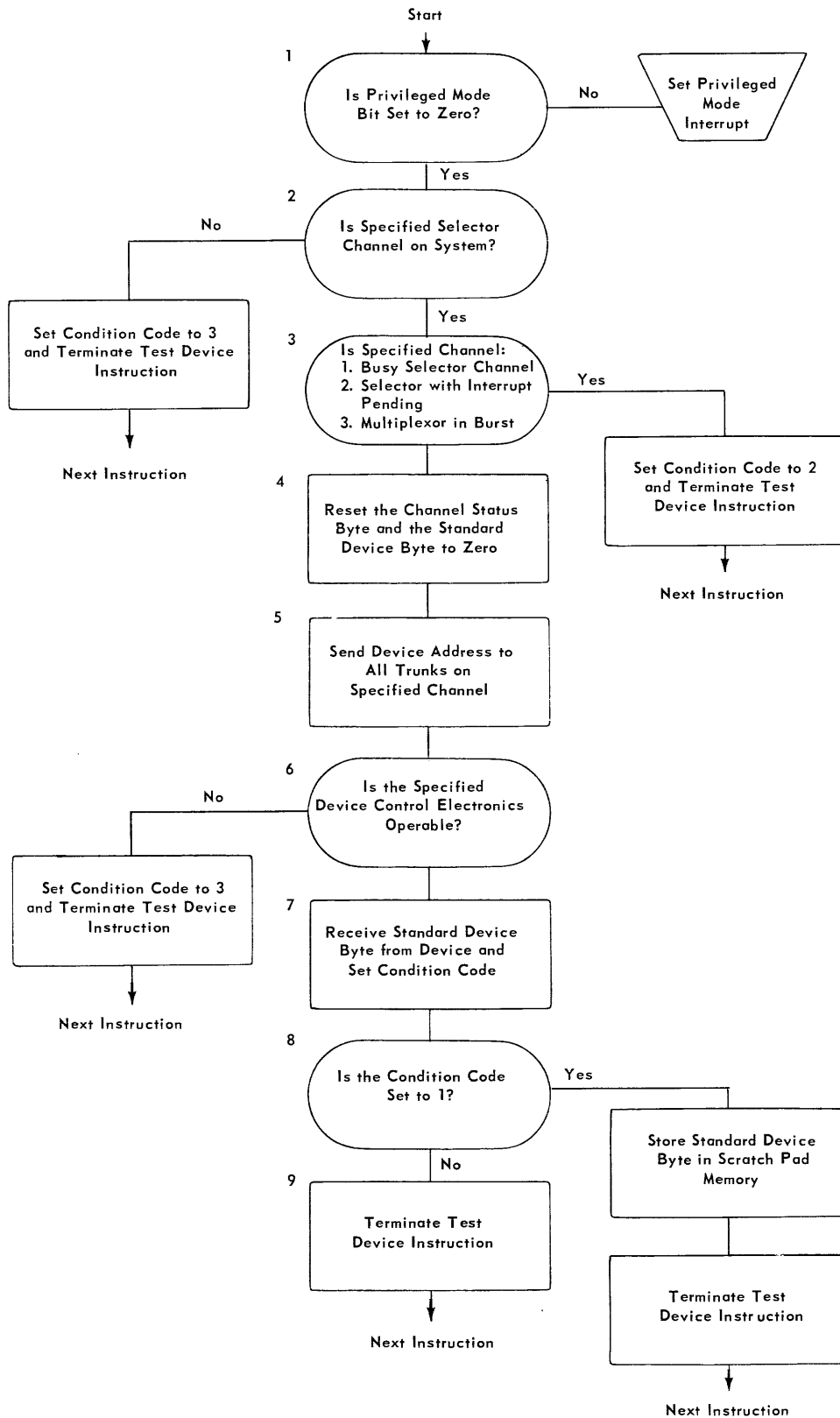


Figure 6. Functional Logic of Test Device Instruction

- Block 7** ♦ If the specified device control electronics is operable, it sends the standard device byte to the processor. This standard device byte is *not* stored in the channel registers. It is used to set the proper condition code as follows:

| Condition Code | Meaning   |
|----------------|---|
| 3              | Device control electronics is inoperable.   |
| 2              | A termination interrupt pending condition exists in the device control electronics on the multiplexor channel.              |
| 2              | The device control electronics is busy working with the specified device.   |
| 2              | The device control electronics is busy working with a device other than the one specified.                                  |
| 1              | An external device request interrupt pending condition exists in the device control electronics on the multiplexor channel. |
| 1              | The specified device is busy but the device control electronics is not busy (i.e. tape rewinding, off-line seek).           |
| 1              | The specified device is inoperable.   |
| 0              | The specified device and control electronics is available.  |

- Block 8** ♦ A test is made to see if the condition code is set to 1. If it is, the standard device byte is transferred to the channel registers for the channel specified, the Test Device instruction is terminated and program control is transferred to the next instruction.

- Block 9** ♦ If the condition code is not set to 1, the Test Device instruction is terminated and control is transferred to the next instruction.

*Notes on Test Device Instruction:*

1. The channel status byte is not stored as a result of this operation.
2. The standard device byte is not stored if the condition codes are 0, 2, or 3.
3. The Channel Address Word (CAW) and Channel Command Word (CCW) are not used by this instruction.
4. If execution of this instruction causes the standard device byte to be stored in the multiplexor channel registers, the program must inhibit interrupts from the multiplexor channel until the standard device byte has been analysed or moved from the channel registers. If interrupts are permitted and one occurs, the standard device byte is destroyed.

- Check Channel Instruction** ♦ The status of an input/output channel can be tested by executing a Check Channel instruction. The Check Channel instruction is a privileged operation and can only be executed if the mode bit (bit position 15 of the Interrupt Status register for the current state) is set to 0. This instruction is executed in the normal mode. Continuation of program execution is delayed until the instruction is terminated.

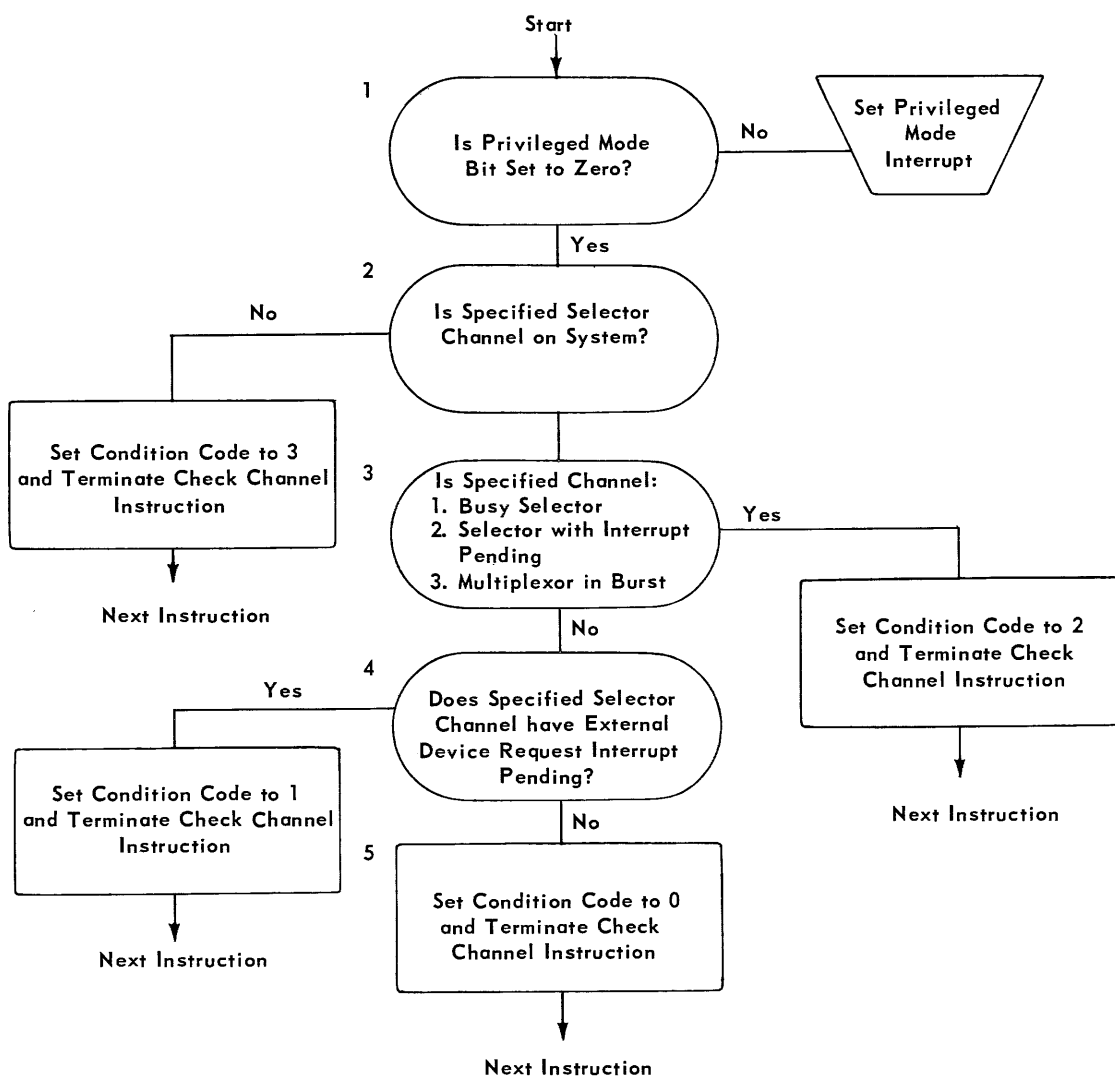


Figure 7. Functional Logic of Check Channel Instruction

**Check Channel instruction**  
(Cont'd)

Only the channel number must be specified in the instruction. Because the Channel Address Word is not referred to by the Check Channel instruction, the Channel Address Word and a Channel Command Word are not required.

Upon execution of a Check Channel instruction, the following events occur (see figure 7).

- Block 1*     ♦ If the privileged mode bit (bit position 15 of the Interrupt Status register) for the current state is not set to 0, the privileged operation bit is set in the Interrupt Flag register and interrupt occurs if permitted.
- Block 2*     ♦ If the specified channel is a selector channel that is not available on the system, the condition code is set to 3, the Check Channel instruction is terminated and program control is transferred to the next instruction.
- Block 3*     ♦ If the specified channel is a selector channel that is busy or has a termination interrupt pending; or if the specified channel is the multiplexor that is operating in the burst mode, the condition code is set to 2, the Check Channel instruction is terminated and program control is transferred to the next instruction.
- Block 4*     ♦ If the specified channel is a selector channel that has an external device request interrupt pending, the condition code is set to 1, the Check Channel instruction is terminated and program control is transferred to the next instruction.
- Block 5*     ♦ If the specified channel is a selector channel that is not busy and has no interrupts pending; or is the multiplexor channel that is not operating in the burst mode, the condition code is set to 0, the Check Channel instruction is terminated and program control is transferred to the next instruction.

*Notes on Check Channel instruction:*

1. The channel status byte and the standard device byte are never stored by this instruction.
2. The Channel Address Word (CAW) and the Channel Command Word (CCW) are not used by this instruction.

**INPUT/OUTPUT STATUS INDICATORS**

♦ Three levels of status information are available to the program to control input/output operation. The first pertains to the setting of the condition code when an input/output instruction is issued. The second level provides more detailed information by storing the channel status byte and the standard device byte in the appropriate input/output channel registers in scratch-pad memory. The third level of status information is generated by, and stored in, the device control electronics until a Sense command is issued. At that time the status information (Sense bytes) are transferred to main memory similar to a data transfer.

**Condition Code**

♦ The condition code is set by the input/output instructions and can be tested by the Branch On Condition instruction. It should be noted that the condition code settings indicate the result of the input/output instructions only. They do not indicate the results of the input/output operation. Condition Code settings for all input/output instructions are as follows:

Condition Code  
(Cont'd)

## Start Device Instruction Condition Code Settings

| Condition Code | I/O Operation Initiated | Meaning   |
|----------------|-------------------------|---|
| 0              | Yes                     | <ol style="list-style-type: none"> <li>1. The device control electronics and the device specified are available.</li> <li>2. The Start Device instruction specifies a Sense command to a device that is inoperable.</li> </ol>  |
| 1              | No                      | <p>This condition code indicates that either the channel status byte or the standard device byte has been stored in the channel registers for the specified channel.</p> <p>The channel status byte is stored under the following conditions:</p> <ol style="list-style-type: none"> <li>1. A parity error occurs while accessing the Channel Address Word or a Channel Command Word. The channel control check bit in the channel status byte is set.</li> <li>2. The Memory Protect feature is not installed and the key in the CAW is not zero. The program check bit in the channel status byte is set.</li> <li>3. The main memory address specified in the CAW is not on a double word boundary. The program check bit in the channel status byte is set.</li> <li>4. The main memory address in the CCW specifies an address outside the available memory for the system. The program check bit in the channel status byte is set.</li> </ol> <p>The standard device byte is stored under the following conditions:</p> <ol style="list-style-type: none"> <li>1. The specified device control electronics on the multiplexor channel indicates that a device request interrupt pending condition is present. The external device request interrupt pending bit in the standard device byte is set.</li> <li>2. The Start Device instruction specifies a command which is other than a Sense command and the addressed device is inoperable. The device inoperable bit in the standard device byte is set.</li> <li>3. The specified device is busy but the device control electronics is not busy (i.e. tape rewinding, off-line seek to a random access device). The device busy bit and the device end bit in the standard device byte is set.</li> </ol> |
| 2              | No                      | <ol style="list-style-type: none"> <li>1. A selector channel is specified that is busy.</li> <li>2. A selector channel is specified that has an interrupt pending (termination or external device request).</li> <li>3. The multiplexor channel is specified and it is operating in burst mode.</li> <li>4. The multiplexor channel is specified and the addressed device control electronics is busy with addressed or non-addressed device.</li> <li>5. The multiplexor channel is specified and the addressed device control electronics has a termination interrupt pending.</li> <li>6. A burst mode operation is directed to the multiplexor and there is a termination interrupt pending on one of the attached device control electronics.</li> </ol>   |
| 3              | No                      | <ol style="list-style-type: none"> <li>1. A selector channel is specified that is not in the system.</li> <li>2. The specified device control electronics is inoperable.</li> </ol>   |

Condition Code  
(Cont'd)

## Halt Device Instruction Condition Code Settings

| Condition Code | I/O Operation Terminated | Meaning  |
|----------------|--------------------------|--|
| 0              | No                       | <ol style="list-style-type: none"> <li>1. The device control electronics or the device specified on the multiplexor channel is not busy. No termination is required.</li> <li>2. A selector channel or the multiplexor channel operating in burst mode is specified and it is not busy. No termination is required.</li> <li>3. The multiplexor channel is specified and the addressed device control electronics has a termination interrupt pending. No termination is required.</li> </ol>  |
| 1              | No                       | <p>This condition code indicates that the specified device is on the multiplexor channel and that the standard device byte has been stored in the channel registers for the multiplexor channel. The channel status byte is never stored.</p> <p>The standard device byte is stored under the following conditions:</p> <ol style="list-style-type: none"> <li>1. The specified device indicates that a device request interrupt pending condition is present. The external device request interrupt pending bit in the standard device byte is set.</li> <li>2. The specified device is busy but the device control electronics is not busy (i.e. tape rewinding). The device busy bit in the standard device byte is set.</li> <li>3. The specified device is inoperable. The device inoperable bit in the standard device byte is set.</li> </ol> |
| 2              | Yes                      | <ol style="list-style-type: none"> <li>1. A selector channel is specified that is busy.</li> <li>2. The multiplexor channel is specified and it is operating in the burst mode.</li> <li>3. The multiplexor channel is specified and the addressed device control electronics and device are busy.</li> </ol>  |
| 3              | No                       | <ol style="list-style-type: none"> <li>1. A selector channel is specified that it is not in the system.</li> <li>2. The specified device control electronics is inoperable.</li> </ol>   |

## Test Device Instruction Condition Code Settings

| Condition Code | Meaning  |
|----------------|--|
| 0              | <p>The device control electronics and the device are available.</p> <p><i>Note:</i> There may be pending interrupts on the multiplexor channel that would prohibit a burst mode operation being initiated.</p>   |
| 1              | <p>This condition code indicates that the standard device byte has been stored in the channel registers for the specified channel. The channel status byte is never stored by this instruction.</p> <p>The standard device byte is stored under the following conditions:</p> <ol style="list-style-type: none"> <li>1. The specified device control electronics on the multiplexor channel indicates that a device request interrupt pending condition is present. The external device request interrupt pending bit in the standard device byte is set.</li> <li>2. The specified device is busy but the device control electronics is not busy (i.e. tape rewinding, off-line seek to a random access device). The device busy bit in the standard device byte is set.</li> <li>3. The specified device is inoperable. The device inoperable bit in the standard device byte is set.</li> </ol> |

**Condition Code**  
*(Cont'd)*
**Test Device Instruction Condition Code Settings (Cont'd)**

| Condition Code | Meaning   |
|----------------|---|
| 2              | <ol style="list-style-type: none"> <li>1. A selector channel is specified that is busy.</li> <li>2. A selector channel is specified that has an interrupt pending (termination or external device request).</li> <li>3. The multiplexor channel is specified and it is operating in burst mode.</li> <li>4. The multiplexor channel is specified and the addressed device control electronics is busy with addressed or non-addressed device.</li> <li>5. The multiplexor channel is specified and the addressed device control electronics has a termination interrupt pending.</li> </ol> |
| 3              | <ol style="list-style-type: none"> <li>1. A selector channel is specified which is not on the system.</li> <li>2. The specified device control electronics is inoperable.</li> <li>3. A device is specified that is not in the system.</li> </ol>   |

**Check Channel Instruction Condition Code Setting**

| Condition Code | Meaning  |
|----------------|--|
| 0              | <ol style="list-style-type: none"> <li>1. The specified selector channel is not busy and has no interrupts pending.</li> <li>2. The specified multiplexor channel is not operating in the burst mode.</li> </ol> |
| 1              | The specified selector channel has an external device request interrupt pending.   |
| 2              | <ol style="list-style-type: none"> <li>1. The specified selector channel is busy or has a terminating interrupt pending.</li> <li>2. The specified multiplexor is operating in the burst mode.</li> </ol>        |
| 3              | A selector channel is specified that is not in the system.   |

**Channel Status Byte**

◆ The channel status byte is stored in Channel Command Register-II (bit positions 8 through 15) for the appropriate channel. It contains information concerning the status of the channel when a channel interrupt occurs, or at the completion of a Start, Halt or Test Device instruction if the condition code indicates that Status is stored. The bit significance of the channel status byte is as follows:

*Bit Position 8* is the program controlled interrupt bit. When set, this bit indicates that a Channel Command Word was accessed which had the program controlled interrupt flag bit set. A channel interrupt occurs for the appropriate channel while the input/output operation specified by the Channel Command Word is being executed.

**Notes:**

1. The program controlled channel interrupt occurs after the first data byte has been transferred.
2. If a Channel Command Word that specifies a burst mode operation is fetched and the program controlled interrupt flag bit is set, the program controlled interrupt does not occur until completion of the burst operation.



**Channel Status Byte**  
 (Cont'd)

*Bit Position 9* is the incorrect length bit. When set, this bit indicates that when the input/output operation was terminated, the byte count specified in the channel command was not equal to the number of bytes received from, or sent to, the input/output device. The incorrect length indicator can be set only if the suppress length indicator flag bit in the channel command word is reset to 0.

The following conditions cause the incorrect length bit to be set:

1. Count High on Input (Read, Read Reverse, Sense). The main memory area specified by the Channel Command Word is not completely filled by transmission from the device. The final byte count in Channel Command Register-II is greater than zero.
2. Count High on Output (Write, Write Control). Data in the main memory area specified by the Channel Command Word is not completely transferred and the device terminated. The final byte count in Channel Command Register-II is greater than zero.

*Notes:*

1. If incorrect length occurs during command chaining and the Suppress Length Indicator flag bit of the current command is reset, the incorrect length bit is set.
2. If incorrect length occurs during the last command of a chain (the Chain Data flag bit is reset), and the Suppress Length Indicator flag of the command is set, the incorrect length bit is not set.

*Bit Position 10* is the program check bit. When set, this bit indicates that a programming error was detected by the channel.

The following conditions cause the program check bit to be set:

1. Invalid Channel Command Word Address. The addressed Channel Command Word is not located on a double word boundary.
2. Invalid Channel Command Word Address. The addressed Channel Command Word is outside the available main memory for the particular installation.
3. Invalid Data Address. The main memory location specified by the data address in the Channel Command Word is outside the available main memory for the particular installation.
4. Invalid Key. The memory protection key in the Channel Address Word is not zero and the system does not have the Memory Protect option installed.

*Notes:*

1. If a program check error occurs during input/output initiation, the operation is suppressed and the program is notified of the error by the condition error setting.
2. If a program check error occurs while the input/output operation is in progress, the operation is terminated and a channel interrupt occurs for the specified channel.
3. If a program check error occurs during chaining (command or data), a channel interrupt occurs for the specified channel and chaining is suppressed.

**Channel Status Byte**  
*(Cont'd)*

*Bit Position 11* is the protection check bit. When set, this bit indicates that the channel tried to store data in a protected main memory area. The operation is terminated and a channel interrupt occurs for the specified channel. If chaining (command or data) is in progress, it is suppressed.

*Bit Position 12* is the channel data check bit. When set, this bit indicates that a parity error was detected in the channel, in main memory, non-addressable main memory or in scratch-pad memory. Reading of characters with bad parity going into memory are replaced with the systems error byte (hexadecimal FF), and the input/output operation is completed. For parity error characters going to a device, (writing) the invalid character is transferred unchanged, the operation is terminated and a channel interrupt occurs for the specified channel. (The transfer of sense byte(s) to memory is not checked for parity.)

*Bit Position 13* is the channel control check bit. When set, this bit indicates that a machine malfunction has occurred affecting the channel controls. Conditions which cause this bit to be set are parity error in the Channel Command Word, data address, or contents of the Channel Command Word. The operation is terminated and a channel interrupt occurs for the specified channel. If chaining (command or data) is in progress, it is suppressed.

*Bit Position 14* is reserved for use by the processor.

*Bit Position 15* is the termination interrupt bit. When set, this bit indicates that a termination interrupt has been effected.

*Important:* The channel status byte is reset *only* when an input/output operation is initiated.

**Standard Device Byte**

◆ The standard device byte is stored in scratch-pad memory in the Assembly/Status register (bit positions 24 through 31) for the appropriate channel. This byte indicates the status of a device after an input/output operation. It may also indicate a device request interrupt.

The standard device byte is automatically stored when:

1. An input/output interrupt is serviced (request or termination).
2. An input/output operation is attempted and the condition code indicates that status bits are stored (channel status byte, standard device byte).

The standard device byte is defined as follows:

*Bit Position 24* is the external device request interrupt pending bit. When set, this bit indicates that a random access device, a data exchange control or a communications device requires servicing.

*Bit Position 25* is the termination interrupt pending bit. When set, this bit indicates that a termination interrupt condition exists in an input/output device.

*Bit Position 26* is the device busy bit. When set, this bit indicates that the specified device is busy and cannot accept another operation.

*Bit Position 27* is the control busy bit. Not applicable.

**Standard Device Byte**  
 (Cont'd)

*Bit Position 28* is the device end bit. When set, this bit indicates that the specified device has terminated. Another operation can be accepted by the device if the device busy bit (26) is not set.

*Bit Position 29* is the secondary indicator bit. When set, this bit indicates that the specified device has additional indicators to be tested. These indicators can be brought into main memory by using the Sense command.

*Bit Position 30* is the device inoperable bit. When set, this bit indicates that the specified device is inoperable.

*Bit Position 31* is the status modifier bit. This bit is used with chaining (command or data). When set, this bit indicates that the next Channel Command Word is skipped. This bit is set as a result of device termination.

**Sense Bytes**

◆ The sense byte, or bytes, are brought into main memory from an input/output device by using the Sense command. These bytes contain status information for the device referred to. The exact status information sent is defined in the Spectra 70 input/output reference manuals for the individual devices.

**CHANNEL  
SERVICING**

◆ The following sections explain in detail the three types of channel servicing which may be performed during input/output operations. They are: servicing a data transfer, end and chain servicing, and interrupt servicing.

Because channel servicing requires that the channel utilize main memory, scratch-pad memory and non-addressable main memory (multiplexor devices), normal mode processing is held off until the servicing has been completed. Consequently, channel servicing is time-shared with normal mode processing. Between service requests, normal mode processing is resumed, or another channel is permitted to service its device(s).

Channel servicing for a device on the multiplexor channel (multiplex mode) requires more time than channel servicing for a device on a selector channel. To balance the system's throughput rate, multiplexor channel servicing is segmented to permit selector channel servicing to break-in if any selector channels require servicing. After all selector(s) demanding service have been satisfied, multiplexor servicing is resumed. This technique insures that the interference to selector channel servicing caused by the multiplexor channel is no greater than that of an additional selector channel.

**Servicing a Data Transfer**

◆ Once an input/output operation has been initiated, it proceeds under control of the appropriate channel and registers in scratch-pad memory and non-addressable main memory (multiplexor devices). When an input/output operation has been initiated and the input/output device is ready to send or receive a data byte, it asks the processor for a service request. When the processor honors this service request, servicing of a data transfer occurs.

Because servicing a data transfer requires that the channel utilize main memory, scratch-pad memory and non-addressable main memory (multiplexor devices), normal mode processing is held off until the servicing has been completed. Servicing of a data transfer is time-shared with normal mode processing. Between service requests, processing is resumed, or another channel is permitted to service its device(s).

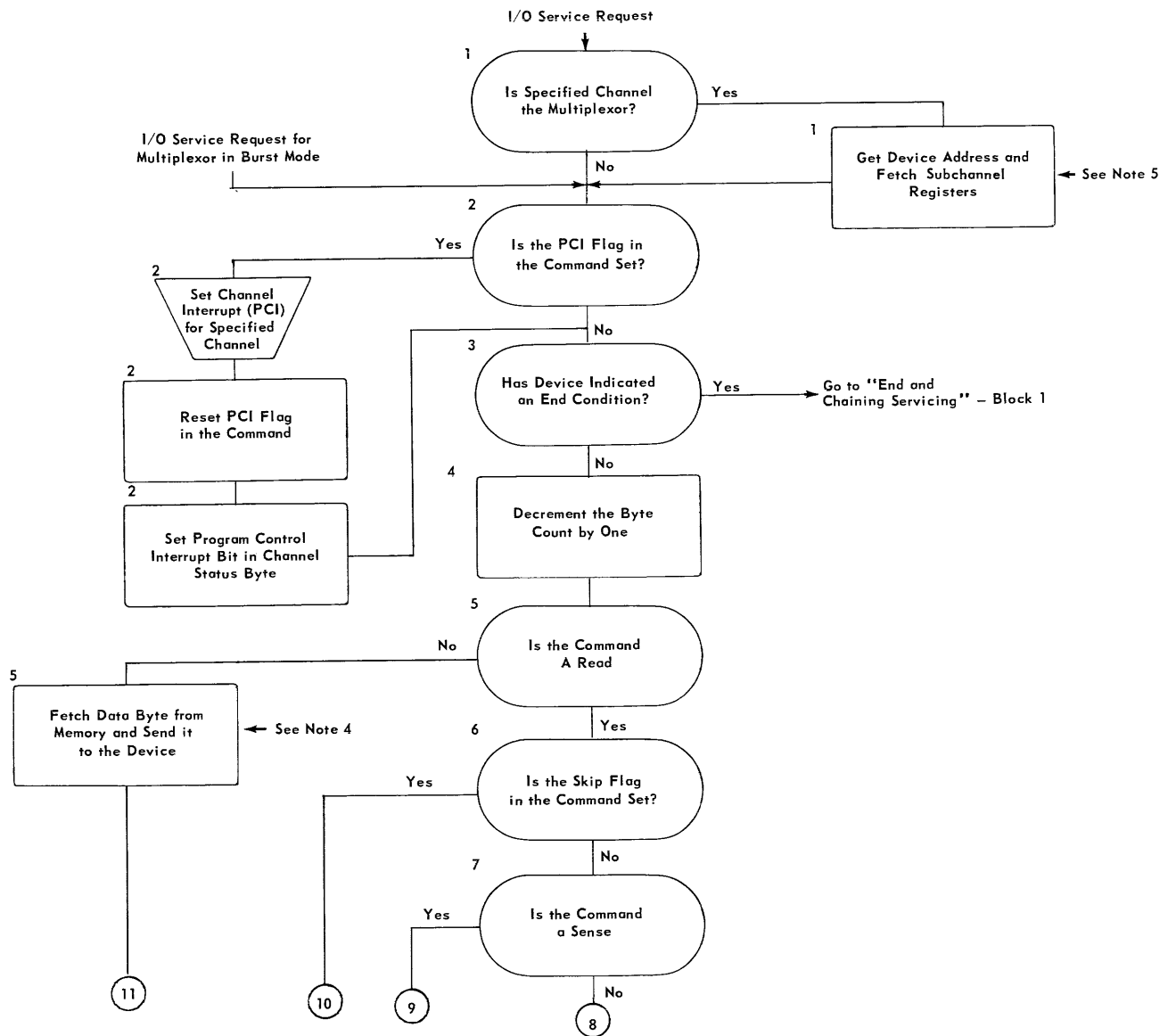


Figure 8. Functional Logic of Servicing a Data Transfer

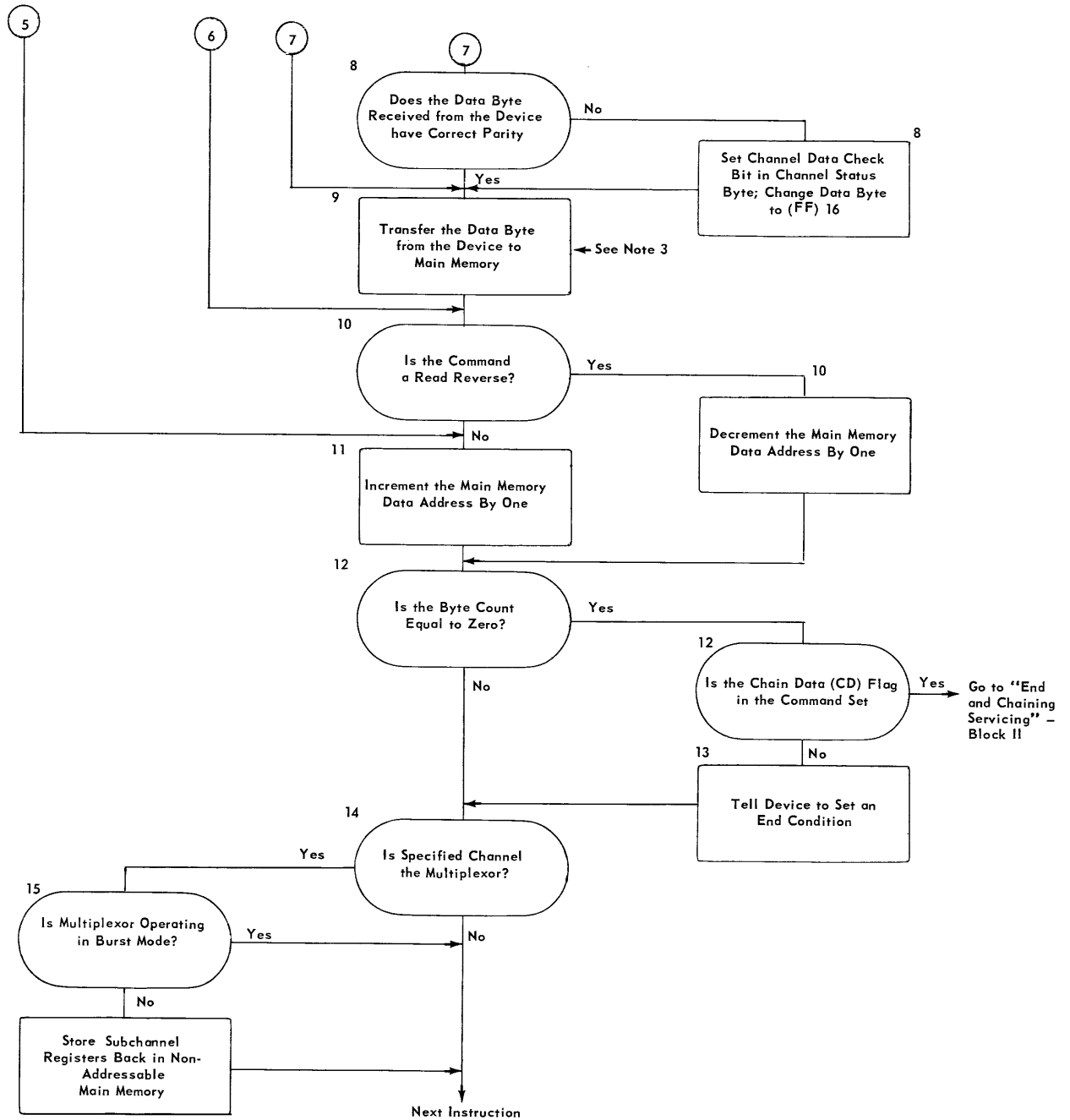


Figure 8. Functional Logic of Servicing a Data Transfer (Cont'd)

# **Servicing a Data Transfer** *(Cont'd)*

If a burst mode operation has been initiated to the multiplexor channel, the channel operates similar to a selector and only one device is serviced. Service requests by devices other than the one operating in burst mode are ignored until the multiplexor channel is operating in the multiplexor mode. This occurs at the conclusion of the burst operation when the last data byte has been serviced (prior to interrupt).

Servicing of a data transfer causes the following events to occur (see figure 8).

## *Block 1*

◆ If the service request comes from a device control electronics connected to the multiplexor channel which is operating in the multiplex mode, the processor gets the device address and fetches the appropriate sub-channel registers in non-addressable main memory. These registers are placed in processor utility registers in scratch-pad memory. (They are *not* sent to the multiplexor channel registers in scratch-pad memory.) If the service request comes from a device control electronics connected to the multiplexor channel which is operating in the burst mode or from a device connected to a selector channel, the appropriate channel registers in scratch-pad memory are used to service the data transfer.

## *Block 2*

◆ A test is made to see if the Program Controlled Interrupt (PCI) flag is set. If it is, the channel interrupt bit is set in the Interrupt Flag register and an interrupt occurs, if permitted. The PCI flag is reset and the program control interrupt bit is set in the channel status byte.

## *Block 3*

◆ A test is made to see if the device control electronics requesting service has indicated an end condition. An end condition is indicated when one of the following occurs:

1. The processor reaches a byte count lapse. If this occurs, the processor tells the device control electronics to indicate an end condition on the next data service request.
2. The device has completed the input/output operation (i.e. a gap is sensed on tape). If this occurs, the device control electronics automatically indicates an end condition. (In this case the byte count is never zero.)

If an end condition has been indicated, the processor goes to End and Chaining Servicing (see figure 9, Block 1).

*Note:* Certain error conditions cause the processor to tell the device control electronics to indicate an end condition on the next data service request (see Notes 3, 4, 5, 6 on Servicing a Data Transfer).

## *Block 4*

◆ If the device control electronics has not indicated an end condition, the byte count is decremented by one.

- Block 5*     ♦ A test is made to see if the command is a read. A read command can be any one of the following:
- Read Forward  
Read Reverse  
Sense
- All other commands (except Transfer in Channel) are write commands. If the command is a write, the data byte is fetched from main memory and sent to the device. Control is then transferred to Block 11.
- Block 6*     ♦ If the command is a read, a test is made to see if the SKIP flag is set. If it is, transfer of the data byte to main memory is bypassed and control is transferred to Block 10.
- Block 7*     ♦ If the SKIP flag is not set, a test is made to see if the command is a Sense. If it is, parity checking of the data byte is bypassed and control is transferred to Block 9.
- Block 8*     ♦ If the command is not a Sense, a test is made to see if the data byte received from the device has correct parity. If it does not, the channel data check bit in the channel status byte is set and the data byte is converted to  $(FF)_{16}$ . The input/output operation continues.
- Block 9*     ♦ The data byte is transferred to the main memory address specified.
- Block 10*    ♦ A test is made to see if the command is a Read Reverse. If it is, the main memory address is decremented by one.
- Block 11*    ♦ If the command is not a Read Reverse, the main memory address is incremented by one.
- Block 12*    ♦ A test is made to see if the byte count has lapsed. If it has, a test is made to see if the Chain Data flag is set. If it is, the processor goes to End and Chaining Servicing (see figure 9, Block 11).
- Block 13*    ♦ If the Chain Data flag is not set, the processor tells the device control electronics to indicate an end condition on the next data service request.
- Block 14*    ♦ A test is made to see if the service request was honored for a device on the multiplexor channel. If it was not, program control continues with the next instruction or with the instruction that was interrupted due to the service request.
- Block 15*    ♦ If the service request was honored for a device on the multiplexor channel, a test is made to see if it is a burst mode operation. If it is not a burst mode operation, the sub-channel registers are sent back to non-addressable main memory. In either case, program control continues with the next instruction or with the instruction that was interrupted due to the service request.

**Servicing a Data Transfer**  
(*Cont'd*)

*Notes on Servicing a Data Transfer:*

1. All input/output data service requests are honored depending on the channel's position in the priority sequence.
2. The following tests occur when a data byte is transferred to main memory:
  - a. The main memory address to which the data byte is to be transferred is tested to see if it is in a memory protected area (Memory Protect feature must be installed). If it is, the protection check bit in the channel status byte is set (no data transfer occurs) and the device control electronics is told to set an end condition on the next data service request (see Block 13).
  - b. The main memory address to which the data byte is to be transferred is tested to see if it is in available main memory for the system. If it is not, the program check bit in the channel status byte is set (no data transfer occurs) and the device control electronics is told to set an end condition on the next data service request (see Block 13).
3. The following tests occur when a data byte is transferred from main memory:
  - a. The main memory address from which the data byte is to be transferred is tested to see if it is in available main memory for the system. If it is not, the program check bit in the channel status byte is set (no data transfer occurs) and the device control electronics is told to set an end condition on the next data service request (see Block 13).
  - b. The data byte to be transferred is checked for correct parity. If parity is not correct, the data check bit in the channel status byte is set and the device control electronics is told to set an end condition on the next data service request (see Block 13).
4. If a main memory parity error occurs while fetching the subchannel registers, the channel control check bit in the channel status byte is set, and the device control electronics is told to set an end condition on the next data service request (see Block 13).
5. If a scratch-pad memory parity error occurs during the servicing of a data transfer, the channel control check bit in the channel status byte is set and the device control electronics is told to set an end condition on the next data service request (see Block 13).



**End and Chaining Servicing**

◆ End and chaining servicing is required when the input/output operation specified by the current command has been completed (normally or abnormally). Entry to this servicing always comes from “servicing a data transfer”. The following conditions cause end and chaining servicing to take place:

1. A device control electronics has indicated an end condition. This end condition is recognized in Servicing a Data Transfer.
2. The byte count in the current command has lapsed and the Chain Data (CD) flag in this command is set. If this condition occurs, entry to End and Chaining Servicing occurs at a point which bypasses the normal end servicing with no chaining and the end servicing with command chaining.

For input/output operations that do not specify chaining, end servicing is used so that the processor can tell the appropriate device control electronics to set an interrupt condition. This interrupt condition is in turn reported to the processor and the appropriate flag in the Interrupt Flag register is set, at which time the interrupt is taken, if permitted.

For input/output operations that specify chaining (command or data), this servicing does one of the following:

1. If the current command specifies command chaining (the CC flag in the command is set) this service is used to fetch the next command in the chain and to send this new command to the input/output device.
2. If the current command specifies data chaining (the CD flag in the command is set) this service is used to fetch the next command in the chain so that the current operation can be continued.

End and Chaining Servicing causes the following events to occur (see figure 9).

*Block 1*

◆ Entry to this block occurs when the input/output device control electronics has indicated an end condition. This end condition is recognized during servicing a data transfer and is generated:

1. automatically by the device, or
2. by the device on command from the processor

The processor receives the standard device byte from the device control electronics. This standard device byte is used by the processor for testing purposes. It is *not* stored in the channel registers.

*Block 2*

◆ The standard device byte is tested to see if the device busy bit is set and the device end bit is reset. This condition normally arises in buffered devices (i.e. card punch, printer) when the buffer has been loaded and the completion of the operation is off-line (no more data has to be sent between the processor and the device control electronics). If this condition exists, the processor tells the device to set another end condition and ask for another service request when the device is no longer busy. Control is then transferred to Block 14.

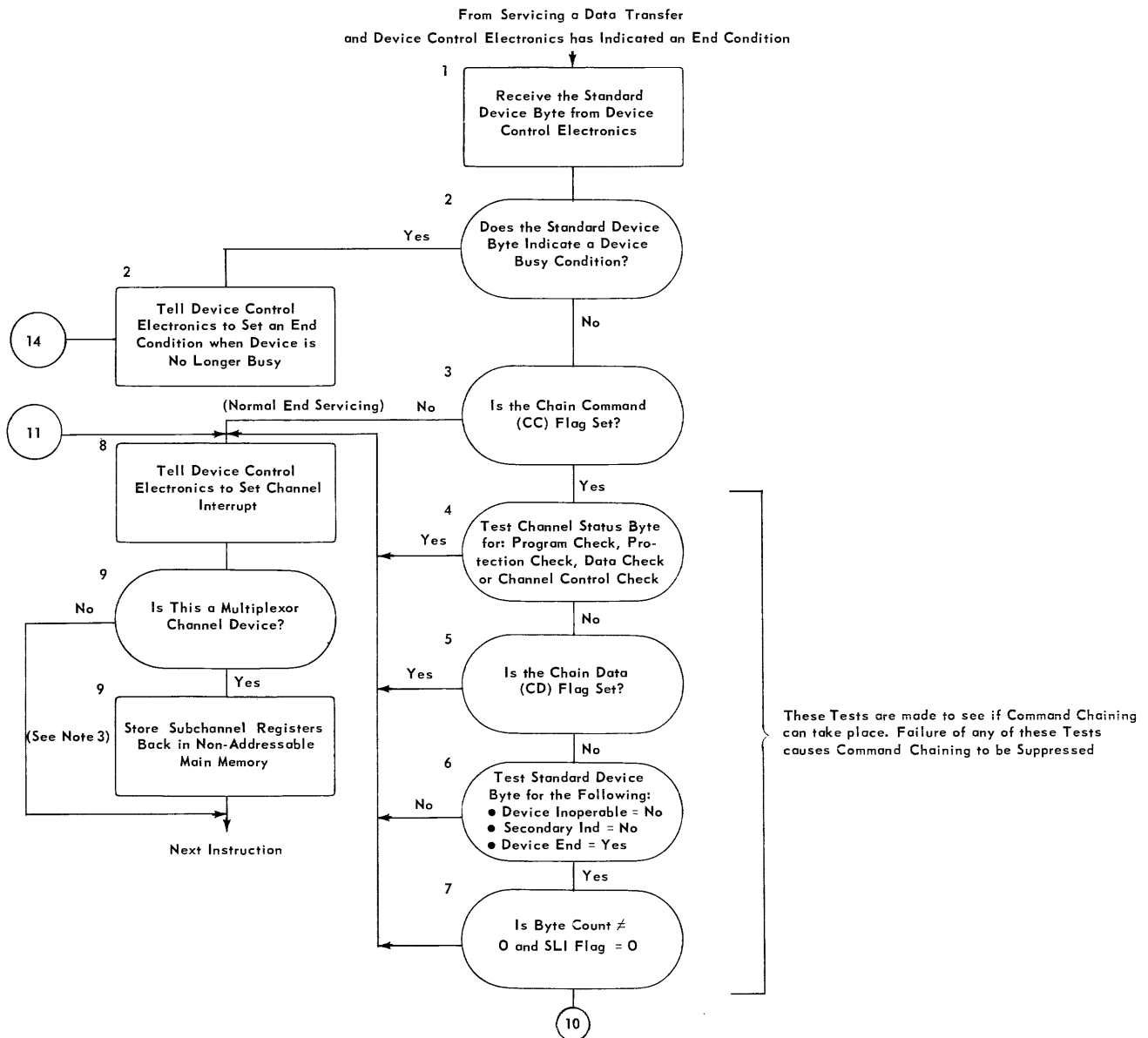


Figure 9. Functional Logic of End and Chaining Servicing

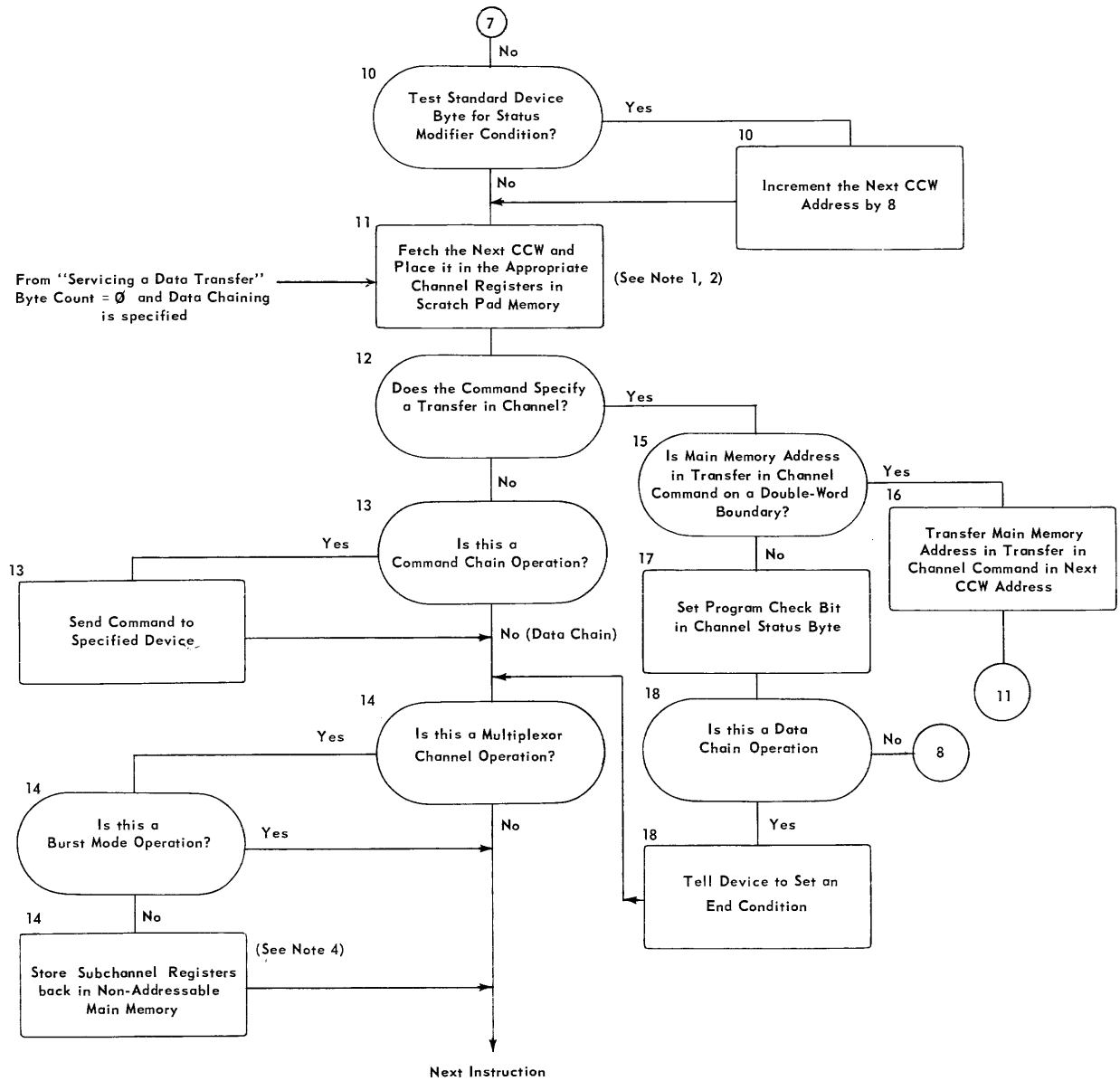


Figure 9. Functional Logic of End and Chaining Servicing (Cont'd)

- Block 3*     ♦ If the device is not busy, a test is made to see if the Chain Command (CC) flag is set. If it is not, control is transferred to Block 8 which causes termination of the command to occur.
- Block 4*     ♦ If the Chain Command (CC) flag is set, a test is made to see if one of the following bits is set in the channel status byte:
- Program Check bit
- Protection Check bit
- Data Check bit (This bit is checked only if the current operation is a write)
- Channel Control Check bit
- If any of the above bits are set (except the data check bit on a Read) control is transferred to Block 8 which causes termination of the command and suppression of command chaining to occur.
- Block 5*     ♦ If none of the bits tested in the channel byte are set, a test is made to see if the Chain Data (CD) flag is set. If the Chain Data flag is set, control is transferred to Block 8 which causes termination of the command and suppression of command chaining to occur.
- Block 6*     ♦ If the Chain Data (CD) flag is not set the standard device byte is tested to see that the following conditions are present:
- Device is operable
- Secondary indicator is not set
- Device end is set
- If any of the above conditions is not present, control is transferred to Block 8 which causes termination of the command and suppression of command chaining to occur.
- Block 7*     ♦ If all of the conditions tested in the standard device byte are present, a test is made to see if the byte count is *not* equal to zero and the Suppress Length Indicator (SLI) flag is equal to zero. If these conditions are present, the program desires an indication of incorrect length, and control is transferred to Block 8 which causes termination of the command and suppression of command chaining to occur.
- Block 8*     ♦ Entry to this block occurs under the following conditions:
- a. A device control electronics has indicated an end condition, the device is not busy and the chain command flag bit is *not* set.
  - b. A device control electronics has indicated an end condition and the chain command flag is set. However, a condition is present which causes command chaining to be suppressed.
- The processor tells the device control electronics is set a channel interrupt condition for the appropriate channel.

- Block 9* ♦ A test is made to see if the device is on the multiplexor channel. If it is, the subchannel registers are sent back to non-addressable main memory. In either case, program control continues with the next instruction or with the instruction that was interrupted due to chaining and/or end servicing.
- Note:* If the operation that was terminated was a burst mode operation, the burst mode is completed at this point and other multiplex mode operations can be directed to devices on the multiplexor channel. The processor does *not* have to wait for the burst mode terminating interrupt to occur.
- Block 10* ♦ Entry to this block occurs when command chaining is to take place. The standard device byte is tested to see if the status modifier bit is set. If it is, the next Channel Command Word (CCW) address is incremented by eight. (The next channel command word in sequence is skipped.)
- Block 11* ♦ In addition to continuing command chaining processing, entry to this block occurs from Servicing a Data Transfer when the following conditions are present:
- The byte count is equal to zero.
  - The Chain Data (CD) flag is set.
- The next Channel Command Word (CCW) is fetched from main memory and placed in the appropriate channel registers. The next Channel Command Word address is incremented by eight.
- Block 12* ♦ A test is made to see if the next command in sequence is a Transfer in Channel command.
- Block 13* ♦ If the command is not a Transfer in Channel command, a test is made to see if this is a command chain or a data chain operation. If it is a command chain operation, the new command is sent to the specified device control electronics. (This is not required if this is a data chain operation.)
- Block 14* ♦ A test is made to see if the chaining servicing has occurred for a device on the multiplexor channel. If it has, a test is made to see if it is a burst mode operation. If it is not a burst mode operation, the subchannel registers are sent back to non-addressable main memory. In all cases, program control continues with the next instruction, or with the instruction that was interrupted due to the chaining servicing.
- Block 15* ♦ If the next command in sequence is a Transfer in Channel command, the main memory address specified by the Transfer in Channel command is tested to see if it is on a double word boundary.
- Block 16* ♦ If the main memory address specified in the Transfer in Channel command is on a double word boundary, this address is placed in the next Channel Command Word address and control is transferred to Block 11 which fetches the CCW specified by the Transfer in Channel command.
- Block 17* ♦ If the main memory address specified in the Transfer in Channel command is *not* on a double word boundary, the program check bit is set in the channel status byte.

## Block 18

◆ A test is made to see if this is a data chain operation. If it is, the device is told to set an end condition on the next data service request and control is transferred to Block 14 to complete the end servicing. If this is a command chain operation (the device has already indicated an end condition) control is transferred to Block 8 where the device control electronics is told to set an interrupt condition.

*Notes On End and Chaining Servicing:*

1. The following test occurs when the next Channel Command Word is fetched:

The main memory address specified is tested to see if it is in available main memory for the system. If it is not, the program check bit in the channel status byte is set; and, if data chaining, the device is told to set an end condition on the next data service request (see Block 2); if command chaining, the device control electronics is told to set a channel interrupt condition (see Block 8).

2. If a main memory parity error occurs when fetching the next Channel Command Word, the channel control check bit in the channel status byte is set; and, if data chaining, the device control electronics is told to set an end condition on the next data service request (see Block 2); if command chaining, the device control electronics is told to set a channel interrupt condition (see Block 8).
3. If a scratch-pad memory parity error occurs when storing the sub-channel registers back in non-addressable main memory the channel control check bit in the channel status byte is set.
4. If a scratch-pad memory parity error occurs when storing the sub-channel registers back in non-addressable main memory, the channel control check bit in the channel status byte is set; and, if data chaining, the device control electronics is told to set an end condition on the next service request (see Block 2); if command chaining, the device control electronics is told to set a channel interrupt condition (see Block 8).

**Interrupt Servicing**

◆ Interrupt servicing occurs when the appropriate flag in the Interrupt Flag register has been set, and the Interrupt Mask register for the current state permits the interrupt and it is taken. This service is required to:

1. Obtain the standard device byte from the device control electronics (if applicable) and store it in the appropriate input/output channel registers.
2. Fetch the appropriate subchannel registers from non-addressable main memory if the interrupt is due to a multiplexor channel device. The subchannel registers are stored in the multiplexor channel registers in scratch-pad memory.

There are three kinds of channel interrupts. They are as follows:

*Programmed Control Interrupt*—This interrupt occurs when a Channel Command Word is fetched and the program controlled interrupt flag bit is

**Interrupt Servicing**  
(Cont'd)

set. This interrupt condition has no effect upon the input/output operation specified by the Channel Command Word. The standard device byte and the subchannel registers are not stored.

*Device Request Interrupt*—This interrupt occurs as a result of a condition arising in an input/output device control electronics. It may occur independent of a processor initiated input/output operation. Examples of this type of interrupt are as follows:

1. A remote processor wishes to send data via a Data Exchange Control. The Data Exchange Control initiates the channel interrupt. (This interrupt occurs independent of a processor initiated input/output operation).
2. The processor initiates an off-line seek to a random access device. When the seek is complete, the random access device control electronics initiates a channel interrupt. (This interrupt occurs in conjunction with a processor initiated input/output operation).

When an external device request interrupt occurs, the standard device byte and the subchannel registers (if a multiplexor device) are stored in the appropriate input/output channel registers.

*Terminating Interrupt*—This interrupt occurs when an input/output operation initiated by the processor has terminated. When this interrupt occurs, the standard device byte and the subchannel registers (if a multiplexor device) are stored in the appropriate input/output channel registers. This is the final servicing of the channel and device. At the completion of this servicing, the channel is free to accept another operation. The contents of the input/output channel registers must be utilized by the program before another operation is initiated. (When another operation is initiated, the contents of these registers are altered.) The following information is available in the input-output channel registers for interrogation by the program:

Channel status byte  
Standard device byte  
Byte count  
Address of next CCW  
Low-order 4 bits of the command code  
Device number

Interrupt servicing causes the following events to occur (see figure 10).

- |                |   |
|----------------|---|
| <i>Block 1</i> | ◆ The device control electronics is asked for the address of the device requiring interrupt servicing.  |
| <i>Block 2</i> | ◆ A test is made to see if the device control electronics is operable. The device control electronics has 50 microseconds to signal the processor that it is operable. If it does not, the processor generates a standard device byte of all zeros. Control is then transferred to Block 4. |
| <i>Block 3</i> | ◆ If the device control electronics is operable, it sends the standard device byte to the processor.  |

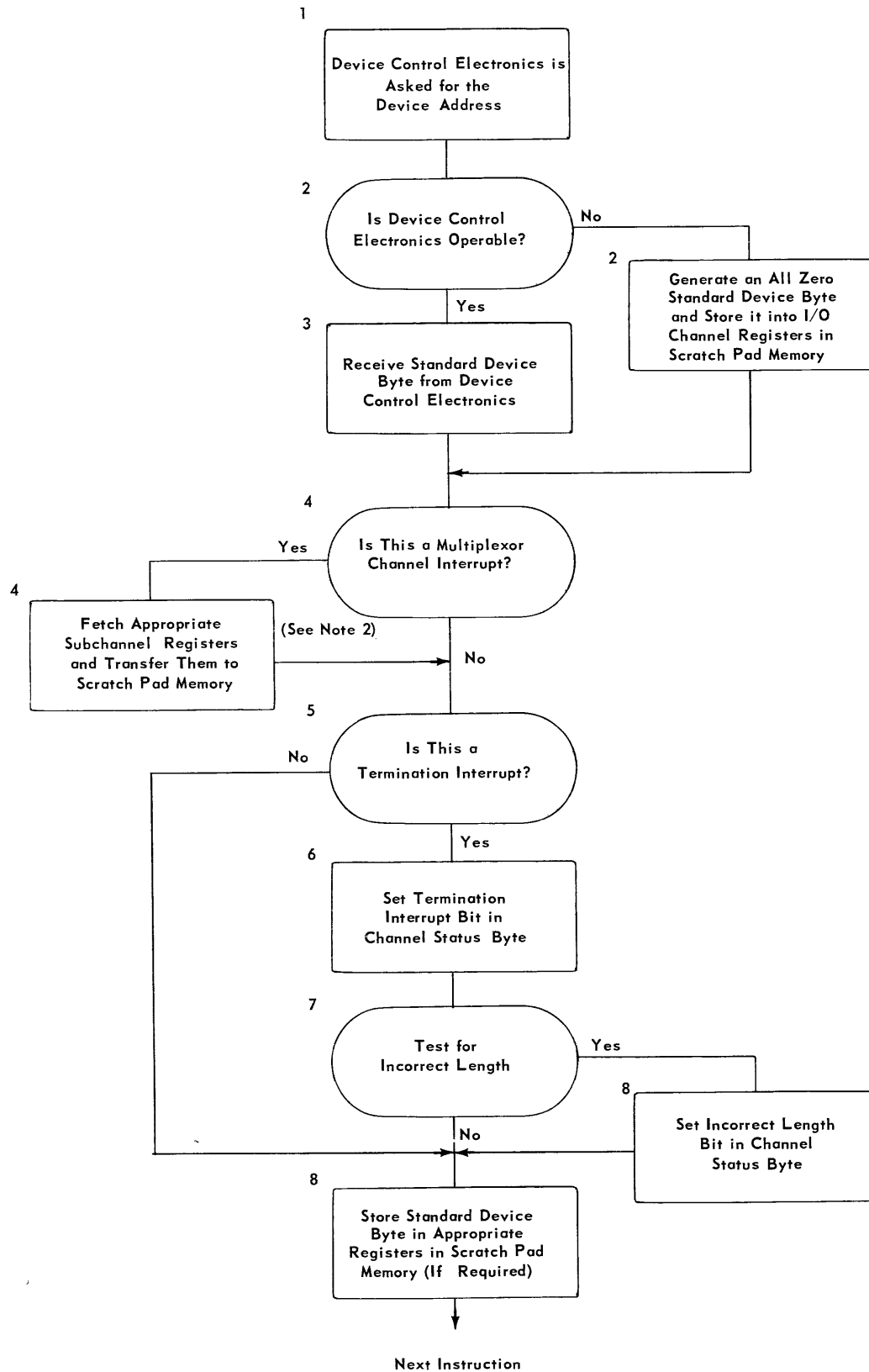


Figure 10. Functional Logic of Interrupt Servicing



- Block 4*     ♦ If the service request comes from a device control electronics connected to the multiplexor channel, the processor uses the device address to fetch the appropriate subchannel registers in non-addressable main memory. The subchannel registers are stored in the input/output channel registers in scratch-pad memory for the multiplexor channel.
- Block 5*     ♦ A test is made to see if this is a terminating interrupt. If it is not (it is a program controlled or a device request interrupt) control is transferred to Block 8.
- Block 6*     ♦ If the interrupt is a terminating interrupt, the termination interrupt bit in the channel status byte is set.
- Block 7*     ♦ A test is made to see if the byte count is *not* equal to zero and the Suppress Length Indicator (SLI) flag is equal to zero. If these conditions are present, the program desires an indication of incorrect length and the incorrect length bit in the channel status byte is set.
- Block 8*     ♦ The standard device byte is stored in the appropriate input/output channel registers and program control continues with the next instruction.
- Note:* On the 70/55 Processor, if the interrupt is a program controlled interrupt, the standard device byte is not stored.

*Notes on Interrupt Servicing:*

1. The device address is always stored in the input/output channel registers in scratch-pad memory if the interrupt is due to a device connected to the multiplexor channel. If the interrupt is due to a device on a selector channel, the device address is stored *only* if it is a device request interrupt.
2. If a main memory parity error occurs when fetching the subchannel registers, the channel control check bit in the channel status byte is set.

## MULTI-PROCESSOR INSTALLATION

### INTRODUCTION

◆ Installations where more than one computer shares peripheral equipment or work loads require extra machine-program communications. To enable this rapid signaling between processors independent of input/output operations the Direct Control feature is provided.

To signal a receiving processor (or processors) a Write Direct instruction is used to effect an external interrupt in the receiving processor. To enable the receiving processor to honor this external interrupt and complete the transfer, a Read Direct instruction is used (refer to Privileged Instructions section). This Write Direct action of one processor to another is analogous to a Supervisor Call instruction and corresponding interrupt of a user's program to the Interrupt Control State ( $P_3$ ).

The Direct Control feature is identical on the 70/35, 70/45 and 70/55 Processors, therefore permitting all three of the processors to be connected in any combination of up to six. Some typical cases for which this feature is used are:

- Request use of a control file.
- Notify that file access has been completed.
- Notify back-up system that a processor machine failure has been detected.
- Notify back-up system that a processor power failure has been detected.
- Request assistance because of program overload.
- Request for task assignments.

### OPERATIONAL CHARACTERISTICS

◆ The 8-bit data byte transmitted from the out line of one processor to the in line of a second processor in a multi-processor installation by means of the Direct Control feature provides 256 code combinations. The code sets can be any required by the program including EBCDIC and ASCII with code interpretation being performed by the program.

When a transmitting processor issues a Write Direct instruction, an external interrupt is set in the receiving processor (specified by the I-Field of the Write Control instruction) in response to the signal. To service the interrupt, the receiving processor issues a Read Direct instruction to accept the control byte and then issues a Write Direct with an acknowledgement code to the transmitting processor. (Write Direct of an acknowledgement code does not require a return acknowledgement.) When an acknowledgement has been received from each of the receiving processors (if more than one connected), the transmitting processor may execute another transmission.

In the event of power failing in a processor, interrupt occurs to processor state  $P_4$ . In a multi-processor installation with the Direct Control feature, the failing processor issues a Write Direct instruction with a data byte of all zero bits to all processors it is connected to in the system.

*Note:* The Direct Control feature does not provide error checking on the data transmitted. When checking is required, it must be performed by program.

**DIRECT CONTROL  
INTERFACE**

◆ The Direct Control interface connects from two to six processors into a multi-processor complex. Each of the processors can have up to six direct control trunks which contain the signal lines that transmit and receive the direct control information. These signal lines function as follows:

**Static Out Lines**

◆ The Static Out lines are logically identical (common) on all trunks, (information on one trunk is identical to information of all other trunks). The state of these Static Out lines is established when a Write Direct instruction is executed and remains static until altered by a subsequent Write Direct instruction. Parity is not generated or checked on these lines. (See Write Direct instruction.)

**Static In Lines**

◆ The Static In lines provide the means for the receiving processor to receive 8-bit bytes of data from other transmitting processors via their Static Out lines. Each trunk may be uniquely sampled by a Read Direct instruction which specifies the desired trunk. (See Read Direct instruction.)

**Signal Out Line**

◆ The Signal Out line provides a signal to the other processors upon execution of a Write Direct instruction. The Direct Control Trunks (DCT) whose Signal Out lines are signaled is specified by the I-Field pattern of the instruction.

**External Signal In Line**

◆ The External Signal In line provides the means for receiving a signal from other processors via their Signal Out lines. The External Signal In line is logically connected to the external signal interrupt flag associated with each Direct Control Trunk (DCT) as indicated:

| <i>Trunk Signaled</i> | <i>External Interrupt Flag</i> |
|-----------------------|--------------------------------|
| DCT #1                | 1                              |
| DCT #2                | 2                              |
| DCT #3                | 3                              |
| DCT #4                | 4                              |
| DCT #5                | 5                              |
| DCT #6                | 6                              |

**Power Failure  
Line (PFND)**

◆ The PFND line is logically identical on all Direct Control Trunks (DCT) in the complex. Its signal is normally up but is dropped upon detection of a power failure. The signal on this line remains down throughout the one millisecond of available program time remaining, and does not come up again until after power has been restored.

**Power Failure Inhibit  
In Line (PFIR)**

◆ The PFIR line provides the means for inhibiting a Read Direct instruction of the associated Static In lines when its signal is dropped. When the signal is dropped, all zeros are read by the receiving processor.

**DUAL-PROCESSOR  
COMPLEX**

◆ The following illustration is presented to demonstrate the manner in which two processors are interconnected. In this instance only one cable is required.

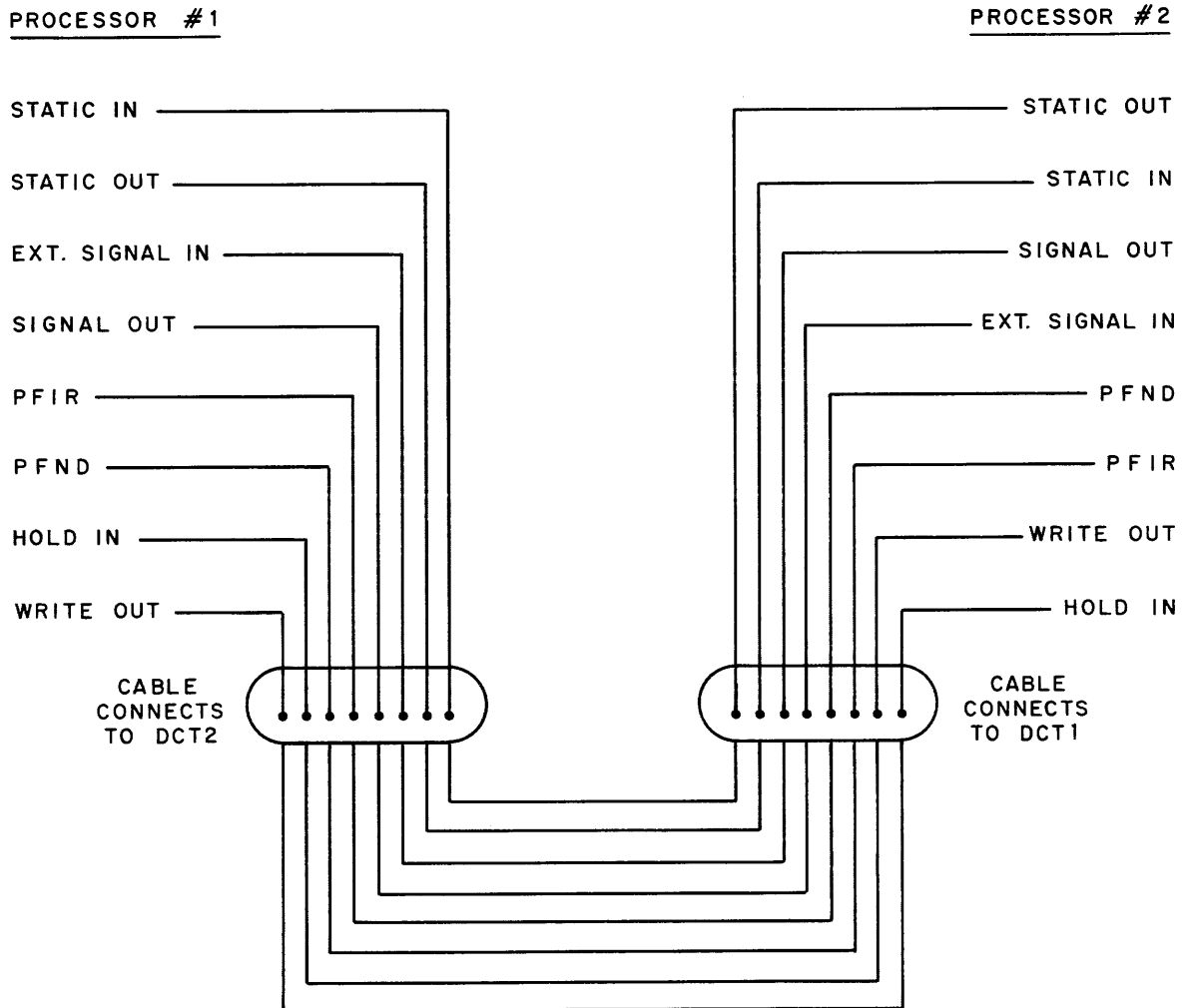


Figure 11. Dual-Processor Complex

**MASTER/SATELLITE  
COMPLEX**

◆ The Master/Satellite complex permits the master processor to communicate with its satellites and the satellites to communicate with the master processor. However, the satellites cannot communicate with each other. The following illustration demonstrates the manner in which the master processor interconnects with up to five satellite processors via the Direct Control Trunks (DCT).

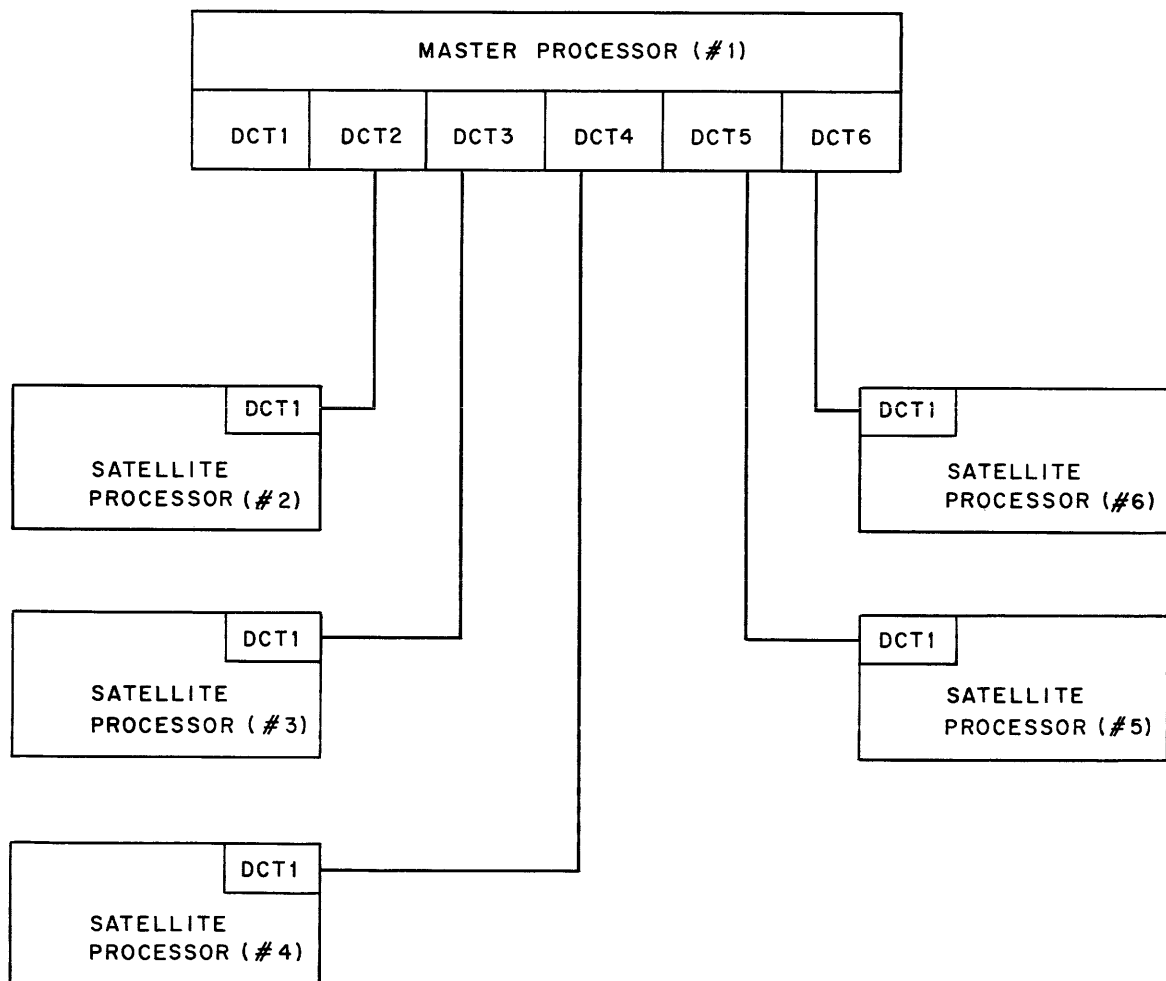


Figure 12. Master/Satellite Complex

# **MAXIMUM MULTI-PROCESSOR COMPLEX**

◆ The following illustration demonstrates the manner in which six processors may be interconnected so that any two processors may communicate.

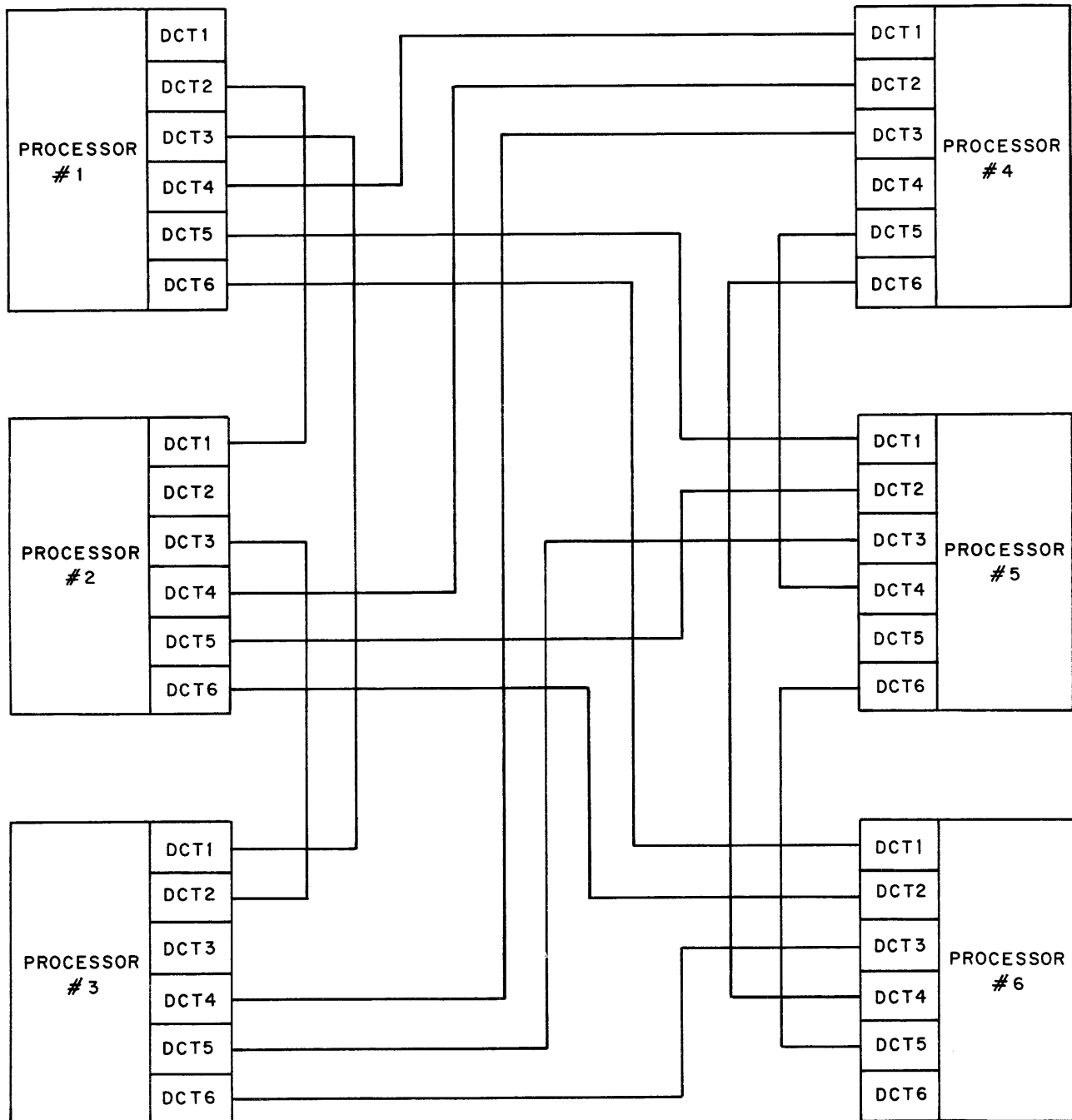


Figure 13. Maximum Multi-Processor Complex

## OPERATIONAL PROCEDURES

### Transmission Procedure

◆ The following sections are furnished to illustrate typical operational procedures when using the Direct Control feature. They are presented for *clarification only* and are not meant to imply fixed and firm standards. For a detailed description of the actual programming procedures, reference should be made to the applicable reference manuals.

◆ *User Program — ( $P_1$ )* The user program in Processing State ( $P_1$ ) contains a Supervisor Call instruction with a Write Direct Interrupt Code. In addition, it contains the following parameters required when interrupt is effected to the operating system in processor state ( $P_3$ ):

Data Byte (8-bit code)

Signal Byte (specifies processor(s) to which  
Write Direct is addressed)

Return Address (for return to normal processing)

*Operating System — ( $P_3$ )* The operating system accepts the Supervisor Call Interrupt and issues a Program Control instruction to ( $P_2$ ). In addition, the location of the user parameters are saved, the processor is set to the Privileged Mode and a change made from ( $P_3$ ) to ( $P_2$ ).

*Supervisor Call Routine — ( $P_2$ )* The Interrupt Weight is used to branch to the Supervisor Call routine where the Supervisor Call Interrupt Code is decoded and a branch is made to the required routine, in this case the Write Direct routine. The Write Direct routine then performs the following:

1. Checks to determine whether Write Direct instruction can be issued or must be stacked in queue.
2. Fetches the user parameters.
3. Sets Write Direct instruction I-Field to the Signal byte, the Address field to the Data byte, and the Return After Interrupt to the user Return Address in ( $P_1$ ).
4. Executes Write Direct instruction.
5. If no acknowledgement is received, sets control in Acknowledge queue.
6. Sets processor to non-privileged mode.
7. After interrupt, executes Program Control instruction and branch to user return address in ( $P_1$ ).

### Response Procedure

◆ *Operating System — ( $P_3$ )* The operating system accepts the Direct Control Interrupt and issues a Program Control instruction to ( $P_2$ ). In addition, the processor is set to privileged mode and a change made from ( $P_3$ ) to ( $P_2$ ).

**Response Procedure**  
(Cont'd)

*Read Direct Routine — ( $P_2$ )* The Interrupt Weight is used to branch to the Read Direct routine. The Read Direct routine then performs the following:

1. Issues a Read Direct instruction to read the Data Byte.
2. Saves the Data Byte and the External Interrupt number (which corresponds to the transmitting processor) for user Read Direct processing.
3. Issues a Program Control instruction to ( $P_1$ ) and sets processor to non-privileged mode.
4. Changes from ( $P_2$ ) to ( $P_1$ ) and branches to user Read Direct routine.

*User Read Direct Routine — ( $P_1$ )* Using the External Interrupt number, the user Read Direct routine determines the transmitting processor number and decodes the Data Byte to determine the type of action required.

If the Power Failure code (all zeros) is received, the processor that is down is removed from the system configuration and a return to normal processing is effected.

For all other codes received, a Write Direct acknowledgement is issued as follows:

1. Supervisor Call is issued with a Write Direct Interrupt Code.
2. A Write Direct instruction with a Data Byte of an Acknowledge Code and a return address of the user Read Direct routine is executed.

When the return is accomplished, the function specified by the Data Byte initially read is performed, and at the end of the Read Direct processing a branch is made back to the ( $P_1$ ) program.



## PRIVILEGED INSTRUCTIONS

### INTRODUCTION

- ◆ The instructions described in this section are called *privileged instructions* and can only be executed if the non-privileged mode bit (bit position 15 in the Interrupt Status register) for the current state is zero.

In addition to the standard privileged instruction set, inclusion of the memory protect and/or the direct control optional features cause additional privileged instructions to be added.

### INSTRUCTION FORMATS

#### RR Format

|         |                |                |
|---------|----------------|----------------|
| Op Code | R <sub>1</sub> | R <sub>2</sub> |
| 0 7     | 8 11           | 12 15          |

#### Description

- ◆ The RR format is used only by the Set Storage Key and the Insert Storage Key instructions. The contents of the general register specified by the R<sub>1</sub> field is the first operand. The general register specified by the R<sub>2</sub> field contains the second operand address.

#### SI Format

|         |                |                |                |
|---------|----------------|----------------|----------------|
| Op Code | I <sub>2</sub> | B <sub>1</sub> | D <sub>1</sub> |
| 0 7     | 8 15           | 16 19          | 20 31          |

#### Description

- ◆ The SI format is used by the Program Control, the Write Direct, the Read Direct instructions and all input/output instructions. The first address (B<sub>1</sub>/D<sub>1</sub>) specifies the main memory location of the first operand. The second operand is the immediate byte in the I<sub>2</sub> field.

#### SS Format

|         |      |                |                |                |                |
|---------|------|----------------|----------------|----------------|----------------|
| Op Code | L    | B <sub>1</sub> | D <sub>1</sub> | B <sub>2</sub> | D <sub>2</sub> |
| 0 7     | 8 15 | 16 19          | 20 31          | 32 35          | 36 47          |

#### Description

- ◆ The SS format is used by the Load Scratch Pad and the Store Scratch Pad instructions. The location of the first operand is specified by the first address (B<sub>1</sub>/D<sub>1</sub>), and the location of the second operand is specified by the second address (B<sub>2</sub>/D<sub>2</sub>). The L field is the number of *words* in addition to the addressed *word* that are to be transferred.

### INTERRUPT ACTION

#### Address Error

#### Addressing

- ◆ The following interrupt conditions can occur as a result of a privileged instruction:

- ◆ An address error interrupt occurs when an address specifies a location outside the available main memory of the particular installation. The operation is terminated at the point of error. The result data and condition code, if produced, are unpredictable. If the address of an instruction is invalid, the operation is suppressed.

|                                   |   |
|-----------------------------------|---|
| <i>Specification</i>              | <p>◆ An address error interrupt occurs when:</p> <ol style="list-style-type: none"><li>1. A Load Scratch Pad or Store Scratch Pad instruction specifies a first or second address which is not on a word boundary.</li><li>2. Bits 28 through 31 of the second operand of a Set Storage Key or Insert Storage Key instruction are not zero.</li><li>3. The memory protect feature is not installed and the protection key in the Interrupt Status register for the current program state is not zero.</li><li>4. The Program Control instruction specifies an instruction address which is not on a halfword boundary.</li></ol> <p>In these error interrupt conditions, the operation is suppressed. The data in main memory and registers is unchanged.</p> |
| <i>Protection</i>                 | <p>◆ An address error interrupt occurs when the storage key and the protection key of the result location do not match. The operation is terminated. The result data is unpredictable. (This interrupt can occur only if the memory protect feature is installed.)</p>  |
| <b>Privileged Operation</b>       | <p>◆ A privileged operation interrupt occurs if execution of any privileged instruction is attempted and the non-privileged mode bit (bit position 15 in the Interrupt Status register) for the current state is 1. The operation is suppressed and the condition code, registers, and main memory are unaltered.</p>   |
| <b>Operation Code Trap</b>        | <p>◆ An operation code trap interrupt occurs under the following conditions:</p> <ol style="list-style-type: none"><li>1. The memory protect feature is not installed and an attempt to execute a Set Storage Key or Insert Storage Key instruction is made.</li><li>2. The direct control feature is not installed and an attempt to execute a Write Direct or Read Direct instruction is made.</li></ol>  |
| <b>SPECIAL<br/>CONSIDERATIONS</b> | <p>◆ The following sections outline programming rules which must be followed in order to maintain compatibility for programs which are to run on all three processors (70/35, 70/45, and 70/55).</p>  |
| <b>Program Mask</b>               | <p>◆ The contents of the P-Counter in scratch-pad memory associated with the current program state does not necessarily contain the active Program Mask. Programs should <i>not</i> address the P-Counter of the current program state via Load or Store Scratch Pad instructions and expect to effect or obtain the active Program Mask. While the 70/45 and 70/55 processors utilize a hardware register to contain the Program Mask (which is updated whenever a change of program state is made or a Set Program Mask instruction is executed), the 70/35 processor utilizes the P-Counter as the active Program Mask at all times.</p>   |
| <b>Scratch-Pad Addresses</b>      | <p>◆ The first address of the Load and of the Store Scratch Pad instructions specifies word locations 0–127 by the seven rightmost bits. Bits to the left of these must be zeros since the 70/35 processor incorporates the scratch-pad area as the lower 128-word portion of non-addressable memory. Addresses greater than 127 will specify portions of 70/35 non-addressable memory not containing scratch pad. This does not occur on the 70/45</p>   |

|   |  |
|---|--|
| <b>Scratch-Pad Addresses<br/>(Cont'd)</b> | and 70/55 processors as scratch pad is implemented separately and wrap-around occurs module 128. (i.e., if location 127 is loaded, location zero may be loaded next.)  |
| <b>Next Instruction Address</b>           | ◆ The contents of the P-Counter in scratch-pad memory associated with the current program state does not necessarily contain the Next Instruction Address (NIA). Programs should <i>not</i> address the P-Counter of the current program state via the Load or the Store Scratch Pad instructions and expect to effect a branch or obtain the address of the next instruction in sequence. While the 70/45 and 70/55 processors utilize the P-Counter as the NIA Register at all times, the 70/35 utilizes a hardware NIA Register. (The contents of the NIA-Field in the P-Counter are updated whenever a change of program state is to be effected or if the hardware NIA Register is to be used as a temporary utility register.) |

## **Load Scratch Pad (LSP)**

### **General Description**

- ◆ Operands from main memory, starting with the storage location specified by the second address ( $B_2/D_2$ ), are loaded in the scratch-pad memory starting at the location specified by the first address ( $B_1/D_1$ ).

### **Format (SS)**

|    |     |                |                |                |                |
|----|-----|----------------|----------------|----------------|----------------|
| D8 | L   | B <sub>1</sub> | D <sub>1</sub> | B <sub>2</sub> | D <sub>2</sub> |
| 0  | 7 8 | 15 16 19 20    | 31 32 35 36    | 47             |                |

### **Condition Code**

- ◆ Unchanged except when the P counter in scratch-pad memory is loaded.

### **Interrupt Action**

- ◆ Privileged operation.

Address error:

Addressing.

Specification.

### **Notes**

- ◆ 1. The L field provides an eight-bit count specifying the number of scratch-pad memory locations to be loaded. An initial count of zero specifies one word to be loaded.
- 2. The first address specifies scratch-pad memory words 0 through 127 by the seven rightmost bits of the address. The bits to the left of the seven-bit address must be zero.
- 3. The second address must be on a word boundary. (This is a program restriction.)
- 4. The processor uses utility registers in scratch-pad memory to execute this instruction. If these registers are included in the range of this instruction results are unpredictable.

## Store Scratch Pad (SSP)

### General Description

◆ Operands from the scratch-pad memory, starting with the location specified by the first address ( $B_1/D_1$ ), are stored in main memory locations, starting with the location specified by the second address ( $B_2/D_2$ ).

### Format (SS)

|    |     |                |                |                |                |
|----|-----|----------------|----------------|----------------|----------------|
| DO | L   | B <sub>1</sub> | D <sub>1</sub> | B <sub>2</sub> | D <sub>2</sub> |
| 0  | 7 8 | 15 16 19       | 20             | 31 32 35 36    | 47             |

### Condition Code

◆ Unchanged.

### Interrupt Action

◆ Privileged operation.

Address error:

Addressing.

Specification.

Protection.

### Notes

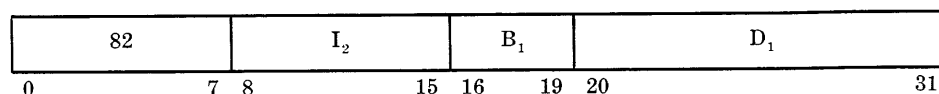
- ◆ 1. The L field provides an eight-bit count specifying the number of scratch-pad memory locations to be stored. An initial count of zero specifies one word to be stored.
- 2. The first address specifies scratch-pad memory words 0 through 127 by the seven rightmost bits of the address. The bits to the left of the seven-bit address must be zero.
- 3. The second address must be on a word boundary. (This is a program restriction.)

## Program Control (PC)

### General Description

◆ This instruction specifies the termination of program execution in the current state, and the initiation of another state under control of the immediate byte in the  $I_2$  field. The address computed from the  $B_1/D_1$  address components of the instruction is stored in the P counter of the state being terminated (bit positions 8–31).

### Format (SI)



### Condition Code

◆ The condition code indicators of the state being terminated are preserved in the state's P counter. The condition code in the P counter of the initiated state is then used to set the condition code indicators.

### Interrupt Action

◆ Privileged operation.

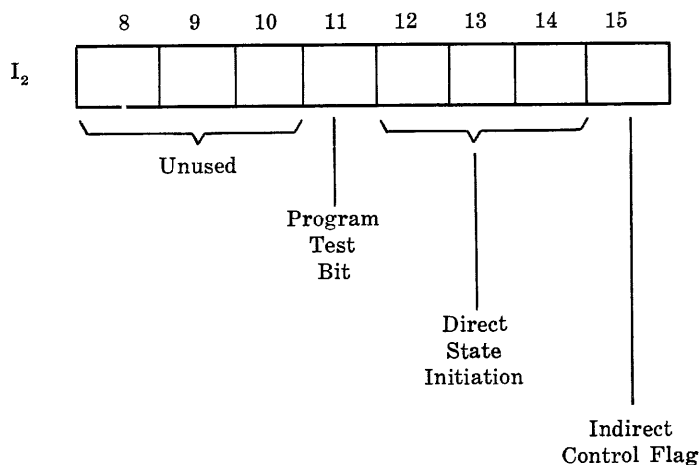
Address error:

Addressing.

Specification.

### Note

◆ 1. The immediate byte in the  $I_2$  field of the instruction is divided into four subfields as follows:



*Bits 8 through 10 are unused. The three bit unused portion must be zero.*

*Bit 11 is the program test bit. If bit 11 = 1, the program test mode is initiated. The program test interrupt bit is set in the Interrupt Flag register of the initiated state.*

The scan of the Interrupt Flag register in the initiated state is delayed until after the first instruction of the initiated state is executed, at which time the scan is made in normal priority.

If bit 11 = 0, the program test mode is not initiated.

**Note  
(Cont'd)**

*Bits 12 through 14* are the direct state initiation bits. The three-bit direct state initiation codes that may be specified are as follows:

- 000 — Go to Machine Condition State  $P_4$ .
- 001 — Go to Interrupt Control State  $P_3$ .
- 010 — Go to Interrupt Response State  $P_2$ .
- 011 — Go to Processing State  $P_1$ .

*Programming Note:* The leftmost bit of the three-bit direct state initiation field must be zero. (This is a programming restriction.)

*Bit 15* is the indirect control flag bit. If indirect state control is specified (bit 15 = 1), the three-bit direct state initiation field is ignored. The three-bit interrupted state identifier (ISI), which indicates the last state interrupted, specifies the state to be initiated. This information is contained in the Interrupt Status register of the state being terminated.

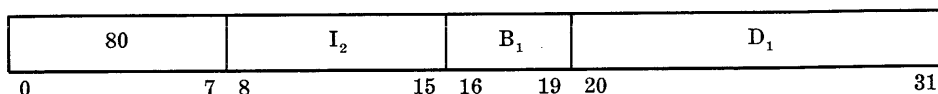
If bit 15 = 0, direct state initiation is used.

**Idle  
(IDL)**

**General Description**

- ◆ This instruction effects an idle mode within the processor by continuously branching back to itself.

**Format  
(SI)**



**Condition Code**

- ◆ Unchanged.

**Interrupt Action**

- ◆ Privileged operation.

**Notes**

- ◆ 1. When this instruction is operating with the I field zero, the Idle light of the console is on.
- 2. Any interrupt occurring while the idle mode is in effect is taken (if permitted via the Interrupt Mask register).
- 3. The B<sub>1</sub> and D<sub>1</sub> fields of this instruction must be zero.
- 4. For normal programming, the I field must be zero. For maintenance programming, bits within the I field, have the following meaning:
  - Bit 15 = 1-set alarm inhibit.
  - Bit 14 = 1-reset alarm inhibit.
  - Bit 13 = 1-set inhibit simultaneity.
  - Bit 12 = 1-reset-inhibit simultaneity.



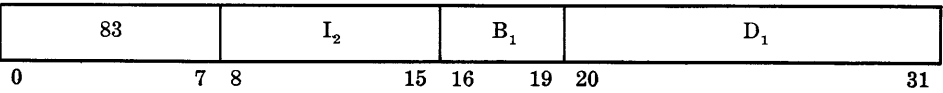
**Diagnose  
(DIG)**

**General Description**

◆ The purpose of this privileged instruction is to provide a means for facilitating maintenance techniques on the 70/35, 70/45 and 70/55 Processors. It is provided for the RCA Customer Service and Engineering Representatives and cannot be used for a program debugging aid.

The mechanics of this instruction are implemented differently for each of the three processors. The Diagnose instruction designed for the 70/35 Processor is unique to that processor, the one designed for the 70/45 is unique to that processor, and the one designed for the 70/55 is unique to that processor.

**Format  
(SI)**



## Start Device (SDV)

### General Description

- ◆ The contents of the general register specified by  $B_1$  are added to the  $D_1$  field. The resultant sum identifies the channel and device to which the instruction applies. These are specified by bit positions 21 through 31 of the sum. The I-field is not used and must be zeros.

The channel address word in main memory location 72 contains the protection key to be used and the address of the first channel command word. The channel command word designated by the channel address word specifies the operation to be performed, the main memory area to be used, and the action to be taken when the operation is completed. The condition code indicates the result of the instruction.

### Format (SI)

|    |       |             |       |
|----|-------|-------------|-------|
| 9C | $I_2$ | $B_1$       | $D_1$ |
| 0  | 7 8   | 15 16 19 20 | 31    |

### Condition Code

- ◆ 0 — input/output operation initiated and channel proceeding with execution.
  - 1 — status bits stored in scratch-pad memory.
  - 2 — busy or interrupt pending.
  - 3 — inoperable.
- (For a detailed description of the condition code settings, see Notes below.)

### Interrupt Action

- ◆ Privileged operation.

### Notes

- ◆ 1. The address portion of this instruction specifies the device and channel as follows:

| Bit Positions |    |    | Channel Specified |
|---------------|----|----|-------------------|
| 21            | 22 | 23 |                   |
| 0             | 0  | 0  | Multiplexor       |
| 0             | 0  | 1  | Selector No. 1    |
| 0             | 1  | 0  | Selector No. 2    |
| 0             | 1  | 1  | Selector No. 3    |
| 1             | 0  | 0  | Selector No. 4    |
| 1             | 0  | 1  | Selector No. 5    |
| 1             | 1  | 0  | Undesignated      |
| 1             | 1  | 1  | Selector No. 6    |

Bit positions 24 through 31 specify one of 256 possible devices.

- The standard device byte and the channel status byte stored by the previous input/output instruction in scratch-pad memory is destroyed if the condition code at the completion of the Start Device instruction is 0 or 1.
- Status storage (channel status byte and standard device byte), if required, occurs before the Start Device instruction terminates.
- Condition Code 0 is set under the following conditions:

**Notes  
(Cont'd)**

- a. The device control electronics and the device specified are available.
  - b. The Start Device instruction specifies a Sense command to a device that is inoperable.
5. Condition Code 1 indicates that either the channel status byte or the standard device byte has been stored in the channel registers in scratch-pad memory for the specified channel.

The channel status byte is stored under the following conditions:

- a. A parity error occurs while accessing the Channel Address Word (CAW) or a Channel Command Word (CCW). The channel control check bit in the channel status byte is set.
- b. The Memory Protect feature is not installed and the key in the CAW is not zero. The program check bit in the channel status byte is set.
- c. The main memory address specified in the CAW is not on a double word boundary. The program check bit in the channel status byte is set.
- d. The main memory address in the CCW specifies an address outside the available memory for the system. The program check bit in the channel status byte is set.

The standard device byte is stored under the following conditions:

- a. The specified device control electronics on the multiplexor channel indicates that a device request interrupt pending condition is present. The external device request interrupt pending bit in the standard device byte is set.
  - b. The Start Device instruction specifies a command which is other than a Sense command and the addressed device is inoperable. The device inoperable bit in the standard device byte is set.
  - c. The specified device is busy but the device control electronics is not busy (i.e., tape rewinding, off-line seek to a random access device end bit in the standard device byte are set.
6. Condition Code 2 is set under the following conditions:
- a. A selector channel is specified that is busy.
  - b. A selector channel is specified that has an interrupt pending (termination or external device request).
  - c. The multiplexor channel is specified and it is operating in burst mode.
  - d. The multiplexor channel is specified and the addressed device control electronics is busy with addressed or non-addressed device.
  - e. The multiplexor channel is specified and the addressed device control electronics has a termination interrupt pending.
  - f. A burst mode operation is directed to the multiplexor and there is a termination interrupt pending on one of the attached device control electronics.
7. Condition Code 3 is set under the following conditions:
- a. A selector channel is specified that is not in the system.
  - b. The specified device control electronics is inoperable.

**Notes  
(Cont'd)**

8. If the condition code is 1, 2 or 3 the input/output operation is not initiated.
9. Parity errors that occur while fetching the CAW or CCW or that occur after the input/output operation has been initiated do not cause a machine check interrupt. A channel interrupt occurs and the program is notified of the error via the channel status byte.
10. If the first CCW is a Transfer in Channel command the Start Device instruction terminates and the condition code is set to 0. However, the specified device control electronics recognizes this command as an illegal operation and causes a channel interrupt to occur.

# Halt Device (HDV)

## General Description

◆ The contents of the general register specified by  $B_1$  are added to the  $D_1$  field, and the resultant sum identifies the channel to be halted. The channel is specified by bit positions 21 through 23 of the sum. If a multiplexor is specified, bit positions 24 through 31 of the sum identify the device to be halted. The I field is not used and must be zeros. Buffered devices operating off-line, and independent of the channel/device control electronics, cannot be stopped by using this instruction. The condition code specifies the results of the instruction.

## Format (SI)

|    |       |       |       |
|----|-------|-------|-------|
| 9E | $I_2$ | $B_1$ | $D_1$ |
| 0  | 7 8   | 15 16 | 19 20 |
|    |       |       | 31    |

## Condition Code

- ◆ 0 — not busy.
- 1 — standard device byte stored in scratch-pad memory.
- 2 — termination accepted.
- 3 — inoperable.

(For a detailed description of the condition code settings, see Notes below.)

## Interrupt Action

- ◆ Privileged operation.

## Notes

- ◆ 1. The address portion of this instruction specifies the device and channel as follows:

| Bit Positions |    |    | Channel Specified |
|---------------|----|----|-------------------|
| 21            | 22 | 23 |                   |
| 0             | 0  | 0  | Multiplexor       |
| 0             | 0  | 1  | Selector No. 1    |
| 0             | 1  | 0  | Selector No. 2    |
| 0             | 1  | 1  | Selector No. 3    |
| 1             | 0  | 0  | Selector No. 4    |
| 1             | 0  | 1  | Selector No. 5    |
| 1             | 1  | 0  | Undesignated      |
| 1             | 1  | 1  | Selector No. 6    |

Bit positions 24 through 31 specify one of 256 possible devices.

2. If a device operating on a selector channel is to be halted, the device number does *not* have to be specified.
3. The channel address word in main memory location 72 and the channel command word are *not* used by this instruction.
4. A termination interrupt occurs when any input/output operation is terminated. Status bits are stored in scratch-pad memory when the termination interrupt occurs.
5. All five flags in CCR-II are cleared if the Halt Device instruction is accepted. Therefore, upon termination, the incorrect length counter in the channel status byte is set if the count is not zero.

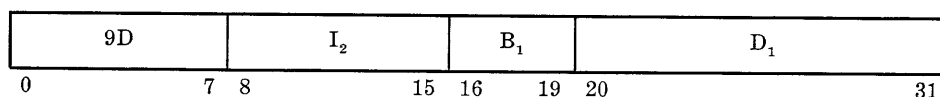
**Notes  
(Cont'd)**

6. A Halt Device instruction that specifies a multiplexor channel that is operating in the burst mode must specify a device that is operating in the burst mode.
7. Condition Code 0 is set under the following conditions:
  - a. The device control electronics or the device specified on the multiplexor channel is not busy. No termination is required.
  - b. A selector channel or the multiplexor channel operating in burst mode is specified and it is not busy. No termination is required.
  - c. The multiplexor channel is specified and the addressed device control electronics has a termination interrupt pending. No termination is required.
8. Condition Code 1 indicates that the specified device is on the multiplexor channel and that the standard device byte has been stored in the channel registers in scratch-pad memory for the multiplexor channel. The channel status byte is never stored.

The standard device byte is stored under the following conditions:

  - a. The specified device indicates that a device request interrupt pending condition is present. The external device request interrupt pending bit in the standard device byte is set.
  - b. The specified device is busy but the device control electronics is not busy (i.e., tape rewinding). The device busy bit in the standard device byte is set.
  - c. The specified device is inoperable. The device inoperable bit in the standard device byte is set.
9. Condition Code 2 is set under the following conditions:
  - a. A selector channel is specified that is busy.
  - b. The multiplexor channel is specified and it is operating in the burst mode.
  - c. The multiplexor channel is specified and the addressed device control electronics and device are busy.
10. Condition Code 3 is set under the following conditions:
  - a. A selector channel is specified that it is not in the system.
  - b. The specified device control electronics is inoperable.
11. Status storage (standard device byte), if required, occurs before the Halt Device instruction terminates.

◆ The contents of the general register specified by B<sub>i</sub> are added to the D<sub>i</sub> field. The resultant sum identifies the channel and device to which the instruction applies. These are specified by bit positions 21 through 31 of the sum. The I-field is not used and must be zeros. The condition code specifies the results of the instruction.



- ◆ 0 — available.
- 1 — standard device byte stored in scratch-pad memory.
- 2 — busy or interrupt pending.
- 3 — inoperable.

- ◆ Privileged operation.

- ◆ 1. The address portion of this instruction specifies the device and channel as follows:

| Bit Positions |    |    | Channel Specified |
|---------------|----|----|-------------------|
| 21            | 22 | 23 |                   |
| 0             | 0  | 0  | Multiplexor       |
| 0             | 0  | 1  | Selector No. 1    |
| 0             | 1  | 0  | Selector No. 2    |
| 0             | 1  | 1  | Selector No. 3    |
| 1             | 0  | 0  | Selector No. 4    |
| 1             | 0  | 1  | Selector No. 5    |
| 1             | 1  | 0  | Undesignated      |
| 1             | 1  | 1  | Selector No. 6    |

Bit positions 24 through 31 specify one of 256 possible devices.

2. The channel address word in main memory location 72 and the channel command word are not used by this instruction.
3. Status storage (standard device byte), if required, occurs before the Test Device instruction terminates.
4. Condition Code 0 is set if the device control electronics and the device are available.

*Note:* There may be pending interrupts on the multiplexor channel that would prohibit a burst mode operation to be initiated.

5. Condition Code 1 indicates that the standard device byte has been stored in the channel registers in scratch-pad memory for the specified channel. The channel status byte is never stored by this instruction.

The standard device byte is stored under the following conditions:

- a. The specified device control electronics on the multiplexor channel indicates that a device request interrupt pending condition

**Notes  
(Cont'd)**

- is present. The external device request interrupt pending bit in the standard device byte is set.
- b. The specified device is busy but the device control electronics is not busy (i.e., tape rewinding, off-line seek to a random access device). The device busy bit in the standard device byte is set.
  - c. The specified device is inoperable. The device inoperable bit in the standard device byte is set.
6. Condition Code 2 is set under the following conditions:
- a. A selector channel is specified that is busy.
  - b. A selector channel is specified that has an interrupt pending (termination or external device request.)
  - c. The multiplexor channel is specified and it is operating in burst mode.
  - d. The multiplexor channel is specified and the addressed device control electronics is busy with addressed or non-addressed device.
  - e. The multiplexor channel is specified and the addressed device control electronics has a termination interrupt pending.
7. Condition Code 3 is set under the following conditions:
- a. A selector channel is specified which is not in the system.
  - b. The specified device control electronics is inoperable.
  - c. A device is specified that is not in the system.



## Check Channel (CKC)

### General Description

◆ The contents of the general register specified by  $B_1$  are added to the  $D_1$  field, and the resultant sum identifies the input/output channel to be tested. This is specified by bit positions 21 through 23 of the sum. Only the channel is tested.

### Format (SI)

|    |                |                |                |
|----|----------------|----------------|----------------|
| 9F | I <sub>2</sub> | B <sub>1</sub> | D <sub>1</sub> |
| 0  | 7 8            | 15 16          | 19 20          |
|    |                |                | 31             |

### Condition Code

- ◆ 0 — a. The specified selector channel is not busy and has no interrupts pending.  
           b. The specified multiplexor channel is not operating in the burst mode.
- 1 — The specified selector channel has an external device request interrupt pending.
- 2 — a. The specified selector channel is busy or has a terminating interrupt pending.  
           b. The specified multiplexor is operating in the burst mode.
- 3 — A selector channel is specified that is not in the system.

### Interrupt Action

- ◆ Privileged operation.

### Notes

- ◆ 1. The address portion of this instruction specifies the channel to be tested as follows:

| Bit Positions |    |    | Channel Specified |
|---------------|----|----|-------------------|
| 21            | 22 | 23 |                   |
| 0             | 0  | 0  | Multiplexor       |
| 0             | 0  | 1  | Selector No. 1.   |
| 0             | 1  | 0  | Selector No. 2.   |
| 0             | 1  | 1  | Selector No. 3    |
| 1             | 0  | 0  | Selector No. 4    |
| 1             | 0  | 1  | Selector No. 5    |
| 1             | 1  | 0  | Undesignated      |
| 1             | 1  | 1  | Selector No. 6    |

2. The channel address word in main memory location 72 and the channel command word are not used by this instruction.
3. The device address (bit positions 24 through 31 of the sum) is not used by this instruction.
4. Status bits (channel status byte and standard device byte) are not stored in scratch-pad memory by this instruction.
5. Current operations proceeding in the specified channel are unaffected by this instruction.

## Insert Storage Key (ISK)

### General Description

- ◆ The storage key of the 2,048-byte main memory block, which is located at the address contained in the general register specified by the second address ( $R_2$ ), is inserted in the general register specified by the first address ( $R_1$ ).

### Format (RR)

|    |       |          |
|----|-------|----------|
| 09 | $R_1$ | $R_2$    |
| 0  | 7 8   | 11 12 15 |

### Condition Code

- ◆ Unchanged.

### Interrupt Action

- ◆ Privileged operation.  
Address error:  
Addressing.  
Specification.  
Operation code trap (if the memory protect feature is not installed).

### Notes

- ◆ 1. The general register specified by the second address ( $R_2$ ) contains the location of the 2,048-byte main memory block in bits 8 through 20. Bits 0 through 7 and 21 through 27 are ignored. Bits 28 through 31 must be zero.
- 2. When the four-bit storage key is inserted into bits 24 through 27 of the general register specified by the first address, bits 0 through 23 are unaltered and bits 28 through 31 are made zero.
- 3. The address of the storage key for a specific 2,048-byte main memory block is specified in  $R_2$  by a binary count as shown in the following examples:

Storage Key Address in  $R_2$

|   |  |  |  |  |  |  |   |   |  |  |  |  |  |  |   |   |   |   |   |                  |    |    |   |   |   |   |   |         |    |    |   |   |    |   |
|---|--|--|--|--|--|--|---|---|--|--|--|--|--|--|---|---|---|---|---|------------------|----|----|---|---|---|---|---|---------|----|----|---|---|----|---|
| IGNORED   |  |  |  |  |  |  |   |   |  |  |  |  |  |  | 0 | 0 | 0 | 0 | 0 | 0                | 0  | 0  | 0 | 0 | 0 | 0 | 0 | IGNORED |    |    | 0 | 0 | 0  | 0 |
| 0   |  |  |  |  |  |  | 7 | 8 |  |  |  |  |  |  |   |   |   |   |   |                  | 20 | 21 |   |   |   |   |   |         | 27 | 28 |   |   | 31 |   |
| Address of Storage<br>key for <i>first</i> 2,048<br>main memory block |  |  |  |  |  |  |   |   |  |  |  |  |  |  |   |   |   |   |   | Must be<br>zeros |    |    |   |   |   |   |   |         |    |    |   |   |    |   |

|   |  |  |  |  |  |  |   |   |  |  |  |  |  |  |   |   |   |   |   |                  |    |    |   |   |   |   |   |         |    |    |   |   |    |   |
|---|--|--|--|--|--|--|---|---|--|--|--|--|--|--|---|---|---|---|---|------------------|----|----|---|---|---|---|---|---------|----|----|---|---|----|---|
| IGNORED   |  |  |  |  |  |  |   |   |  |  |  |  |  |  | 0 | 0 | 0 | 0 | 0 | 0                | 0  | 0  | 0 | 0 | 0 | 1 | 0 | IGNORED |    |    | 0 | 0 | 0  | 0 |
| 0   |  |  |  |  |  |  | 7 | 8 |  |  |  |  |  |  |   |   |   |   |   |                  | 20 | 21 |   |   |   |   |   |         | 27 | 28 |   |   | 31 |   |
| Address of Storage<br>key for <i>third</i> 2,048<br>main memory block |  |  |  |  |  |  |   |   |  |  |  |  |  |  |   |   |   |   |   | Must be<br>zeros |    |    |   |   |   |   |   |         |    |    |   |   |    |   |

|   |  |  |  |  |  |  |   |   |  |  |  |  |  |  |   |   |   |   |   |                  |    |    |   |   |   |   |   |         |    |    |   |   |    |   |
|---|--|--|--|--|--|--|---|---|--|--|--|--|--|--|---|---|---|---|---|------------------|----|----|---|---|---|---|---|---------|----|----|---|---|----|---|
| IGNORED   |  |  |  |  |  |  |   |   |  |  |  |  |  |  | 0 | 0 | 0 | 0 | 0 | 0                | 0  | 0  | 0 | 1 | 0 | 0 | 1 | IGNORED |    |    | 0 | 0 | 0  | 0 |
| 0   |  |  |  |  |  |  | 7 | 8 |  |  |  |  |  |  |   |   |   |   |   |                  | 20 | 21 |   |   |   |   |   |         | 27 | 28 |   |   | 31 |   |
| Address of Storage<br>key for <i>tenth</i> 2,048<br>main memory block |  |  |  |  |  |  |   |   |  |  |  |  |  |  |   |   |   |   |   | Must be<br>zeros |    |    |   |   |   |   |   |         |    |    |   |   |    |   |

## Set Storage Key (SSK)

### General Description

◆ The storage key of a 2,048-byte main memory block located at the address contained in the general register specified by the second address ( $R_2$ ) is set according to the value contained in the register specified by the first address ( $R_1$ ).

### Format (RR)

|                              |         |       |
|------------------------------|---------|-------|
| 08                           | $R_1$   | $R_2$ |
| 0                      7   8 | 11   12 | 15    |

### Condition Code

◆ Unchanged.

### Interrupt Action

◆ Privileged operation.

Address error:

Addressing.

Specification.

Operation code trap (if the memory protect feature is not installed).

### Notes

- ◆ 1. Bits 8 through 20 of the register specified by the second address ( $R_2$ ) contain the location of the storage key for a 2,048-byte main memory block. Bits 0 through 7 and 21 through 27 are ignored. Bits 28 through 31 must be zero.
- 2. Bits 24 through 27 of the general register specified by the first address ( $R_1$ ) contain the four-bit storage key to be assigned. Bits 0 through 23 and 28 through 31 are ignored.
- 3. The address of the storage key for a specific 2,048-byte main memory block is specified in  $R_2$  by a binary count (see examples under Insert Storage Key description).

## Write Direct (WRD)

### General Description

◆ The eight-bit byte specified by the first address ( $B_1/D_1$ ) is accessed and transmitted to all units via the Static Out lines. The eight-bit I field specifies the Signal Out lines to be pulsed. The Static Out lines remain as specified until the next Write Direct instruction.

### Format (SI)

|    |   |       |    |       |    |       |    |
|----|---|-------|----|-------|----|-------|----|
| 84 |   | $I_2$ |    | $B_1$ |    | $D_1$ |    |
| 0  | 7 | 8     | 15 | 16    | 19 | 20    | 31 |

### Condition Code

◆ Unchanged.

### Interrupt Action

◆ Privileged operation.

Address error:

Addressing.

Operation code trap (if Direct Control option is not installed).

### Notes

- Each trunk has only one Signal Out line and is pulsed according to the following pattern:

| <u>I-Field</u> | <u>Trunk(s) Pulsed</u>  |
|----------------|-------------------------|
| Bit 0 = 1      | Six                     |
| Bit 1 = 1      | Five                    |
| Bit 2 = 1      | Four                    |
| Bit 3 = 1      | Three                   |
| Bit 4 = 1      | Two                     |
| Bit 5 = 1      | One                     |
| Bit 6 = 0      | Reserved (Must be zero) |
| Bit 7 = 0      | Reserved (Must be zero) |

More than one I-Field bit may be set to 1 providing pulses for sending over more than one direct control trunk. This permits sending the same byte to all processors connected to the transmitting processor.

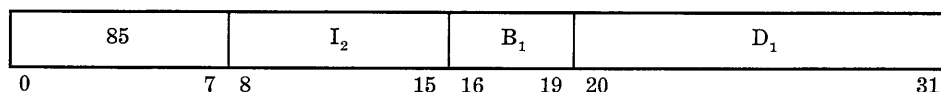
- A processor cannot Write Direct to itself. The I-Field bit associated with the transmitting processor must always be reset to zero. (This is a programming restriction.)

## Read Direct (RDD)

### General Description

- ◆ The eight-bit I field specifies one of up to five possible sets of Direct Control trunks to be sampled. The sampled eight-bit byte is transferred to the main memory location specified by the first address ( $B_1/D_1$ ) from the Static In lines.

### Format



### Condition Code

- ◆ Unchanged.

### Interrupt Action

- ◆ Privileged operation.  
     Address error:  
         Addressing.  
         Protection.  
     Operation code trap (if Direct Control option is not installed).

### Notes

- ◆ 1. Each of the six Direct Control trunks has a set of Direct In lines which are sampled according to the following pattern:

| <u>I-Field</u> | <u>Trunk Sampled</u>  |
|----------------|-----------------------|
| Bit 0 = 1      | Six                   |
| Bit 1 = 1      | Five                  |
| Bit 2 = 1      | Four                  |
| Bit 3 = 1      | Three                 |
| Bit 4 = 1      | Two                   |
| Bit 5 = 1      | One                   |
| Bit 6 = 0      | Unused (Must be zero) |
| Bit 7 = 0      | Unused (Must be zero) |

The program must specify only one I-Field bit set to 1, otherwise results of the instruction are unpredictable.

2. A processor cannot Read Direct to itself. The I-Field bit associated with the receiving processor must always be reset to zero. (This is a programming restriction.)
3. This instruction may be prolonged by the presence of a HOLD signal. If so, timer updating may be skipped. However, I/O servicing will not be affected.

## PROCESSOR STATE CONTROL INSTRUCTIONS

### INTRODUCTION

◆ There are two control instructions that can be used in the *Processing State* ( $P_1$ ). These instructions are Supervisor Call, and Set Program Mask. These instructions can also be executed in any other state.

The Supervisor Call instruction enables the program to switch from any state to the *Interrupt Control State* ( $P_3$ ). Through this operation a program in any processor state can communicate with and initiate the *Interrupt Control State* ( $P_3$ ) programs.

The Set Program Mask instruction permits the user to specify whether or not the program is to be interrupted for any of the following errors:

1. significance error.
2. exponent underflow.
3. decimal overflow.
4. fixed-point overflow.

The execution of the Set Program Mask instruction causes the condition code and program mask bits in the P counter of the state in which the system is operating to be set to the value specified by the instruction. This instruction always changes the condition code.

### INSTRUCTION FORMAT

#### RR Format

| Op Code |   |   |    | R <sub>1</sub> |    | R <sub>2</sub> |  |
|---------|---|---|----|----------------|----|----------------|--|
| 0       | 7 | 8 | 11 | 12             | 15 |                |  |

#### Description

◆ The RR format is used for the Supervisor Call and Set Program Mask instructions. For the Set Program Mask instruction, the R<sub>2</sub> field is ignored. The contents of the general register specified by the R<sub>1</sub> field form the first operand.

For the Supervisor Call instruction, the R<sub>1</sub> and R<sub>2</sub> fields are combined to become an immediate operand. This operand does not refer to any register, but is a value which is placed in the Interrupt Status Register (ISR) of the initiated state to provide communication with the software in this state.

### CONDITION CODE UTILIZATION

◆ The condition code is changed by the Set Program Mask instruction. The condition code and program mask bits of the current P counter are replaced by the contents of the general register (bits 2-7) specified by the first address of the instruction.

### INTERRUPT ACTION

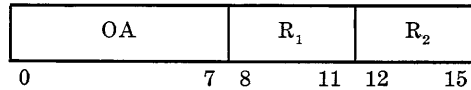
◆ No error interrupts can occur as a result of using the instructions in this section. The Supervisor Call instruction causes an interrupt, but this interrupt is the desired result of its execution.

## Supervisor Call (SVC)

### General Description

◆ The  $R_1$  and  $R_2$  fields provide an interruption code and this code is placed into the rightmost byte of the Interrupt Status Register (ISR) of the program state in which this instruction is issued. The supervisor call interrupt flag bit (priority 21) is set in the Interrupt Flag register and a program interrupt may occur depending on the associated mask bit in the Interrupt Mask register of the current state.

### Format (RR)



### Condition Code

◆ Unchanged.

### Interrupt Action

◆ None.

### Note

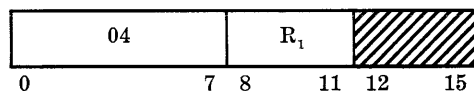
◆ If a higher priority interrupt is honored upon executing this instruction, the flag bit (priority 21) will be set and the Supervisor Call byte stored in the ISR so that when it is honored, the results are independent of any higher priority interrupts.

## Set Program Mask (SPM)

### General Description

◆ Bits 2-7 of the general register specified by the first address ( $R_1$ ) establish new program masks and condition code setting for the current program state.

### Format (RR)



### Condition Code

◆ The condition code is set according to bits 2 and 3 of the general register specified by  $R_1$  as follows:

Condition Code Setting

| 2 | 3 | Result                       |
|---|---|------------------------------|
| 0 | 0 | Set condition code 0 (zero). |
| 0 | 1 | Set condition code 1.        |
| 1 | 0 | Set condition code 2.        |
| 1 | 1 | Set condition code 3.        |

### Program Mask

◆ The program mask is set according to bits 4-7 of the general register specified by  $R_1$  as follows:

Program Mask Setting

| Bit | Result                |
|-----|-----------------------|
| 4   | Fixed-point overflow. |
| 5   | Decimal overflow.     |
| 6   | Exponent underflow.   |
| 7   | Significance error.   |

### Note

◆ The contents of the P-counter and the register specified by the first address are unaltered.



## FIXED-POINT INSTRUCTIONS

### INTRODUCTION

◆ Using fixed-point instructions, binary arithmetic is performed on operands used as addresses, index quantities, counts, and fixed-point data. Generally, the operands involved are 32 bits long and signed. One of the general registers always holds one operand. The other operand is in either main memory or in a general register. Negative quantities are in the two's-complement form.

This instruction set performs the following functions:

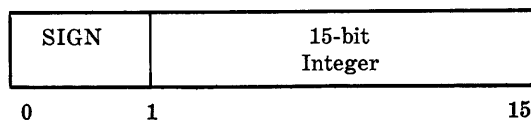
1. loading.
2. storing.
3. comparing.
4. shifting.
5. sign control.
6. radix conversion of fixed-point operands.
7. adding.
8. subtracting.
9. multiplying.
10. dividing.

The result of all sign control, compare, shift, add, and subtract operations is reflected in the condition code.

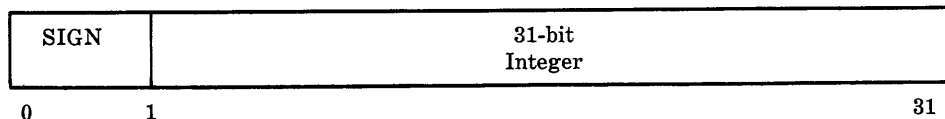
### DATA FORMAT

◆ A fixed-length format of a one-bit sign followed by the integer field makes up fixed-point numbers. In one of the general registers, the number is a 31-bit integer field. The complete 32-bit register is occupied by the fixed-point quantity and sign. A 64-bit operand, with a 63-bit integer field, is used by some shift, multiply, and divide instructions. A pair of adjacent registers, addressed by the even address of the leftmost register, contains these longer operands. The sign-bit of the rightmost register becomes part of the integer field. The same register can be specified for both operands in register-to-register operations (except for the Divide instructions). In main memory, fixed-point operands are in either a 32-bit word or a 16-bit halfword. The integer fields are then either 31 bits or 15 bits. Radix conversion operations always use a 64-bit decimal field. Integral storage boundaries for these units of data must be observed. Halfword, full-word, or double-word operands are addressed with one, two, or three low-order address bits set to zero. Half-word operands are extended to full words when they are fetched from main memory and used as a full-word operand.

Halfword Fixed-Point Number



Full-word Fixed-Point Number



### REPRESENTATION OF NUMBERS

◆ All fixed-point operands are treated as signed integers. True binary notation with a sign bit of zero is the representation of positive numbers. Two's-complement notation with a sign bit of one is the representation of negative numbers. To obtain the two's complement of a number, the value of each bit is changed and a one is added to the low-order bit.

**REPRESENTATION  
OF NUMBERS**  
(Cont'd)

This number representation can be regarded as the low-order part of an infinitely long representation of the number. A positive number has all zero bits, including the sign, to the left of the most significant bit of the number. A negative number has all one bits, including the sign, to the left of the most significant bit of the number. When an operand is to be extended with high-order bits, the extension is made by prefixing the operand with bits equal to the high-order bit of the operand.

A negative zero is not included in two's-complement notation. In the number range, the set of positive numbers is one less than the set of negative numbers. The *maximum* negative number is made up of an all-zero integer field with a one-bit sign. The maximum positive number consists of all 1's in the integer field with a zero-bit sign. The complement of the maximum negative number cannot be represented in the processor. For example, on a subtraction from zero that produces the complement of the maximum negative number, a fixed-point overflow exception is noted and the number remains unchanged. If the final result is within the representable range, then an overflow does not result (such as a subtraction from minus one). The representation of the product of two maximum negative numbers is a double-length positive number.

An overflow carries into the leftmost bit, which is the sign, and changes it. In algebraic shifting, however, the sign bit is unchanged even when significant bits in a shift left instruction are shifted out.

**INSTRUCTION  
FORMATS**

◆ The following three formats (RS, RX, RR) are used for fixed-point operations:

**RS Format**

| Op Code | R <sub>1</sub> | R <sub>3</sub> | B <sub>2</sub> | D <sub>2</sub> |
|---------|----------------|----------------|----------------|----------------|
| 0       | 7 8            | 11 12          | 15 16          | 19 20          |
|         |                |                |                | 31             |

*Description*

◆ An address is formed by adding the contents of the general register specified by B<sub>2</sub> to the displacement of field D<sub>2</sub>. The address formed is that of the main memory location of the second operand in the Load and Store Multiple instructions. In the shift operations, the result formed designates the amount of shift. The R<sub>1</sub> and R<sub>3</sub> fields specify the general register boundaries for Load and for Store Multiple instructions. In shift operations, R<sub>1</sub> specifies the general register holding the first operand, and R<sub>3</sub> is ignored.

**RX Format**

| Op Code | R <sub>1</sub> | X <sub>2</sub> | B <sub>2</sub> | D <sub>2</sub> |
|---------|----------------|----------------|----------------|----------------|
| 0       | 7 8            | 11 12          | 15 16          | 19 20          |
|         |                |                |                | 31             |

*Description*

◆ An address is formed by adding the contents of general registers specified by the X<sub>2</sub> and B<sub>2</sub> fields to the displacement field D<sub>2</sub>. This address specifies the main memory location of the second operand in the operation. The R<sub>1</sub> field designates the general register containing the first operand.

**RR Format**

| Op Code | R <sub>1</sub> | R <sub>2</sub> |
|---------|----------------|----------------|
| 0       | 7 8            | 11 12          |
|         |                | 15             |

*Description*

◆ In this format, the R<sub>1</sub> field specifies the general register holding the first operand. The R<sub>2</sub> field specifies the general register holding the second operand. The same register can be specified for both operands.

**Notes**

- ◆ 1. A zero in an  $X_2$  or  $B_2$  field indicates there is no corresponding address component to enter in the forming of an address in either the RX or RS format.
- 2. Except for the instructions Store and Convert to Decimal, results of fixed-point operations replace the first operand.
- 3. Except for storing the result, the contents of general registers and main memory locations used in the operations are not changed.
- 4. It is possible to designate the same general register both for operand locations and for address modification. Address modification occurs prior to operation execution.

**CONDITION CODE  
UTILIZATION**

- ◆ The condition code indicates the results of fixed-point sign control, add, subtract, shift, and compare instructions. The code is not changed by any other fixed-point instruction. Decision making by branch on condition operations can be done after those instructions which set the code.

For most arithmetic instructions, the Condition Codes 0, 1, or 2 indicate respectively a zero, less than zero, or greater than zero result. Condition Code 3 is set for overflow result. In comparison instructions, the Condition Codes 0, 1, or 2 indicate that the first operand is equal to, less than, or greater than the second operand. In add and subtract logical instructions, the Condition Codes 2 and 3 indicate either a zero or non-zero result with a carry from the sign bit. The Condition Codes 0 and 1 indicate the same conditions with no carry out of the sign position. Instructions that cause the condition code to be set and the meaning of the setting are as follows:

| Instruction        | Condition Code Setting |          |            |          |
|--------------------|------------------------|----------|------------|----------|
|                    | 0                      | 1        | 2          | 3        |
| Add Word           | Zero                   | < Zero   | > Zero     | Overflow |
| Add Halfword       | Zero                   | < Zero   | > Zero     | Overflow |
| Add Logical        | Zero                   | Not Zero | Zero Carry | Carry    |
| Compare Word       | Equal                  | Low      | High       | —        |
| Compare Halfword   | Equal                  | Low      | High       | —        |
| Load and Test      | Zero                   | < Zero   | > Zero     | —        |
| Load Complement    | Zero                   | < Zero   | > Zero     | Overflow |
| Load Negative      | Zero                   | < Zero   | —          | —        |
| Load Positive      | Zero                   | —        | > Zero     | Overflow |
| Shift Left Double  | Zero                   | < Zero   | > Zero     | Overflow |
| Shift Left Single  | Zero                   | < Zero   | > Zero     | Overflow |
| Shift Right Double | Zero                   | < Zero   | > Zero     | —        |
| Shift Right Single | Zero                   | < Zero   | > Zero     | —        |
| Subtract Word      | Zero                   | < Zero   | > Zero     | Overflow |
| Subtract Halfword  | Zero                   | < Zero   | > Zero     | Overflow |
| Subtract Logical   | —                      | Not Zero | Zero Carry | Carry    |

## **INTERRUPT ACTION**

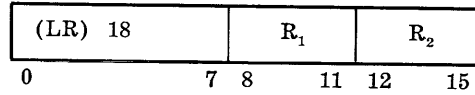
|                             |   |
|-----------------------------|---|
|                             | <p>◆ The following interrupt conditions can occur as a result of fixed-point instructions:</p>  |
| <b>Address Error</b>        |   |
| <i>Addressing</i>           | <p>◆ An address error interrupt occurs when an address specifies a location outside the available main memory. The operation is terminated at the point of error. The result data and the condition code, if produced, are unpredictable.</p>   |
| <i>Specification</i>        | <p>◆ An address error interrupt occurs when an instruction specifies a:</p> <ol style="list-style-type: none"> <li>1. Full-word operand that is not located on a 32-bit boundary.</li> <li>2. Halfword operand that is not located on a 16-bit boundary.</li> <li>3. Double-word operand that is not located on a 64-bit boundary.</li> <li>4. Register with an odd-numbered address when using an even/odd pair containing a 64-bit operand.</li> </ol> <p>The instruction is suppressed. The condition code, data in main memory, and registers remain unchanged.</p> |
| <i>Protection</i>           | <p>◆ An address error interrupt occurs when the storage key and the protection key of the result location do not match. The operation is suppressed and the condition code and data in the registers and main memory are unaltered. The only exception is the Store Multiple instruction which is terminated. The amount of data stored is unpredictable. (This interrupt can only occur if the memory protect feature is installed.)</p>   |
| <b>Data Error</b>           | <p>◆ A data error interrupt occurs when an invalid digit or sign code of the decimal operand is encountered in the Convert to Binary instruction. The operation is suppressed and the condition code and data in the register and main memory are unaltered.</p>  |
| <b>Fixed-Point Overflow</b> | <p>◆ A fixed-point overflow interrupt occurs when the results overflow in sign control, add, subtract or shift operations. The operation is completed by placing the truncated result in the register and setting Condition Code 3. Overflow bits are lost. If the fixed point program mask bit is reset, interrupt will not occur and the flag in the IFR will not be set.</p>   |
| <b>Divide Error</b>         | <p>◆ A divide error interrupt occurs when the quotient would exceed the register size in division, or the result of a Convert to Binary instruction exceeds 31 bits. The operation is suppressed and the data in the registers remains unaltered.</p>   |

**Load Word  
(LR) (L)**

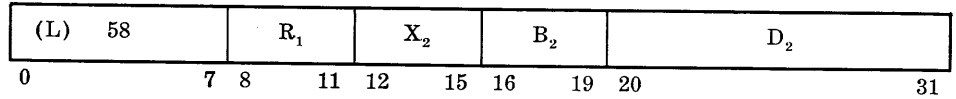
**General Description**

◆ The operand specified by the second address ( $R_2$  or  $X_2/B_2/D_2$ ) is loaded into the general register specified by the first address ( $R_1$ ).

**Format  
(RR)**



**(RX)**



**Condition Code**

◆ Unchanged.

**Interrupt Action**

◆ Address error:

Addressing (RX format).

Specification (RX format).

**Note**

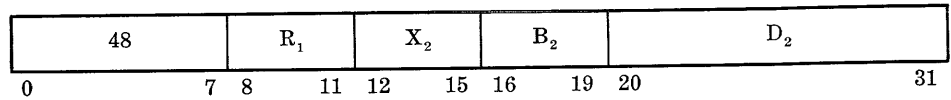
◆ The operand in the register or main memory location specified by the second address remains unchanged.

## **Load Halfword (LH)**

### **General Description**

◆ The halfword operand in the main memory specified by the second address ( $X_2/B_2/D_2$ ) is loaded into the general register specified by the first address ( $R_1$ ).

### **Format (RX)**



### **Condition Code**

◆ Unchanged.

### **Interrupt Action**

◆ Address error:  
Addressing.  
Specification.

### **Notes**

- ◆ 1. When the halfword (second operand) is fetched from main memory, it is expanded to a full word by propagating the sign-bit value through the 16 high-order positions of the receiving register.
- 2. The operand specified by the second address is unaltered.

## **Load and Test (LTR)**

### **General Description**

- ◆ The operand in the register specified by the second address ( $R_2$ ) is loaded into the general register specified by the first address ( $R_1$ ). The condition code is determined by the magnitude and the sign of the loaded operand.

### **Format (RR)**

|    |   |   |    |       |    |       |  |
|----|---|---|----|-------|----|-------|--|
| 12 |   |   |    | $R_1$ |    | $R_2$ |  |
| 0  | 7 | 8 | 11 | 12    | 15 |       |  |

### **Condition Code**

- ◆ 0 — result is zero.
- 1 — result is less than zero.
- 2 — result is greater than zero.
- 3 — not used.

### **Interrupt Action**

- ◆ None.

### **Notes**

- ◆ 1. The same register can be specified for both  $R_1$  and  $R_2$ . If this is done, the operation is equivalent to a test with no data movement.
- 2. The operand specified by the second address ( $R_2$ ) is unaltered.

## Load Complement (LCR)

### General Description

◆ The two's complement of the operand in the register specified by the second address ( $R_2$ ) is loaded into the general register specified by the first address ( $R_1$ ). The condition code is determined by the magnitude and the sign of the loaded operand.

### Format (RR)

|    |  |  |  |  |  |   |   |       |  |    |       |  |  |    |
|----|--|--|--|--|--|---|---|-------|--|----|-------|--|--|----|
| 13 |  |  |  |  |  |   |   | $R_1$ |  |    | $R_2$ |  |  |    |
| 0  |  |  |  |  |  | 7 | 8 |       |  | 11 | 12    |  |  | 15 |

### Condition Code

◆ 0 — result is zero.  
 1 — result is less than zero.  
 2 — result is greater than zero.  
 3 — overflow.

### Interrupt Action

◆ Fixed-point overflow.

### Notes

- ◆ 1. Zero operands remain constant and unchanged under complementation.
- 2. A fixed-point overflow interrupt occurs when the maximum negative number is complemented.
- 3. The operand specified by the second address is unaltered.



# **Load Positive (LPR)**

## **General Description**

◆ The operand in the register specified by the second address ( $R_2$ ) is made positive, if negative, and loaded into the general register specified by the first address ( $R_1$ ). In loading the absolute value of the operand, negative numbers are complemented and positive numbers remain unaltered. The magnitude of the absolute value determines the condition code.

## **Format (RR)**

|    |  |  |  |  |  |   |   |       |  |    |    |       |  |    |  |
|----|--|--|--|--|--|---|---|-------|--|----|----|-------|--|----|--|
| 10 |  |  |  |  |  |   |   | $R_1$ |  |    |    | $R_2$ |  |    |  |
| 0  |  |  |  |  |  | 7 | 8 |       |  | 11 | 12 |       |  | 15 |  |

## **Condition Code**

- ◆ 0 — result is zero.
- 1 — not used.
- 2 — result greater than zero.
- 3 — overflow on complement.

## **Interrupt Action**

- ◆ Fixed-point overflow.

## **Notes**

- ◆ 1. A fixed-point overflow interrupt exists if a maximum negative number is complemented.
- 2. The operand specified by the second address is unaltered.

## **Load Negative (LNR)**

### **General Description**

◆ The two's complement of the operand in the register specified by the second address ( $R_2$ ) is loaded into the general register specified by the first address ( $R_1$ ). In loading the operand value, positive numbers are complemented and negative numbers remain unaltered. The magnitude of the loaded value determines the condition code setting.

### **Format (RR)**

|    |   |   |    |       |    |       |  |
|----|---|---|----|-------|----|-------|--|
| 11 |   |   |    | $R_1$ |    | $R_2$ |  |
| 0  | 7 | 8 | 11 | 12    | 15 |       |  |

### **Condition Code**

◆ 0 — result is zero.  
 1 — result is less than zero.  
 2 — not used.  
 3 — not used.

### **Interrupt Action**

◆ None.

### **Notes**

◆ 1. A zero operand is not altered and retains a positive sign.  
 2. The operand specified by the second address is unaltered.

## Load Multiple (LM)

### General Description

◆ The set of general registers, beginning with the register specified by the first address ( $R_1$ ) and ending with the register specified by the third address ( $R_3$ ), is loaded with operands from main memory. The second address ( $B_2/D_2$ ) specifies the main memory location of the first word to be loaded. Loading of the general registers continues in the ascending order of their addresses beginning with the register specified by  $R_1$ . As many words as needed are fetched from the main memory location specified, continuing up to, and including, the register specified by  $R_3$ .

### Format (RS)

|    |       |       |       |       |
|----|-------|-------|-------|-------|
| 98 | $R_1$ | $R_3$ | $B_2$ | $D_2$ |
| 0  | 7 8   | 11 12 | 15 16 | 19 20 |
|    |       |       |       | 31    |

### Condition Code

◆ Unchanged.

### Interrupt Action

◆ Address error:  
Addressing.  
Specification.

### Notes

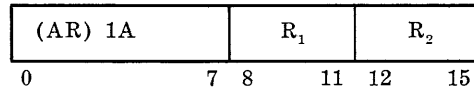
- ◆ 1. If  $R_1$  and  $R_3$  specify the same register, only one word is loaded.
- ◆ 2. If the register specified by  $R_3$  is less than the register specified by  $R_1$ , wrap-around occurs from register 15 to 0.
- ◆ 3. The operands specified by the second address are unaltered.

**Add Word**  
**(AR) (A)**

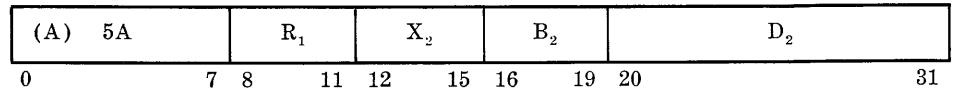
**General Description**

◆ The operand specified by the first address ( $R_1$ ) is added to the operand specified by the second address ( $R_2$  or  $X_2/B_2/D_2$ ) and the sum is placed in the general register specified by the first address ( $R_1$ ). The magnitude and the sign of the sum determine the condition code setting.

**Format**  
**(RR)**



**(RX)**



**Condition Code**

- ◆ 0 — sum is zero.
- 1 — sum is less than zero.
- 2 — sum is greater than zero.
- 3 — overflow.

**Interrupt Action**

- ◆ Fixed-point overflow.
- ◆ Address error:
  - Addressing (RX format).
  - Specification (RX format).

**Notes**

- ◆ 1. All 32 bits of both operands participate in the addition. If the high-order numeric bit position of the result and the carries out of the sign bit position disagree, an overflow condition exists. The overflow does not alter the sign bit created by the carries.
- 2. A negative overflow results in a positive sum and a positive overflow results in a negative sum with overflow bits being lost.
- 3. A zero result is always positive.
- 4. The operand specified by the second address is unaltered.

## Add Halfword (AH)

### General Description

- ◆ The halfword operand specified by the second address ( $X_2/B_2/D_2$ ) is added to the operand specified by the first address ( $R_1$ ) and the sum is placed into the register specified by the first address ( $R_1$ ). The sign and the magnitude of the sum determine the condition code setting.

### Format (RX)

|    |  |  |  |  |  |  |  |       |   |  |  |       |    |  |  |       |    |  |  |       |    |  |  |  |  |  |  |  |  |  |  |  |    |
|----|--|--|--|--|--|--|--|-------|---|--|--|-------|----|--|--|-------|----|--|--|-------|----|--|--|--|--|--|--|--|--|--|--|--|----|
| 4A |  |  |  |  |  |  |  | $R_1$ |   |  |  | $X_2$ |    |  |  | $B_2$ |    |  |  | $D_2$ |    |  |  |  |  |  |  |  |  |  |  |  |    |
| 0  |  |  |  |  |  |  |  | 7     | 8 |  |  | 11    | 12 |  |  | 15    | 16 |  |  | 19    | 20 |  |  |  |  |  |  |  |  |  |  |  | 31 |

### Condition Code

- ◆ 0 — sum is zero.
- 1 — sum is less than zero.
- 2 — sum is greater than zero.
- 3 — overflow

### Interrupt Action

- ◆ Fixed-point overflow.
- ◆ Address error:  
Addressing.  
Specification.

### Notes

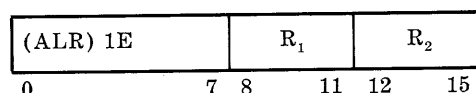
- ◆ 1. The halfword in main memory specified by the second address is expanded to full-word length prior to the addition by propagating the sign bit value through the high-order 16 positions. The addition is completed by adding all 32 bits of both operands.
- 2. An overflow exists if the high-order numeric result bit and the carry out of the sign-bit position disagree. The sign is not corrected after overflow occurs. A negative overflow results in a positive sum and a positive overflow results in a negative sum with the overflow bits being lost.
- 3. The operand specified by the second address is unaltered.

# **Add Logical (ALR) (AL)**

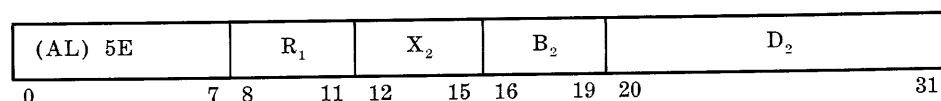
## **General Description**

◆ The operand specified by the second address ( $R_2$  or  $X_2/B_2/D_2$ ) is logically added (32-bit unsigned) to the operand specified by the first address ( $R_1$ ). The sum is placed in the general register specified by the first address. The condition code is determined by the relation of the sum to a zero number and the occurrence of a carry out of the sign bit position. An overflow on such carries is not recognized and does not set an interrupt condition.

## **Format (RR)**



## **(RX)**



## **Condition Code**

- ◆ 0 — sum is zero and no carry.
- 1 — sum is not zero and no carry.
- 2 — sum is zero with a carry.
- 3 — sum is not zero with a carry.

## **Interrupt Action**

- ◆ Address error:
  - Addressing (RX format).
  - Specification (RX format).

## **Notes**

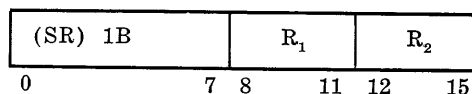
- ◆ 1. All 32 bits of the operands participate in the logical addition.
- 2. The operand specified by the second address is unaltered.

# **Subtract Word (SR) (S)**

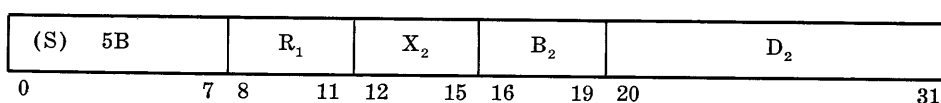
## **General Description**

◆ The operand specified by the second address ( $R_2$  or  $X_2/B_2/D_2$ ) is subtracted from the operand specified by the first address ( $R_1$ ) and the difference is placed in the general register specified by the first address ( $R_1$ ). The magnitude and the sign of the difference determine the condition code setting.

## **Format (RR)**



## **(RX)**



## **Condition Code**

- ◆ 0 — difference is zero.
- 1 — difference is less than zero.
- 2 — difference is greater than zero.
- 3 — overflow.

## **Interrupt Action**

- ◆ Fixed-point overflow.
- Address error:
  - Addressing (RX format).
  - Specification (RX format).

## **Notes**

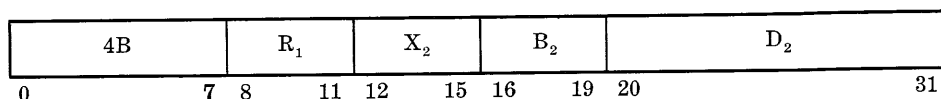
- ◆ 1. The operation is accomplished by adding the one's complement of the second operand and a one in the low-order position of the first operand. The one's complement of a number is obtained by changing all the 1 bits to 0 bits and all the 0 bits to 1 bits. All 32 bits are involved in the operation. An overflow exists if the high-order numeric result bit and the carry out of the sign bit position disagree.
- 2. The difference between a maximum negative number and another maximum negative number is zero with no overflow.
- 3. When the same register is specified for  $R_1$  and  $R_2$ , the operation is equivalent to clearing  $R_1$  to zero.
- 4. The operand specified by the second address is unaltered.

## Subtract Halfword (SH)

### General Description

- ◆ The halfword operand specified by the second address ( $X_2/B_2/D_2$ ) is expanded and subtracted from the operand specified by the first address ( $R_1$ ). The difference is placed in the general register specified by  $R_1$ . The sign and the magnitude of the difference determine the condition code setting.

### Format (RX)



### Condition Code

- ◆ 0 — difference is zero.
- 1 — difference is less than zero.
- 2 — difference is greater than zero.
- 3 — overflow.

### Interrupt Action

- ◆ Fixed-point overflow.
- Address error:  
Addressing.  
Specification.

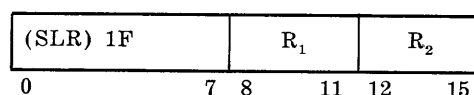
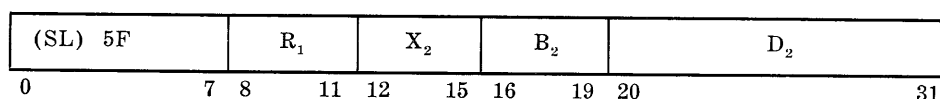
### Notes

- ◆ 1. The halfword in main memory specified by the second address is expanded to full-word length by propagating the sign bit value through the 16 high-order positions.
- 2. The subtraction is completed by adding the one's complement of the second operand and a one in the low-order position of the first operand. All 32 bits are involved in the operation.
- 3. An overflow exists if the high-order numeric result bit and the carry out of the sign bit position disagree.
- 4. The operand specified by the second address is unaltered.



**Subtract Logical  
(SLR) (SL)****General Description**

◆ The operand specified by the second address ( $R_2$  or  $X_2/B_2/D_2$ ) is logically subtracted (32-bit unsigned) from the operand specified by the first address ( $R_1$ ). The difference is placed in the general register specified by the first address. The condition code is determined by the relation of the sum to a zero number and the occurrence of a carry out of the sign bit position. An overflow on such carries is not recognized and does not set an interrupt condition.

**Format  
(RR)****(RX)****Condition Code**

- ◆ 0 — not used.  
 1 — difference is not zero and no carry.  
 2 — difference is zero with a carry.  
 3 — difference is not zero with a carry.

**Interrupt Action**

- ◆ Address error:  
     Addressing (RX format).  
     Specification (RX format).

**Notes**

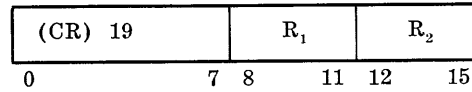
- ◆ 1. A zero difference cannot occur without a carry out of the sign position.  
 2. Logical subtraction is accomplished by adding the one's complement of the second operand and a one in the low-order position of the first operand.  
 3. All 32 bits of the operands participate in the logical subtraction without change to the resulting sign bit.  
 4. The operand specified by the second address is unaltered.

## Compare Word (CR) (C)

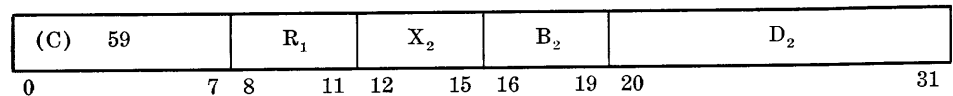
### General Description

◆ The operand specified by the first address ( $R_1$ ) is compared with the operand specified by the second address ( $R_2$  or  $X_2/B_2/D_2$ ). Both operands remain unaltered. The result of the comparison determines the condition code setting.

### Format (RR)



### (RX)



### Condition Code

- ◆ 0 — operands are equal.
- 1 — the operand specified by the first address is low.
- 2 — the operand specified by the first address is high.
- 3 — not used.

### Interrupt Action

- ◆ Address error:
  - Addressing (RX format).
  - Specification (RX format).

### Note

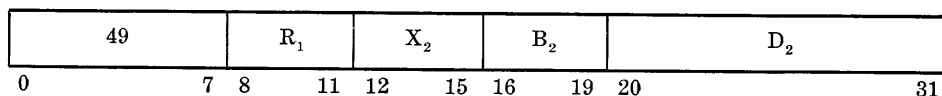
- ◆ Both operands are considered as 32-bit signed integers and the comparison is algebraic.

## Compare Halfword (CH)

### General Description

◆ The operand specified by the first address ( $R_1$ ) is compared with the halfword operand expanded to a full word, specified by the second address ( $X_2/B_2/D_2$ ). Both operands remain unaltered. The result of the comparison determines the condition code setting.

### Format (RX)



### Condition Code

- ◆ 0 — operands are equal.
- 1 — the operand specified by the first address is low.
- 2 — the operand specified by the first address is high.
- 3 — not used.

### Interrupt Action

- ◆ Address error:  
Addressing.  
Specification.

### Notes

- ◆ 1. The halfword in storage specified by the second address is expanded to full-word length by propagating the sign bit value through the 16 high-order positions.
- 2. Both operands are considered as 32-bit signed integers and the comparison is algebraic.

**Multiply Word**  
**(MR) (M)****General Description**

◆ The operand (multiplicand) specified by the first address ( $R_1$ ) is multiplied by the operand (multiplier) specified by the second address ( $R_2$  or  $X_2/B_2/D_2$ ). The double-length product is loaded into the register specified by the first address ( $R_1$ ), which must be an even number, and the next odd-numbered register.

**Format**  
**(RR)**

|         |       |          |
|---------|-------|----------|
| (MR) 1C | $R_1$ | $R_2$    |
| 0       | 7 8   | 11 12 15 |

**(RX)**

|        |       |          |          |       |
|--------|-------|----------|----------|-------|
| (M) 5C | $R_1$ | $X_2$    | $B_2$    | $D_2$ |
| 0      | 7 8   | 11 12 15 | 16 19 20 | 31    |

**Condition Code**

◆ Unchanged.

**Interrupt Action**

◆ Address error:  
Addressing (RX format).  
Specification.

**Notes**

- ◆ 1. The first address ( $R_1$ ) must always refer to the even-numbered register of an even/odd pair. The multiplicand is taken from the odd-numbered register of the pair. The original contents of the even-numbered register, which is replaced by the product, is ignored. An overflow cannot occur.
- 2. Only when two maximum negative numbers are multiplied does the product exceed 62 significant bits. This product produces 63 significant bits.
- 3. In two's-complement notation, the sign bit is propagated right, up to the first significant product bit.
- 4. The sign of the product is determined algebraically. A zero result is always positive.
- 5. The least significant digit of the product goes into the odd-numbered register.
- 6. The operand specified by the second address (multiplier) is unaltered except when the first and second addresses specify the same (even numbered) register. In this case the multiplier is taken from the even register, the multiplicand is taken from the odd register and the product is placed into the even/odd pair.

## Multiply Halfword (MH)

### General Description

◆ The operand (multiplicand) specified by the first address ( $R_1$ ) is multiplied by the halfword operand (multiplier) specified by the second address ( $X_2/B_2/D_2$ ). The product of the operands replaces the contents of the register specified by the first address ( $R_1$ ).

### Format (RX)

|    |       |       |       |       |
|----|-------|-------|-------|-------|
| 4C | $R_1$ | $X_2$ | $B_2$ | $D_2$ |
| 0  | 7 8   | 11 12 | 15 16 | 19 20 |
|    |       |       |       | 31    |

### Condition Code

◆ Unchanged.

### Interrupt Action

◆ Address error:  
Addressing.  
Specification.

### Notes

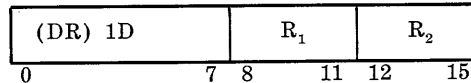
- ◆ 1. The halfword operand in main memory is expanded to a full word before multiplication by propagating the sign bit value through the 16 high-order positions. Both operands are considered as 32-bit signed integers. The multiplicand is replaced by the low order 32 bits of the product.
- 2. The product usually occupies 46 bits of significance except when both operands are maximum negative numbers and occupy 47 bits.
- 3. The bits to the left of the 32 low-order bits of the product are not tested for significance. No overflow indication is given. Since the bits to the left of the low-order 32 are ignored, the sign of the result may differ from the true sign of the product, if the product exceeds 32 bits.
- 4. The operand specified by the second address is unaltered.
- 5. A zero product is always positive.

**Divide  
(DR) (D)**

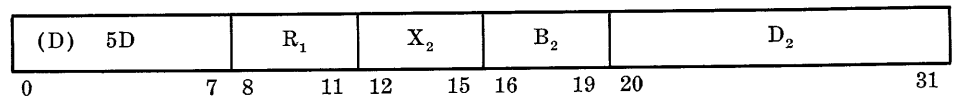
**General Description**

◆ The double-word operand (dividend) specified by the first address ( $R_1$ ) is divided by the operand (divisor) specified by the second address ( $R_2$  or  $X_2/B_2/D_2$ ). The quotient and remainder replace the double-word operand in the registers specified by the first address ( $R_1$ ). The register specified by the first address must be the even-numbered register of an even/odd pair.

**Format  
(RR)**



**(RX)**



**Condition Code**

◆ Unchanged.

**Interrupt Action**

◆ Address error:  
Addressing (RX format).  
Specification.  
Divide Error.

**Notes**

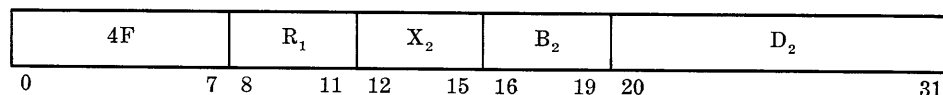
- ◆ 1. The dividend, a 64-bit signed integer, is replaced by a 32-bit signed quotient and a 32-bit signed remainder; the remainder is placed in the even-numbered register and the quotient is placed in the odd-numbered register. The divisor is a 32-bit signed integer and is unaltered.
- 2. A divide error interrupt occurs when the magnitude of the dividend to the divisor is such that the quotient cannot be expressed by a 32-bit signed integer. (The divisor must be greater in absolute value than the first word of the dividend.)
- 3. The sign of the quotient is determined algebraically except that a zero quotient as a zero remainder is always positive.
- 4. The remainder has the same sign as the dividend.

## Convert to Binary (CVB)

### General Description

◆ The radix of the double-word operand in main memory specified by the second address ( $X_2/B_2/D_2$ ) is converted from decimal to binary notation and loaded into the general register specified by the first address ( $R_1$ ). The operand in main memory is treated as a right-justified signed integer before and after the conversion.

### Format (RX)



### Condition Code

◆ Unchanged.

### Interrupt Action

◆ Address error:  
Addressing.  
Specification.  
Data error.  
Divide error.

### Notes

- ◆ 1. The double-word operand in main memory (15 digits plus sign) must be in the packed decimal format. The operand is checked for valid sign and digit codes. The sign representation depends on the current decimal code (ASCII or EBCDIC).
- 2. The maximum decimal number that can be converted and still be contained in a 32-bit register is  $(2,147,483,647)_{10}$  positive and  $(2,147,483,648)_{10}$  negative. A larger decimal number causes a divide error interrupt.
- 3. Negative decimal zero is converted to positive binary zero.
- 4. The operand specified by the second address remains unaltered in main memory.

**Convert to Decimal  
(CVD)****General Description**

◆ The radix of the operand specified by the first address ( $R_1$ ) is converted from binary to decimal notation and stored at the double-word main memory area specified by the second address ( $X_2/B_2/D_2$ ). The operand is treated as a right-justified signed integer before and after the conversion.

**Format  
(RX)**

|    |  |  |  |  |  |  |  |       |   |  |    |       |  |  |    |       |  |  |    |       |  |  |  |  |  |  |  |  |  |  |  |  |    |
|----|--|--|--|--|--|--|--|-------|---|--|----|-------|--|--|----|-------|--|--|----|-------|--|--|--|--|--|--|--|--|--|--|--|--|----|
| 4E |  |  |  |  |  |  |  | $R_1$ |   |  |    | $X_2$ |  |  |    | $B_2$ |  |  |    | $D_2$ |  |  |  |  |  |  |  |  |  |  |  |  |    |
| 0  |  |  |  |  |  |  |  | 7     | 8 |  | 11 | 12    |  |  | 15 | 16    |  |  | 19 | 20    |  |  |  |  |  |  |  |  |  |  |  |  | 31 |

**Condition Code**

◆ Unchanged.

**Interrupt Action**

◆ Address error:  
Addressing.  
Specification.  
Protection.

**Notes**

- ◆ 1. The result is placed in the double-word main memory location in the packed decimal format of 15 digits plus sign.
- 2. The low-order four bits of the result are the sign which is generated according to the current decimal code, EBCDIC or ASCII.
- 3. The maximum binary number (32-bit signed integer) that can be converted is (2,147,483,647) positive and (2,147,483,648) negative. No overflow can occur.

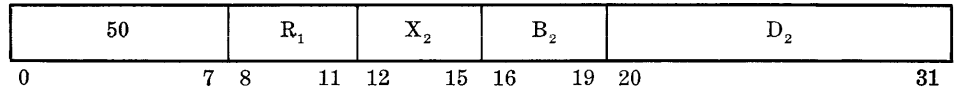


## Store Word (ST)

### General Description

◆ The operand in the general register specified by the first address ( $R_1$ ) is stored in the main memory location specified by the second address ( $X_2/B_2/D_2$ ).

### Format (RX)



### Condition Code

◆ Unchanged.

### Interrupt Action

◆ Address error:  
Addressing.  
Specification.  
Protection.

### Notes

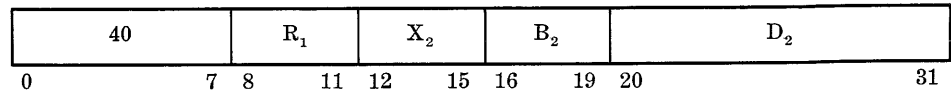
- ◆ 1. The complete contents (32 bits) of the general register specified by the first address are placed unaltered in main memory.
- 2. The operand specified by the first address is unaltered.

## Store Halfword (STH)

### General Description

- ◆ The rightmost half (16 bits) of the operand in the general register specified by the first address ( $R_1$ ) is stored unaltered in the halfword main memory location specified by the second address ( $X_2/B_2/D_2$ ).

### Format (RX)



### Condition Code

- ◆ Unchanged.

### Interrupt Action

- ◆ Address error:  
Addressing.  
Specification.  
Protection.

### Notes

- ◆ 1. The 16 high-order bits of the operand specified by the first address field are ignored by the operation.
- 2. The operand specified by the first address is unaltered.

## Store Multiple (STM)

### General Description

◆ The operands in the set of general registers, beginning with the register specified by the first address ( $R_1$ ) and ending with the register specified by the third address ( $R_3$ ), are stored in main memory locations starting with the location specified by the second address ( $B_2/D_2$ ). The second address ( $B_2/D_2$ ) refers to the main memory location where the first operand (word) is to be stored. Storing of the operands continues in the ascending order of the register number specified by  $R_1$ , up to and including  $R_3$ , storing as many words as indicated in the main memory locations that immediately follow the initial operand.

### Format (RS)

|    |       |       |       |       |    |    |    |    |    |
|----|-------|-------|-------|-------|----|----|----|----|----|
| 90 | $R_1$ | $R_3$ | $B_2$ | $D_2$ |    |    |    |    |    |
| 0  | 7     | 8     | 11    | 12    | 15 | 16 | 19 | 20 | 31 |

### Condition Code

◆ Unchanged.

### Interrupt Action

◆ Address error:  
Addressing.  
Specification.  
Protection.

### Notes

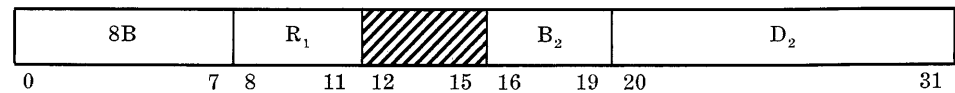
- ◆ 1. If the same register is specified for  $R_1$  and  $R_3$ , only one word is stored.
- ◆ 2. If  $R_3$  is less than  $R_1$ , the register addresses wrap around from 15 to 0. For instance, all registers can be stored by making  $R_3$  one less than  $R_1$ .
- ◆ 3. The operands in the set of registers designated are unaltered.

## Shift Left Single (SLA)

### General Description

◆ The integer portion of the operand in the general register specified by the first address ( $R_1$ ) is shifted left the number of positions specified by the second address ( $B_2/D_2$ ). The second address is used as a count and not to address data. The low-order six bits of the second address constitute the count. The remaining bits are ignored.

### Format (RS)



### Condition Code

- ◆ 0 — result is zero.
- 1 — result is less than zero.
- 2 — result is greater than zero.
- 3 — overflow.

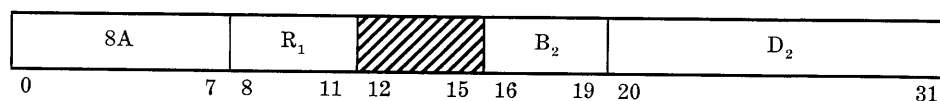
### Interrupt Action

- ◆ Fixed-point overflow.

### Notes

- ◆ 1. All 31 bit positions of the integer are shifted. The sign is not altered. Zeros are inserted in the right-hand end of the operand for each shift.
- 2. If a bit is shifted out of the left-hand end that is not identical to the sign bit, a fixed-point overflow condition exists.

- ◆ The integer portion of the operand in the general register specified by the first address ( $R_1$ ) is shifted right the number of positions specified by the second address ( $B_2/D_2$ ). The second address is used as a count and not to address data. The low-order six bits of the second address field constitute the count. The remaining bits are ignored.



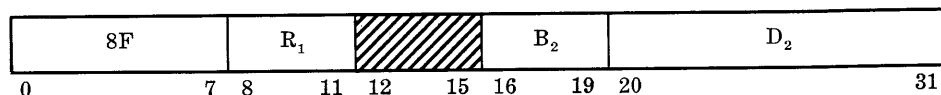
- ◆ 0 — result is zero.
- 1 — result is less than zero.
- 2 — result is greater than zero.
- 3 — not used.

◆ None.

- ◆ 1. All 31 bit positions of the integer are shifted. The sign is not altered. The sign bit is propagated through the positions vacated in the left end of the operand. The bits shifted out to the right are lost.
- 2. Shifting to the right is equivalent to low-order truncation or division by powers of two.
- 3. Shifts greater than 31 cause all significant bits to be lost. A zero for positive numbers and a minus one for negative numbers is the result of such shifts.
- 4. Fixed-point positive numbers go towards zero; Fixed-point negative numbers go towards minus one.

**Shift Left Double  
(SLDA)****General Description**

◆ The integer portion of the double-word operand specified by the first address ( $R_1$ ) and the first address plus one is shifted left the number of positions specified by the second address ( $B_2/D_2$ ). The first address ( $R_1$ ) specifies an even-numbered register of an even/odd pair that contains the 63-bit integer to be shifted. The second address is used as a count and not to address data. The low-order six bits of the second address field constitute the count. The remaining bits are ignored.

**Format  
(RS)****Condition Code**

- ◆ 0 — result is zero.  
 1 — result is less than zero.  
 2 — result is greater than zero.  
 3 — overflow.

**Interrupt Action**

- ◆ Fixed-point overflow.  
 Address error:  
 Specification.

**Notes**

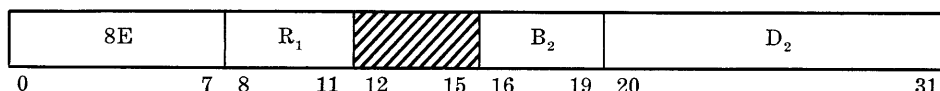
- ◆ 1. All 63 bit positions of the integer are shifted. The sign bit (position 0) in the even register is not altered. Zeros are inserted in the right-hand end of the double-word operand for each shift.
2. If a bit is shifted out of the left-hand end that is not identical to the sign bit, a fixed-point overflow condition exists.

# Shift Right Double (SRDA)

## General Description

◆ The integer portion of the double-word operand specified by the first address ( $R_1$ ) and the first address plus one is shifted right the number of positions specified by the second address ( $B_2/D_2$ ). The first address ( $R_1$ ) specifies an even-numbered register of an even/odd pair that contains the 63-bit integer to be shifted. The second address is used as a count and not to address data. The low-order six bits of the second address constitute the count. The remaining bits are ignored.

## Format (RS)



## Condition Code

- ◆ 0 — result is zero.
- 1 — result is less than zero.
- 2 — result is greater than zero.
- 3 — not used.

## Interrupt Action

- ◆ Address error:  
Specification.

## Notes

- ◆ 1. All 63 bit positions of the integer are shifted. The sign bit in the leftmost position of the even-numbered register is not altered. This sign bit is propagated through the positions vacated in the left end of the double-word operand. The bits shifted out to the right are lost.
- 2. A shift count of zero provides a double-word sign and magnitude check.

# DECIMAL ARITHMETIC INSTRUCTIONS

## INTRODUCTION

◆ Decimal arithmetic is performed on data in packed format. In this format, two decimal digits are placed in one byte (four bits each). The operands may be variable in length, and must contain a sign in the rightmost four bits.

All decimal instructions are two-address, SS-type format. The instruction set includes addition, subtraction, comparison, multiplication, and division. Since data sent to, and from, external devices are usually in zoned (unpacked) format (one digit in one byte), there are also instructions for converting to, and from, packed and zoned format. All decimal arithmetic instructions are standard features on the 70/35, 45, and 55 processors.

## DATA FORMATS

### Packed Format

◆ The formats for decimal data in high-speed memory are:

| Byte  |       | Byte  |       | Byte  |       | Byte  |       | Byte  |       | Byte  |      |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|------|
| Digit | Digit | Digit | Digit | Digit | Digit | Digit | Digit | Digit | Digit | Digit | Sign |

In packed format, one byte represents two decimal digits. The rightmost half-byte (4 bits) of a field represents the sign.

### Zoned Format

| Byte |       | Byte |       | Byte |       | Byte |       | Byte |       | Byte |       |
|------|-------|------|-------|------|-------|------|-------|------|-------|------|-------|
| Zone | Digit | Zone | Digit | Zone | Digit | Zone | Digit | Zone | Digit | Sign | Digit |

In zoned format, the low-order four bits of each eight-bit byte contain the decimal digit and the high-order four bits contain the zone. The high-order four bits of the rightmost byte of a field contain the sign of the field.

### Description of Formats

◆ Decimal arithmetic instructions operate from right to left. The addresses specify the leftmost byte of the operand, and the length specifies the additional number of bytes that are to the right of the addressed byte. The fields specified by the addresses can be variable in length beginning at any byte in main memory and consisting of from 1 to 16 eight-bit bytes. Results of operations are always placed in the first operand field. The result never exceeds the limits set by the address and length of the first operand field. If a decimal arithmetic operation results in a carry outside the operand limits, a decimal overflow interrupt occurs. If the first operand is longer than the second, the second operand is extended with high-order zeros up to the length of the first operand during operation execution (in addition and subtraction only). This extension never changes main memory.

Because the code configurations of digits and sign are verified while arithmetic operations are performed, improper overlapping of fields is recognized as a data error. The arithmetic instruction set (except Pack, Unpack, Move with Offset) should not specify overlapping fields unless the rightmost byte of the fields coincide.

In the move-type instructions of this set (Pack, Unpack, Move with Offset), no checking is made for valid codes. Consequently, overlapping is permitted without any restrictions. (Although unusual results are possible, overlapping is dangerous.)



## REPRESENTATION OF NUMBERS

◆ Decimal operands in packed format are four-bit, binary-coded, decimal digits packed two to a byte. The operands may be variable in length and must contain a sign in the rightmost four bits of the rightmost byte. The digit and sign codes are as follows:

Digit and Sign Codes

| Digit | Code | Sign | Code |
|-------|------|------|------|
| 0     | 0000 | +    | 1010 |
| 1     | 0001 | —    | 1011 |
| 2     | 0010 | +    | 1100 |
| 3     | 0011 | —    | 1101 |
| 4     | 0100 | +    | 1110 |
| 5     | 0101 | +    | 1111 |
| 6     | 0110 |      |      |
| 7     | 0111 |      |      |
| 8     | 1000 |      |      |
| 9     | 1001 |      |      |

EBCDIC or ASCII sign or zone codes are generated for the decimal arithmetic results depending on the setting of the decimal code bit in the Interrupt Status Register. When the decimal code bit is set for EBCDIC, the following codes are generated:

| Sign |       | Zone |
|------|-------|------|
| Plus | Minus |      |
| 1100 | 1101  | 1111 |

When the decimal code bit is set for ASCII, the following codes are generated:

| Sign |       | Zone |
|------|-------|------|
| Plus | Minus |      |
| 1010 | 1011  | 0101 |

*Note:* The codes  $(1110)_2$  and  $(1111)_2$  are accepted as plus signs. However, if an arithmetic operation is performed on a field with these signs, the sign of the result will be in EBCDIC or ASCII, as shown above.

## INSTRUCTION FORMAT

◆ Decimal arithmetic instructions use the two-address, SS format as follows:

SS Format

| Op Code |  | L <sub>1</sub> |  | L <sub>2</sub> |  | B <sub>1</sub> |  | D <sub>1</sub> |  |    |  | B <sub>2</sub> |  | D <sub>2</sub> |  |    |  |  |  |    |  |  |  |    |  |  |  |    |  |    |  |    |  |    |  |  |  |
|---------|--|----------------|--|----------------|--|----------------|--|----------------|--|----|--|----------------|--|----------------|--|----|--|--|--|----|--|--|--|----|--|--|--|----|--|----|--|----|--|----|--|--|--|
| 0       |  | 7              |  | 8              |  | 11             |  | 12             |  | 15 |  |                |  | 16             |  | 19 |  |  |  | 20 |  |  |  | 31 |  |  |  | 32 |  | 35 |  | 36 |  | 47 |  |  |  |

### Description

◆ The contents of the general register specified by B<sub>1</sub> are added to the contents of the displacement field (D<sub>1</sub>) to obtain the main memory location of the leftmost byte of the first operand. The length (L<sub>1</sub>) of the first address specifies the *number of bytes that are to the right* of the location obtained above, thus giving the processor the address of the rightmost byte of the first operand. The length of the operand can be from one to 16 bytes, since

**SS Format  
(Cont'd)**

$L_1$  can be from 0000 to 1111. The address and size of the second operand is obtained in the same way using  $B_2$ ,  $D_2$  and  $L_2$ .

Results of operations are always stored in the first operand field and never exceed the limits specified by the address and length. The second operand is not changed in an add-type instruction unless the second operand addresses the same rightmost byte as the first operand.

*Note:* A zero in the  $B_1$  or  $B_2$  field indicates that no general register is to be used.

**CONDITION CODE  
UTILIZATION**

◆ The condition code is set as a result of all add-type and comparison operations. No other decimal arithmetic instructions affect the condition code.

The condition code setting has a different meaning for the comparison operation result than for the add-type result. The results of the following decimal arithmetic instructions cause the indicated condition code settings:

| Instruction      | Condition Code Setting |        |        |          |
|------------------|------------------------|--------|--------|----------|
|                  | 0                      | 1      | 2      | 3        |
| Add Decimal      | Zero                   | < Zero | > Zero | Overflow |
| Subtract Decimal | Zero                   | < Zero | > Zero | Overflow |
| Zero and Add     | Zero                   | < Zero | > Zero | Overflow |
| Compare Decimal  | Equal                  | Low    | High   | —        |

**INTERRUPT ACTION**

◆ The following interrupt conditions can occur as a result of a decimal arithmetic instruction.

**Address Error**

*Addressing*

◆ An address error interrupt exists when an address specifies a location outside the available main memory of the particular installation. The operation is terminated at the point of error. The result data and the condition code are unpredictable.

*Specification*

◆ An address error interrupt exists when a multiplier or divisor size exceeds 15 digits plus sign; or when the multiplier size or the divisor size is equal to, or greater than, the multiplicand or dividend size, respectively. The instruction is suppressed. The condition code, data in main memory, and registers remain unchanged.

*Protection*

◆ An address error interrupt exists when the protection key and the storage key of the result location do not match. The operation is terminated. The result data and condition code are unpredictable. (This interrupt can occur only if the memory protect feature is installed.)

**Data Error**

◆ A data error interrupt exists in decimal arithmetic when an invalid sign (not greater than nine) or digit code (not zero through nine) is detected in an operand, a multiplicand has insufficient high-order zeros, or there is incorrect overlapping of operands. The operation is terminated. The result data and the condition code setting are unpredictable.

|                         |   |
|-------------------------|---|
| <b>Decimal Overflow</b> | ◆ A decimal overflow interrupt exists when the result field of an Add Decimal, Subtract Decimal, or Zero and Add instruction is too small to contain the overflow data. The operation is completed by ignoring the overflow data, and setting the condition code to 3. If the decimal overflow program mask bit is reset, interrupt will not occur and the flag in the IFR will not be set. |
| <b>Divide Error</b>     | ◆ A divide error interrupt occurs when the quotient is greater than the specified data field, including division by zero, or the dividend does not have one leading zero. Division is suppressed and the dividend and divisor remain unchanged in main memory.  |

## Add Decimal (AP)

### General Description

◆ The operand specified by the second address ( $B_2/D_2$ ) is added algebraically to the operand specified by the first address ( $B_1/D_1$ ). The result is stored in the field specified by the first address. The sign and the magnitude of the sum determine the condition code.

The operands can be variable in length up to 16 bytes and must be in packed format. If operands overlap, their rightmost byte location must coincide.

The addition of the two operands can cause decimal overflow. Two conditions which cause overflow are:

1. a carry out of the high-order position of the result.
2. a second operand that is larger than the first operand and significant result positions are lost.

### Format (SS)

| FA | L <sub>1</sub> | L <sub>2</sub> | B <sub>1</sub> | D <sub>1</sub> | B <sub>2</sub> | D <sub>2</sub> |    |
|----|----------------|----------------|----------------|----------------|----------------|----------------|----|
| 0  | 7 8 11         | 12 15          | 16 19          | 20             | 31 32 35       | 36             | 47 |

### Condition Code

- ◆ 0 — sum is zero.  
 1 — sum is less than zero.  
 2 — sum is greater than zero.  
 3 — overflow.

### Interrupt Action

- ◆ Address error:  
 Addressing.  
 Protection.  
 Data error.  
 Decimal overflow.

### Notes

- ◆ 1. High-order zeros are supplied for *either* operand during instruction execution.  
 2. All signs and digits are checked for validity.  
 3. The operand specified by the second address is unaltered.  
 4. Processing is from right to left.  
 5. A zero result is always positive except when high-order digits are lost because of overflow. In overflow, a zero result has the sign of the correct result.

## **Subtract Decimal (SP)**

### **General Description**

◆ The operand specified by the second address ( $B_2/D_2$ ) is subtracted algebraically from the operand specified by the first address ( $B_1/D_1$ ). The result is stored in the field specified by the first address. The sign and the magnitude of the difference determine the condition code.

The operands can be variable in length up to 16 bytes and must be in packed format. If operands overlap, their rightmost byte location must coincide.

The subtraction of two operands can cause decimal overflow.

### **Format (SS)**

| FB | L <sub>1</sub> | L <sub>2</sub> | B <sub>1</sub> | D <sub>1</sub> | B <sub>2</sub> | D <sub>2</sub> |
|----|----------------|----------------|----------------|----------------|----------------|----------------|
| 0  | 7 8 11         | 12 15          | 16 19          | 20             | 31 32 35       | 36             |
|    |                |                |                |                |                | 47             |

### **Condition Code**

- ◆ 0 — difference is zero.
- 1 — difference is less than zero.
- 2 — difference is greater than zero.
- 3 — overflow.

### **Interrupt Action**

- ◆ Address error:
  - Addressing.
  - Protection.
  - Data error.
  - Decimal overflow.

### **Notes**

- ◆ 1. High-order zeros are supplied for *either* operand during instruction execution.
- 2. All signs and digits are checked for validity.
- 3. The operand specified by the second address is unaltered.
- 4. Processing is from right to left.
- 5. A zero difference is always positive except when high-order digits are lost because of overflow. In overflow, a zero result has the sign of the correct difference.

## Zero and Add (ZAP)

### General Description

◆ The operand specified by the second address ( $B_2/D_2$ ) is loaded into the location specified by the first address ( $B_1/D_1$ ). The operation is equivalent to an addition to zero and the result of the addition determines the condition code.

The operands may be variable in length up to 16 bytes and must be in packed format. High-order zeros are provided when necessary. Operands may overlap if their rightmost byte locations coincide, or if the rightmost byte of the first operand is to the right of the rightmost byte of the second operand.

A second operand that is longer than the first operand causes overflow.

### Format (SS)

|    |                |                |                |                |                |                |    |
|----|----------------|----------------|----------------|----------------|----------------|----------------|----|
| F8 | L <sub>1</sub> | L <sub>2</sub> | B <sub>1</sub> | D <sub>1</sub> | B <sub>2</sub> | D <sub>2</sub> |    |
| 0  | 7 8 11         | 12 15          | 16 19          | 20             | 31 32 35       | 36             | 47 |

### Condition Code

- ◆ 0 — result is zero.
- 1 — result is less than zero.
- 2 — result is greater than zero.
- 3 — overflow.

### Interrupt Action

- ◆ Address error:
  - Addressing.
  - Protection.
  - Data error.
  - Decimal overflow.

### Notes

- ◆ 1. Only the second operand is checked for valid sign and digit codes.
- 2. The second operand is unaltered.
- 3. Processing is from right to left.
- 4. A zero result is positive except when high-order digits are lost because of overflow. In overflow, a zero result has the sign of the second operand.

## **Compare Decimal (CP)**

### **General Description**

◆ The operand specified by the first address ( $B_1/D_1$ ) is algebraically compared with the operand specified by the second address ( $B_2/D_2$ ). The results of the comparison determine the condition code.

The operands may be variable in length up to 16 bytes and must be in packed format. The shorter operand is extended with high-order zeros when the operands are unequal in length. If operands overlap, their right-most byte location must be identical.

Overflow cannot occur as a result of this operation.

### **Format (SS)**

|    |                |                |                |                |                |                |
|----|----------------|----------------|----------------|----------------|----------------|----------------|
| F9 | L <sub>1</sub> | L <sub>2</sub> | B <sub>1</sub> | D <sub>1</sub> | B <sub>2</sub> | D <sub>2</sub> |
| 0  | 7 8 11         | 12 15          | 16 19 20       | 31             | 32 35 36       | 47             |

### **Condition Code**

- ◆ 0 — the fields are numerically equal.
- 1 — the first operand is algebraically less than the second operand.
- 2 — the first operand is algebraically greater than the second operand.

### **Interrupt Action**

- ◆ Address error:  
Addressing.  
Data error.

### **Notes**

- ◆ 1. All signs and digits are checked for validity.
- 2. Both operands are unaltered.
- 3. Comparison is from right to left.
- 4. A positive zero compares equally to a negative zero.

## Multiply Decimal (MP)

### General Description

◆ The operand specified by the first address (multiplicand) is multiplied by the operand specified by the second address (multiplier). The product is stored in the location of the first operand, right-justified.

The operands may be variable in length and must be in packed format. Operands can overlap if their rightmost byte locations coincide.

The second operand (multiplier) must be shorter than the first operand (multiplicand) and must not exceed eight bytes in length (15 digits plus sign). Otherwise, an address error (specification) occurs.

The multiplicand must have high-order zero digits equal to the number of digits in the multiplier, or a data error occurs. The maximum product size is 31 digits.

### Format (SS)

|    |                |                |                |                |                |                |    |
|----|----------------|----------------|----------------|----------------|----------------|----------------|----|
| FC | L <sub>1</sub> | L <sub>2</sub> | B <sub>1</sub> | D <sub>1</sub> | B <sub>2</sub> | D <sub>2</sub> |    |
| 0  | 7 8 11         | 12 15          | 16 19          | 20             | 31 32 35       | 36             | 47 |

### Condition Code

◆ Unchanged.

### Interrupt Action

◆ Address error:  
Addressing.  
Protection.  
Specification.  
Data error.

### Notes

- ◆ 1. All signs and digits are checked for validity.
- 2. The second operand is unaltered unless operands overlap.
- 3. Overflow cannot occur.
- 4. The sign of the product is determined by the rules of algebra, even if one, or both, operands are zero; that is, minus zero is a possible result.



## Divide Decimal (DP)

### General Description

◆ The operand specified by the first address (the dividend) is divided by the operand specified by the second address (the divisor) and the result (quotient plus remainder) replaces the first operand. The quotient is placed leftmost in the first operand field. The remainder, which has a size equal to the divisor size, is placed rightmost in the first operand field.

The operands may be variable in length and must be in packed format. Overlapping is allowed if the rightmost byte locations are identical. The second operand (the divisor) must be shorter than the first operand (the dividend) and must not exceed eight bytes in length (15 digits plus sign). If either rule is not observed, an address error (specification) occurs.

The dividend must have at least one high-order zero. Otherwise, a decimal divide error occurs.

Together, the quotient and remainder occupy the entire dividend field after division. Therefore, the address of the quotient field is the address of the dividend field and its size in bytes is  $L_1 - L_2 - 1$ . The quotient and remainder are signed integers which are right-aligned in the first operand.

No overflow can occur. A quotient that is larger than the number of digits allowed causes a decimal divide error.

### Format (SS)

|    |        |       |          |       |          |       |
|----|--------|-------|----------|-------|----------|-------|
| FD | $L_1$  | $L_2$ | $B_1$    | $D_1$ | $B_2$    | $D_2$ |
| 0  | 7 8 11 | 12 15 | 16 19 20 | 31    | 32 35 36 | 47    |

### Condition Code

◆ Unchanged.

### Interrupt Action

◆ Address error:  
Addressing.  
Protection.  
Specification.  
Data error.  
Decimal divide error.

### Notes

- ◆ 1. All signs and digits are checked for validity.
- 2. The second operand is unaltered.
- 3. The sign of the quotient is determined by the rules of algebra from dividend and divisor signs. The sign of the remainder has the same value as the dividend sign.
- 4. The first address plus  $(L_1 - L_2)$  specifies the address of the remainder. The length of the remainder is specified by  $L_2$ .

**Pack  
(PACK)**

**General Description**

◆ The operand specified by the second address ( $B_2/D_2$ ) is converted from zoned format to packed format and the result is placed in the location specified by the first address ( $B_1/D_1$ ).

The operand specified by the second address must be in zoned format. The sign is obtained from the zone portion of the rightmost byte of the second operand and is placed in the rightmost four bits of the first operand (result field). All other zones are ignored. The four-bit numeric portions (stripping the four-bit zone) of each byte are then placed adjacent to the sign, and to each other, to fill the result field.

The result is extended with high-order zeros if the second operand field is shorter than the first. If the first operand field is not large enough to contain all the significant digits from the second operand field, the remaining digits are ignored. The operands may overlap.

**Format  
(SS)**

| F2 |   | L <sub>1</sub> |    | L <sub>2</sub> |    | B <sub>1</sub> |    | D <sub>1</sub> |  |  |    | B <sub>2</sub> |    | D <sub>2</sub> |  |    |  |
|----|---|----------------|----|----------------|----|----------------|----|----------------|--|--|----|----------------|----|----------------|--|----|--|
| 0  | 7 | 8              | 11 | 12             | 15 | 16             | 19 | 20             |  |  | 31 | 32             | 35 | 36             |  | 47 |  |

**Condition Code**

◆ Unchanged.

**Interrupt Action**

◆ Address error:  
Addressing.  
Protection.

**Notes**

- ◆ 1. Signs and digits are not checked for validity.
- 2. The second operand is not changed except when the operands overlap.
- 3. Processing is from right to left, one byte at a time.

## Unpack (UNPK)

### General Description

◆ The operand specified by the second address ( $B_2/D_2$ ) is converted from packed format to zoned format and the result is placed in the location specified by the first address ( $B_1/D_1$ ).

Each of the eight-bit bytes of the packed, second-operand field represents two four-bit digits. Each of the four-bit digits is stored in a byte of the first operand field in the low-order four-bit positions. If the Decimal Code is EBCDIC, a zone code of 1111 is inserted into the high-order four bits of each byte. If the Decimal Code is ASCII, a zone code of 0101 is inserted. These zones are inserted in all but the zone portion of the right-most byte, which receives the sign of the packed operand.

If the first operand is not large enough to receive the significant digits of the second operand, the remaining digits are ignored. The second-operand field is extended with zero digits before unpacking.

### Format (SS)

|    |                |                |                |                |                |                |    |    |    |    |    |    |    |
|----|----------------|----------------|----------------|----------------|----------------|----------------|----|----|----|----|----|----|----|
| F3 | L <sub>1</sub> | L <sub>2</sub> | B <sub>1</sub> | D <sub>1</sub> | B <sub>2</sub> | D <sub>2</sub> |    |    |    |    |    |    |    |
| 0  | 7              | 8              | 11             | 12             | 15             | 16             | 19 | 20 | 31 | 32 | 35 | 36 | 47 |

### Condition Code

◆ Unchanged.

### Interrupt Action

◆ Address error:  
Addressing.  
Protection.

### Notes

- ◆ 1. Signs and digits are not checked for validity.
- 2. The second operand is not altered, except when operands overlap.
- 3. Processing is from right to left.

## MOVE with OFFSET (MVO)

### General Description

◆ The operand specified by the second address ( $B_2/D_2$ ) is offset 4 bits to the left (a 1-digit left shift) and is placed to the left of, and adjacent to, the low-order four bits of the operand specified by the first address ( $B_1/D_1$ ).

If the first operand is not large enough to receive all bytes of the second operand, the remaining bytes are ignored. If the second operand is shorter than the first operand, the second operand is extended with high-order zeros. The first and second operands may overlap.

### Format (SS)

|    |                |                |                |                |                |                |    |    |    |    |    |    |    |
|----|----------------|----------------|----------------|----------------|----------------|----------------|----|----|----|----|----|----|----|
| F1 | L <sub>1</sub> | L <sub>2</sub> | B <sub>1</sub> | D <sub>1</sub> | B <sub>2</sub> | D <sub>2</sub> |    |    |    |    |    |    |    |
| 0  | 7              | 8              | 11             | 12             | 15             | 16             | 19 | 20 | 31 | 32 | 35 | 36 | 47 |

### Condition Code

◆ Unchanged.

### Interrupt Action

◆ Address error:  
Addressing.  
Protection.

### Notes

- ◆ 1. Signs and digits are not checked for validity.
- 2. The second operand is not changed except when operands overlap.
- 3. Processing is from right to left.
- 4. The initial low-order 4-bit digit of the operand specified by the first address is left unaltered.

## LOGICAL INSTRUCTIONS

### INTRODUCTION

◆ Logical instructions are used to manipulate data. The operands are usually treated as eight-bit bytes. Some logical operations require a single eight-bit byte specified as an operand; others may have variable-length operands composed of many eight-bit bytes. Some instructions operate on the zone portion only, or on the digit portion only, of the bytes of a variable-length operand. Some instructions have an operand that is part of the immediate instruction being executed. Finally, there is a group of instructions that provide for bit shifting.

Operands are in either main memory or general registers. Processing of data in main memory is from left-to-right starting at any byte location. Processing in general registers usually involves the entire contents of a general register, or in some cases, two general registers.

The Edit instruction is the only instruction which requires that the data be in packed decimal data. The Edit instruction converts packed decimal data into alphanumeric characters with editing under the control of a mask pattern.

The logical instruction set includes moving, comparing, bit testing, translating, editing, shifting, and bit connecting.

The condition code is set by all instructions except the moving, translating, and shifting instructions.

### DATA FORMAT

◆ Data in general registers usually involves the entire 32 bits. There is no distinction made between sign and numeric bits. In some operations, only the least significant eight bits of the general register are involved, and in another case, the least significant 24 bits are involved. In addition, there are some shift operations in which an even/odd numbered pair of general registers is involved.

The storage data in memory-to-register operations resides in either a 32-bit word or an eight-bit byte. A word must be oriented on word boundaries (i.e., the address of the 32-bit word must have the two low-order bits zero).

The storage data in memory-to-memory operations have a variable length format and can have a field size of up to 256 bytes starting at any byte location. Processing is from left to right.

Instructions that specify an operand that is part of the immediate instruction being executed are restricted to a field size of one eight-bit byte.

The Translate and Test and the Edit and Mark instructions imply the use of General Register 1\*. An address of 24 bits may be placed in this register during the execution of these instructions. The Translate and Test instruction also implies the use of General Register 2 where an insertion of an eight-bit function byte may be placed during the execution of the instruction.

Overlapping of fields in memory-to-memory operations may or may not affect the operands of the various instructions. The execution of some

\* When these instructions are executed in  $P_3$ , General Registers 13 and 14 are used; in  $P_4$ , General Registers 9 and 10 are used.

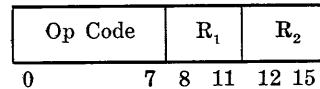
**DATA FORMAT**  
(Cont'd)

logical instructions does not change the operands. Other instructions, such as Move, Edit, and Translate, replace one operand with new data, and this data is handled one eight-bit byte at a time. This procedure enables the user to determine the effect overlapping fields have on the execution of the instruction. Unpredictable results can occur while overlapping fields are being edited. Overlapping fields are valid for all other operations.

**INSTRUCTION  
FORMATS**

◆ The logical instructions use the following five instruction formats (RR, RX, RS, SI, SS) :

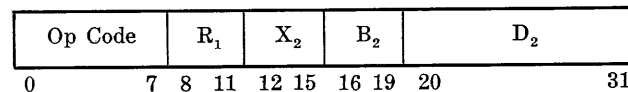
**RR Format**



*Description*

◆ In the RR format, the contents of the general register specified by R<sub>1</sub> are called the first operand. The contents of the general register specified by R<sub>2</sub> are called the second operand.

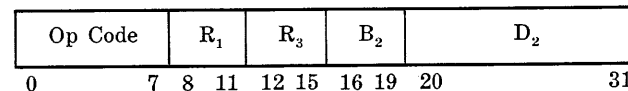
**RX Format**



*Description*

◆ In the RX format, the contents of the general register specified by R<sub>1</sub> are called the first operand. To obtain the address of the second operand, the contents of the general registers specified by X<sub>2</sub> and B<sub>2</sub> are added to the contents of the D<sub>2</sub> field.

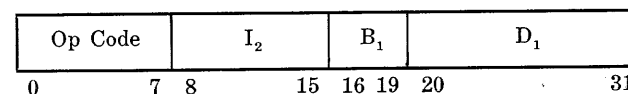
**RS Format**



*Description*

◆ In the RS format, which is only used for shift instructions in this instruction set, the contents of the general register specified by R<sub>1</sub> are called the first operand. There is no actual storage address formed by adding the contents of the general register specified by B<sub>2</sub> and the contents of D<sub>2</sub>. Instead, this sum specifies the number of bits to be shifted by the shift operations. The R<sub>3</sub> field is ignored in the shift operation.

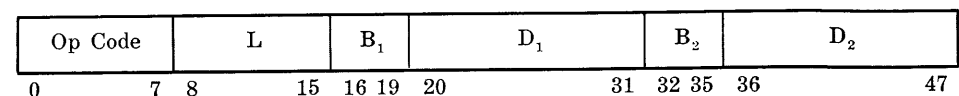
**SI Format**



*Description*

◆ In the SI format, the contents of the general register specified by B<sub>1</sub> are added to the contents of the D<sub>1</sub> field to obtain the address of the first operand. The second operand is the immediate eight-bit byte in the I<sub>2</sub> field of the instruction.

**SS Format**



*Description*

◆ In the SS format, the contents of the general register specified by B<sub>1</sub> are added to the contents of the D<sub>1</sub> field to obtain the address of the leftmost byte of the first operand. The L field specifies the number of additional bytes in the operand that are to the right of the first operand. To obtain

**SS Format  
(Cont'd)**

the second operand address, the contents of the general register specified by B<sub>2</sub> are added to the contents of the D<sub>2</sub> field. The length of the second operand is the same as the length of the first.

The use of a zero in the X<sub>2</sub>, B<sub>1</sub>, or B<sub>2</sub> field of any instruction indicates that no register is to be used as a component of the instruction. Instructions may use a general register for both address modification and operand location. Addresses are always modified before an instruction is executed.

**CONDITION CODE  
UTILIZATION**

◆ The condition code is set as a result of using most of the logical instructions. The condition code setting has a different meaning when using different instructions and can be tested by subsequent branch on condition instructions for decision making. Altogether, there are five types of result meanings. The instructions which cause the condition code to be set and the meaning of the setting are as follows:

| Instruction        | Condition Code Setting |            |          |     |
|--------------------|------------------------|------------|----------|-----|
|                    | 0                      | 1          | 2        | 3   |
| AND                | Zero                   | Not Zero   | —        | —   |
| Compare Logical    | Equal                  | Low        | High     | —   |
| Edit               | Zero                   | < Zero     | > Zero   | —   |
| Edit and Mark      | Zero                   | < Zero     | > Zero   | —   |
| Exclusive OR       | Zero                   | Not Zero   | —        | —   |
| OR                 | Zero                   | Not Zero   | —        | —   |
| Test Under Mask    | Zero                   | Mixed      | —        | One |
| Translate and Test | Zero                   | Incomplete | Complete | —   |

**INTERRUPT ACTION**

◆ The following interrupt conditions can occur as a result of logical instructions:

**Address Error**

*Addressing*

◆ An address error interrupt occurs when an address specifies a location outside the available memory. At the point of error the operation is terminated. The result data and condition code, if affected, are unpredictable.

*Specification*

◆ An address error interrupt occurs when a full-word operand is not located on a word boundary in a storage-to-register operation, or when an odd register is specified as the first register in an instruction which performs an operation on an even/odd pair of general registers. The operation is suppressed.

*Protection*

◆ An address error interrupt occurs when the storage key and the protection key of the result location do not match. The operation is suppressed and the condition code, registers, and main memory are unaltered. The variable-length memory-to-memory instructions are the only exception, in which case the operation is terminated and the result data and the condition code setting are unpredictable. (This interrupt can only occur if the memory protect feature is installed.)

**Data Error**

◆ A data error occurs if a digit code of the second operand in the Edit instruction or Edit and Mark instruction is invalid. The operation is terminated, and the result data and condition code setting are unpredictable.

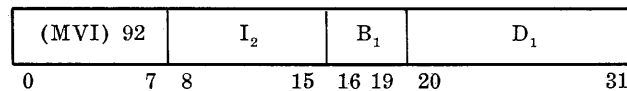
**Move  
(MVI) (MVC)**

**General Description**

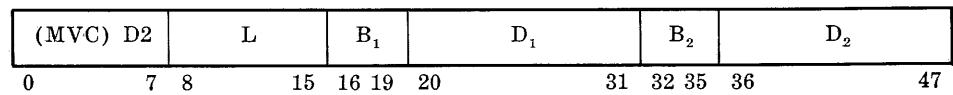
◆ To process the SS format Move instruction, the source field specified by the second address ( $B_2/D_2$ ) is moved into the destination field specified by the first address ( $B_1/D_1$ ). This format is used for a main memory-to-main memory move.

For the SI format Move instruction, the immediate byte in the  $I_2$  field of the instruction being executed is stored in the main memory location specified by the first address ( $B_1/D_1$ ).

**Format  
(SI)**



**(SS)**



**Condition Code**

◆ Unchanged.

**Interrupt Action**

◆ Address error:  
    Addressing.  
    Protection.

**Notes**

- ◆ 1. The bytes being moved are not inspected or changed.
- 2. Processing is from left to right and overlapping of fields is permitted.
- 3. The second operand is not altered, unless operands overlap in the SS format.
- 4. It is possible to propagate one byte through an entire field by having the first operand address specify one location to the right of the second operand address.



## Move Numerics (MVN)

### General Description

- ◆ The low-order four bits of each byte in the source operand specified by the second address ( $B_2/D_2$ ) are placed into the low-order four bits of the corresponding byte of the destination operand specified by the first address ( $B_1/D_1$ ).

### Format (SS)

|    |   |   |    |                |    |                |    |                |    |                |    |
|----|---|---|----|----------------|----|----------------|----|----------------|----|----------------|----|
| D1 |   | L |    | B <sub>1</sub> |    | D <sub>1</sub> |    | B <sub>2</sub> |    | D <sub>2</sub> |    |
| 0  | 7 | 8 | 15 | 16             | 19 | 20             | 31 | 32             | 35 | 36             | 47 |

### Condition Code

- ◆ Unchanged.

### Interrupt Action

- ◆ Address error:  
Addressing.  
Protection.

### Notes

- ◆ 1. The numerics are not changed or checked for validity.
- 2. The operand specified by the second address is not altered, unless operands overlap.
- 3. Processing is from left to right.
- 4. The high-order four bits of the source and destination operand bytes are not altered.
- 5. The operand fields may overlap in any way and may be variable in length.

## Move Zones (MVZ)

### General Description

- ◆ The high-order four bits of each byte in the source operand specified by the second address ( $B_2/D_2$ ) are placed into the high-order four bits of the corresponding byte of the destination operand specified by the first address ( $B_1/D_1$ ).

### Format (SS)

|    |     |                |                |                |                |
|----|-----|----------------|----------------|----------------|----------------|
| D3 | L   | B <sub>1</sub> | D <sub>1</sub> | B <sub>2</sub> | D <sub>2</sub> |
| 0  | 7 8 | 15 16 19       | 20             | 31 32 35       | 36 47          |

### Condition Code

- ◆ Unchanged.

### Interrupt Action

- ◆ Address error:  
Addressing.  
Protection.

### Notes

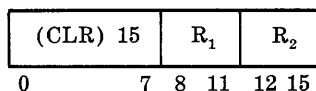
- ◆ 1. The zones are not changed or checked for validity.
- 2. The operand specified by the second address is not altered, unless operands overlap.
- 3. Processing is from left to right.
- 4. The low-order four bits of the source and destination operand bytes are not altered.
- 5. The operand fields may overlap in any way and may be variable in length.

## Compare Logical (CLR) (CL) (CLI) (CLC)

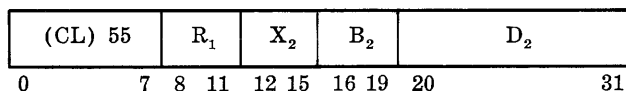
### General Description

◆ The operand specified by the first address is logically compared with the operand specified by the second address (RR format:  $R_1$  to  $R_2$ ; RX format:  $R_1$  to  $X_2/B_2/D_2$ ; SI format:  $B_1/D_1$  to  $I_2$ ; SS format:  $B_1/D_1$  to  $B_2/D_2$ ). The result of the comparison determines the condition code. These instructions process all bits as part of an unsigned binary quantity. All codes are valid and the instruction is terminated on inequality or when the operand bytes have been exhausted.

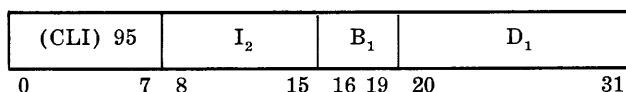
### Format (RR)



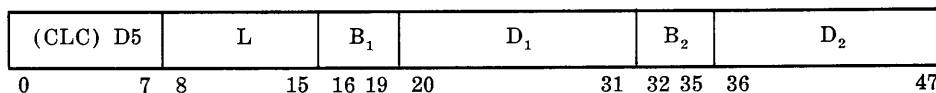
### (RX)



### (SI)



### (SS)



### Condition Code

- ◆ 0 — the operands are equal.  
 1 — the first operand is less than the second operand.  
 2 — the first operand is greater than the second operand.  
 3 — not used.

### Interrupt Action

- ◆ Address error:  
     Addressing (RX, SI, SS only).  
     Specification (RX only).

### Notes

- ◆ 1. Both operands are unaltered.  
 2. In the SI format, the immediate byte in the  $I_2$  field of the instruction being executed is the second operand.  
 3. Processing is from left to right and can extend to field lengths of 256 bytes.  
 4. The operation can be used for alphanumeric comparisons.

**AND**  
**(NR) (N) (NI) (NC)**

**General Description**

◆ These instructions perform a logical “AND” operation on two operands bit-by-bit according to the following rules:

**Rules of Logical “AND” Operation**

| If Bit in<br>First Operand is | And Bit in<br>Second Operand is | Then Bit in<br>Result is |
|-------------------------------|---------------------------------|--------------------------|
| 0                             | 0                               | 0                        |
| 0                             | 1                               | 0                        |
| 1                             | 0                               | 0                        |
| 1                             | 1                               | 1                        |

The logical product of the operation is placed in the location specified by the first address ( $R_1$  or  $B_1/D_1$ ) and determines the condition code.

**Format**  
**(RR)**

|         |              |       |
|---------|--------------|-------|
| (NR) 14 | $R_1$        | $R_2$ |
| 0       | 7 8 11 12 15 |       |

**(RX)**

|        |              |          |       |       |
|--------|--------------|----------|-------|-------|
| (N) 54 | $R_1$        | $X_2$    | $B_2$ | $D_2$ |
| 0      | 7 8 11 12 15 | 16 19 20 |       | 31    |

**(SI)**

|         |        |          |       |
|---------|--------|----------|-------|
| (NI) 94 | $I_2$  | $B_1$    | $D_1$ |
| 0       | 7 8 15 | 16 19 20 | 31    |

**(SS)**

|         |        |          |       |          |       |
|---------|--------|----------|-------|----------|-------|
| (NC) D4 | $L$    | $B_1$    | $D_1$ | $B_2$    | $D_2$ |
| 0       | 7 8 15 | 16 19 20 | 31    | 32 35 36 | 47    |

**Condition Code**

- ◆ 0 — result is zero.  
1 — result not zero.  
2 — not used.  
3 — not used.

**Interrupt Action**

- ◆ Address error:  
Addressing (RX, SI, SS only).  
Protection (SI, SS only).  
Specification (RX only).

**Notes**

- ◆ 1. The second operand is unaltered, unless operands overlap in the SS format.  
2. In the SI format, the immediate byte in the  $I_2$  field of the instruction being executed is the second operand.  
3. Processing is from left to right.  
4. All operands and results are valid.  
5. The “AND” instruction is also used to set a bit to zero.

**OR**  
**(OR) (O) (OI) (OC)**

**General Description**

◆ This instruction performs a logical “OR” operation on two operands bit-by-bit according to the following rules:

**Rules for Logical “OR” Operation**

| If Bit in<br>First Operand is | And Bit in<br>Second Operand is | Then Bit in<br>Result is |
|-------------------------------|---------------------------------|--------------------------|
| 0                             | 0                               | 0                        |
| 0                             | 1                               | 1                        |
| 1                             | 0                               | 1                        |
| 1                             | 1                               | 1                        |

The logical result of the operation is placed in the location specified by the first address ( $R_1$  or  $B_1/D_1$ ) and determines the condition code.

**Format**  
**(RR)**

|         |              |       |
|---------|--------------|-------|
| (OR) 16 | $R_1$        | $R_2$ |
| 0       | 7 8 11 12 15 |       |

**(RX)**

|        |              |          |       |       |
|--------|--------------|----------|-------|-------|
| (O) 56 | $R_1$        | $X_2$    | $B_2$ | $D_2$ |
| 0      | 7 8 11 12 15 | 16 19 20 |       | 31    |

**(SI)**

|         |        |          |       |
|---------|--------|----------|-------|
| (OI) 96 | $I_2$  | $B_1$    | $D_1$ |
| 0       | 7 8 15 | 16 19 20 | 31    |

**(SS)**

|         |        |          |             |       |       |
|---------|--------|----------|-------------|-------|-------|
| (OC) D6 | $L$    | $B_1$    | $D_1$       | $B_2$ | $D_2$ |
| 0       | 7 8 15 | 16 19 20 | 31 32 35 36 |       | 47    |

**Condition Code**

- ◆ 0 — result is zero.  
1 — result is not zero.  
2 — not used.  
3 — not used.

**Interrupt Action**

- ◆ Address error:  
Addressing (RX, SI, SS only).  
Protection (SI, SS only).  
Specification (RX only).

**Notes**

- ◆ 1. The second operand is unaltered, unless operands overlap in the SS format.  
2. In the SI format, the immediate byte in the  $I_2$  field of the instruction being executed is the second operand.  
3. Processing is from left to right.  
4. All operands and results are valid.  
5. The “OR” instruction is also used to set a bit to one.

# **Exclusive OR (XR) (X) (XI) (XC)**

## **General Description**

◆ These instructions perform an Exclusive “OR” operation on two operands bit-by-bit according to the following rules:

### **Rules for Exclusive “OR” Operation**

| If Bit of<br>First Operand is | And Bit of<br>Second Operand is | Then Bit in<br>Result is |
|-------------------------------|---------------------------------|--------------------------|
| 0                             | 0                               | 0                        |
| 0                             | 1                               | 1                        |
| 1                             | 0                               | 1                        |
| 1                             | 1                               | 0                        |

The modulo-two sum (binary addition without carries) of the operation is placed in the location specified by the first address ( $R_1$  or  $B_1/D_1$ ) and determines the condition codes.

## **Format (RR)**

|         |        |       |
|---------|--------|-------|
| (XR) 17 | $R_1$  | $R_2$ |
| 0       | 7 8 11 | 12 15 |

## **(RX)**

|        |        |       |       |       |
|--------|--------|-------|-------|-------|
| (X) 57 | $R_1$  | $X_2$ | $B_2$ | $D_2$ |
| 0      | 7 8 11 | 12 15 | 16 19 | 20 31 |

## **(SI)**

|         |       |          |       |
|---------|-------|----------|-------|
| (XI) 97 | $I_2$ | $B_1$    | $D_1$ |
| 0       | 7 8   | 15 16 19 | 20 31 |

## **(SS)**

|         |     |          |       |       |       |
|---------|-----|----------|-------|-------|-------|
| (XC) D7 | L   | $B_1$    | $D_1$ | $B_2$ | $D_2$ |
| 0       | 7 8 | 15 16 19 | 20 31 | 32 35 | 36 47 |

## **Condition Code**

- ◆ 0 — result is zero.
- 1 — result is other than zero.
- 2 — not used.
- 3 — not used.

## **Interrupt Action**

- ◆ Address error:  
Addressing (RX, SI, SS only).  
Protection (SI, SS only).  
Specification (RX only).

## **Notes**

- ◆ 1. The second operand is unaltered, unless operands overlap in the SS format.
- 2. In the SI format, the immediate byte in the  $I_2$  field of the instruction being executed is the second operand.
- 3. Processing is from left to right.
- 4. All operands and results are valid.
- 5. These instructions may be used to complement a number (one's complement).

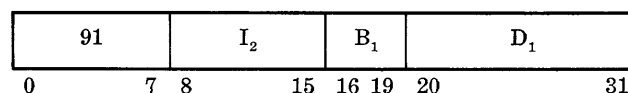
## **Test Under Mask (TM)**

### **General Description**

◆ The operand (byte) specified by the first address ( $B_1/D_1$ ) is tested against the immediate I field (byte) as a mask. The result determines the condition code. The I field is used as an eight-bit mask and is made to correspond one-for-one with the bits of the byte in main memory that is specified by the first address.

A bit in the byte being examined is said to be selected when the corresponding mask bit is a one. When the mask bit is a zero, the bit in main memory is ignored.

### **Format (SI)**



### **Condition Code**

- ◆ 0 — selected bits all zero or mask is all zero.
- 1 — selected bits mixed zero and one.
- 2 — not used.
- 3 — selected bits all one's.

### **Interrupt Action**

- ◆ Address error:  
Addressing.

### **Note**

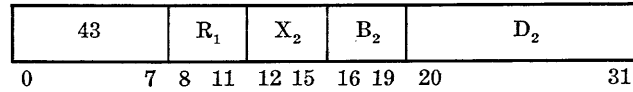
- ◆ The operands are unaltered.

## Insert Character (IC)

### General Description

- ◆ The eight-bit byte specified by the second address ( $X_2/B_2/D_2$ ) is loaded into the rightmost byte of the general register specified by the first address ( $R_1$ ). The remaining bits of the register are unaltered.

### Format (RX)



### Condition Code

- ◆ Unchanged.

### Interrupt Action

- ◆ Address error:  
Addressing.

### Note

- ◆ The operand specified by the second address is not altered or inspected.

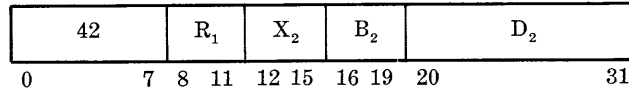


## Store Character (STC)

### General Description

- ◆ The rightmost eight-bit byte of the general register specified by the first address ( $R_2$ ) is stored into the main memory location specified by the second address ( $X_2/B_2/D_2$ ).

### Format (RX)



### Condition Code

- ◆ Unchanged.

### Interrupt Action

- ◆ Address error:  
    Addressing.  
    Protection.

### Note

- ◆ The operand specified by the first address is not altered or inspected.

## Load Address (LA)

### General Description

◆ The final main memory address specified by the second operand ( $X_2/B_2/D_2$ ) is loaded into the rightmost 24 bits of the general register specified by the first address ( $R_1$ ). The leftmost eight bits of the register are set to zeros.

The contents of the registers specified by the  $X_2$  and  $B_2$  fields are added to the contents of the  $D_2$  field of the instruction to obtain an address. This is the address that is loaded into the register specified by the first address. Any carry beyond the rightmost 24 bits is ignored.

### Format (RX)

|    |        |       |          |       |
|----|--------|-------|----------|-------|
| 41 | $R_1$  | $X_2$ | $B_2$    | $D_2$ |
| 0  | 7 8 11 | 12 15 | 16 19 20 | 31    |

### Condition Code

◆ Unchanged.

### Interrupt Action

◆ None.

### Notes

- ◆ 1. All specified address arithmetic is computed before loading.
- 2.  $R_1$ ,  $X_2$  and  $B_2$  may specify the same register; however  $R_1$  only may specify register 0.
- 3. This instruction can be used to increment the low-order 24 bits of a general register (other than 0) by the contents of the  $D_2$  field. The register to be incremented is specified by  $R_1$ , and either  $X_2$  (with  $B_2$  set to zero) or  $B_2$  (with  $X_2$  set to zero). Since  $R_1$  and  $X_2$  or  $B_2$  must specify the same register, register zero cannot be incremented (a zero in the  $B_2$  or  $X_2$  field indicates that the corresponding address component is absent).
- 4. Main memory is not accessed by this instruction.

**Translate  
(TR)**

**General Description**

◆ The variable length operand specified by the first address ( $B_1/D_1$ ) is translated, byte-for-byte, according to the byte translation table specified by the second address ( $B_2/D_2$ ). The result replaces the bytes in the field specified by the first address.

The bytes of the first operand are termed the argument bytes. Bytes of the first operand are selected for translation from left-to-right, one byte at a time. Each argument byte is added to the second operand address, which is the starting location of a translation table. This sum, in turn, addresses a byte location within the table containing a function byte. The function byte at this location replaces the original argument byte of the first operand.

The operation terminates when the first operand bytes have been exhausted.

**Format  
(SS)**

|    |     |                |                |                |                |
|----|-----|----------------|----------------|----------------|----------------|
| DC | L   | B <sub>1</sub> | D <sub>1</sub> | B <sub>2</sub> | D <sub>2</sub> |
| 0  | 7 8 | 15 16 19       | 20             | 31 32 35 36    | 47             |

**Condition Code**

◆ Unchanged.

**Interrupt Action**

◆ Address error:  
Addressing.  
Protection.

**Notes**

- ◆ 1. The translation table is unaltered unless overlap occurs.
- 2. The field to be translated and the translation table are addressed by their leftmost byte.
- 3. The length of a table, in general, must be 256 bytes, unless the domain of argument bytes is limited to a specific subset by the program and data.
- 4. The L field specifies the length of the first operand minus one (binary 00000001 = 2 bytes).

## Translate and Test (TRT)

### General Description

◆ The variable length operand, which is specified by the first address ( $B_1/D_1$ ), is used as the argument (byte-by-byte) to reference a list (functions) specified by the second address ( $B_2/D_2$ ). The functions referenced are inspected for zero or non-zero. If a non-zero is encountered, the address of the argument byte is loaded into General Register 1 (General Register 13 in  $P_3$ ; General Register 9 in  $P_4$ ) and the function byte is loaded into the rightmost end of General Register 2 (General Register 14 in  $P_3$ ; General Register 10 in  $P_4$ ). Whenever zeros are encountered in the function list, the operation proceeds to the next byte. The first operand is unaltered.

The bytes of the first operand are termed the argument bytes. Processing of the first operand is from left-to-right, one byte at a time. Each argument byte is added to the second operand, which is the starting location of the translate table. This sum, in turn, addresses a byte location within the table, which is termed a function byte. Then, the function byte retrieved from the table is inspected for all zeros.

If the function byte is all zeros, the operation proceeds to the next argument byte and continues processing. If the function byte is not all zeros, the instruction inserts the address of the argument byte in the low-order 24 bits of General Register 1 (13 or 9) and inserts the retrieved non-zero function byte in the low-order eight-bits of General Register 2 (14 or 10). The high-order eight bits of General Register 1 (13 or 9) and high-order 24 bits of General Register 2 (14 or 10) are unaltered.

The operation terminates when a (non-zero) function byte is accessed or when the first operand field is exhausted.

### Format (SS)

| DD | L   | $B_1$    | $D_1$ | $B_2$       | $D_2$ |
|----|-----|----------|-------|-------------|-------|
| 0  | 7 8 | 15 16 19 | 20    | 31 32 35 36 | 47    |

### Condition Code

- ◆ 0 — accessed function bytes all zeros.
- 1 — a non-zero function byte is encountered before the first operand field is exhausted.
- 2 — the last function byte is non-zero.
- 3 — not used.

### Interrupt Action

- ◆ Address error:  
Addressing.

### Notes

- ◆ 1. The variable length field specified by the first address is unaltered.
- 2. If non-zero functions do not occur, General Registers 1 (13 or 9) and 2 (14 or 10) are unaltered.
- 3. The first operand and the translation table are addressed by their leftmost bytes.
- 4. The length of the table, in general, must be 256 bytes, unless the domain of argument bytes is limited to a specific subset by the program and data.
- 5. The L field specifies the length of the first operand minus one.
- 6. This instruction is useful for scanning input streams and locating delimiters for variable length records and fields.
- 7. In processor states  $P_1$  and  $P_2$ , General Registers 1 and 2 are used. In processor state  $P_3$ , General Registers 13 and 14 are used. In processor state  $P_4$ , General Registers 9 and 10 are used.

**Edit  
(ED)**

**General Description**

◆ The variable length source field specified by the second address ( $B_2/D_2$ ) is changed from packed format to zoned format with the results edited under the control of a mask pattern. The result of the operation replaces the mask pattern specified by the first address ( $B_1/D_1$ ) and determines the condition code.

The L field applies to the mask pattern (first address field). The source digits are processed left-to-right, one byte at a time. The leftmost four bits of each byte are examined first and the rightmost four bits of each byte are held available for the next mask character that calls for digit examination. Immediately after the leftmost four bits have been examined, the rightmost four bits are checked for a sign code. When one of the sign codes is encountered, these bits are no longer treated as a digit. A new character is fetched from the mask pattern for the next digit to be examined.

**Format  
(SS)**

| DE | L   | B <sub>1</sub> | D <sub>1</sub> | B <sub>2</sub> | D <sub>2</sub> |
|----|-----|----------------|----------------|----------------|----------------|
| 0  | 7 8 | 15 16 19       | 20             | 31 32 35       | 36             |
|    |     |                |                |                | 47             |

**Editing Rules**

◆ Editing includes sign control, punctuation control, zero suppression or check protection, and also facilitates blanking of all-zero fields. In addition, multiple fields of digits can be edited in one operation, and numeric data can be combined with alphabetic and special characters.

Editing rules depend on the control code, significance, and the source digit, and are given as follows:

**Editing Rules**

| Control Codes       | Hexadecimal Code | Decimal Code | Function   |
|---------------------|------------------|--------------|--|
| Filler              | Any              | Any          | *Replaces leading zeros.   |
| Start Significance  | 21               | 33           | Stops replacement of leading zeros. Also acts as a digit select code.  |
| Digit Select        | 20               | 32           | Specifies digit position in data (replaced by filler code if appears after a negative sign has been sensed). |
| Field Separator     | 22               | 34           | Indicates editing of a new field is to begin (replaced by filler code).                                      |
| Insertion Character | Any              | Any          | Inserted in the result.  |

\* The most common filler characters are the blank and the asterisk.

1. Source digits are examined only when a digit select code  $(20)_{16}$  or a start significance code  $(21)_{16}$  is encountered in the mask pattern.
2. Significance is established either:
  - a. upon encountering a non-zero digit in the source field.
  - b. after encountering a start significance code  $(21)_{16}$  within the mask pattern.

**Editing Rules  
(Cont'd)**

3. If significance has *not* been established, every control code or insertion character encountered in the mask pattern (including the start significance code) is replaced by the filler character.
4. If significance *has* been established, every digit select code (20)<sub>16</sub> or start significance code (21)<sub>16</sub> encountered in the mask pattern is replaced by a digit from the source field, which is expanded by attaching a zone.
5. If significance *has* been established, every insertion character (other than the digit select, start significance, or field separator codes) encountered within the mask pattern is left in place without alteration.
6. Significance is disestablished by:
  - a. encountering a field separator code (22)<sub>16</sub> in the mask pattern.
  - b. encountering a positive (plus) sign within the rightmost four bits of a source field byte.
7. A negative (minus) sign within the rightmost four bits of a source byte does *not* disestablish significance. Additional digit select codes encountered in the mask pattern are replaced by filler characters, but insertion characters are left in place without alteration.
8. Field separator codes (22)<sub>16</sub> are always replaced by the filler character.

*Note:* The filler character is obtained from the mask pattern as part of the editing operation. The first character (leftmost byte) of the mask pattern is used as a filler character and is left unchanged in the result, except:

- a. when it is a digit select code.
- b. when it is a start significance code.

In these codes, a source digit is examined and, when non-zero, inserted in the result field.

To facilitate blanking out all-zero result fields, or triggering negative field special processing, the condition code is used to indicate the sign and zero status of the last field edited. All digits examined are tested for zero, and the presence, or absence, of an all-zero source field is indicated in the condition code at the termination of the editing operation. Sign significance is also indicated by the condition code.

**Condition Code**

- ◆ 0 — indicates a zero source field regardless of whether or not significance is established.
- 1 — indicates non-zero result field with significance established to indicate less than zero.
- 2 — indicates non-zero result field with no significance established to indicate greater than zero.
- 3 — not used.

*Note:* The condition code setting reflects only the field following the last (rightmost) field separator code of the mask pattern for multiple-field-editing operations.

**Interrupt Action**

- ◆ Address error:  
Addressing.  
Protection.  
Data error.

**Notes**

- ◆ 1. The leftmost four-bits of any source field byte must be a valid digit, otherwise a data error interrupt occurs.
- 2. The rightmost four-bits of any source field byte can be either a digit or a sign.
- 3. Multiple field editing is possible by using the field separator code within the mask pattern.
- 4. The zones of the expanded source digits can be either EBCDIC or ASCII, as specified by the mode code. When the mode code specifies EBCDIC, zone code 1111 is generated. When the mode code specifies ASCII, the zone code 0101 is generated.
- 5. The rightmost four bits of any source field byte can be a digit or sign as follows:

| Codes                  | Definition |
|------------------------|------------|
| 0000 → 1001            | Digits     |
| 1010, 1100, 1110, 1111 | Plus sign  |
| 1011, 1101             | Minus sign |

- 6. Overlapping of fields yields unpredictable results.

## Edit and Mark (EDMK)

### General Description

◆ The variable length source field specified by the second address ( $B_2/D_2$ ) is changed from packed format to zoned format and the results are edited under control of a mask pattern. The result of the operation replaces the mask pattern specified by the first address ( $B_1/D_1$ ) and determines the condition code. In addition, the address of each first significant result digit is stored in General Register 1 (General Register 13 in  $P_3$ ; General Register 9 in  $P_4$ ).

The operation of this instruction is identical to the Edit instruction except for the additional function of inserting a byte address in General Register 1 (13 or 9). The destination address of the digit that establishes significance within the source field being edited is loaded into the rightmost 24 bits of General Register 1 (13 or 9). The leftmost eight bits are unaltered. The address is *not* loaded when significance is forced by recognition of the start significance code in the mask pattern.

The Edit and Mark instruction facilitates the insertion of floating currency symbols, sign indicators, relational operators, and other editing symbols (\$, +, -, <, >, etc.). The address loaded into the register is one byte to the right of the address where such a symbol would be inserted. (The Branch on Count instruction, with zero in the  $R_2$  field, can be used to reduce the loaded address by one.)

Because the address is *not* loaded when significance is forced by the start significance code, the address of the byte immediately to the right of the start significance code in the mask pattern field should be loaded in General Register 1 (13 or 9) before an Edit and Mark instruction is executed.

### Format (SS)

| DF | L   | $B_1$    | $D_1$ | $B_2$       | $D_2$ |
|----|-----|----------|-------|-------------|-------|
| 0  | 7 8 | 15 16 19 | 20    | 31 32 35 36 | 47    |

### Condition Code

- ◆ 0 — indicates a zero source field whether or not significance is established.
- 1 — indicates non-zero result field with significance established to indicate less than zero.
- 2 — indicates non-zero result field with no significance established to indicate greater than zero.
- 3 — not used.

### Interrupt Action

- ◆ Address error:  
Addressing.  
Protection.  
Data error.

### Notes

- ◆ 1. All notes of the Edit instruction are applicable to the Edit and Mark instruction.
- 2. The address of the byte is loaded each time significance is established and a non-zero character is inserted into the result field.



**Notes  
(Cont'd)**

3. The address is loaded into the rightmost 24 bits of General Register 1 (13 or 9). The leftmost eight bits are unaltered.
4. When a single instruction is used to edit multiple fields, the address of the first significant digit of each field is loaded into the register. However, only the address of the last field processed will be available upon completion of the instruction.
5. In processor states  $P_1$  and  $P_2$ , General Register 1 is used. In processor state  $P_3$ , General Register 13 is used. In processor state  $P_4$ , General Register 9 is used.

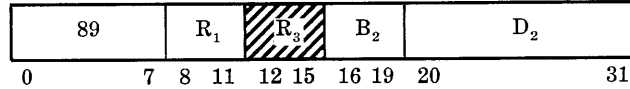
### **Shift Left Single Logical (SLL)**

**General Description**

◆ The entire contents of the general register specified by the first address ( $R_1$ ) are shifted left the number of bit positions specified by the second address ( $B_2/D_2$ ). The  $R_3$  field is ignored.

The second address does not refer to a main memory location. The low-order six bits of the second address are used as the count to specify the number of bits of shifting to be done. The remaining bits are ignored.

**Format  
(RS)**



**Condition Code**

◆ Unchanged.

**Interrupt Action**

◆ None.

**Notes**

- ◆ 1. High-order bits of the register are shifted out and lost.
- 2. Zeros are placed into the right end of the register.
- 3. All 32 bits of the specified register are shifted.

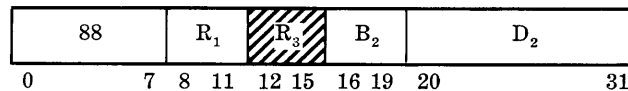
## Shift Right Single Logical (SRL)

### General Description

◆ The entire contents of the general register specified by the first address ( $R_1$ ) are shifted right by the number of bit positions specified by the second address ( $B_2/D_2$ ). The  $R_3$  field is ignored.

The second address does not refer to a main memory location. The low-order six bits of the second address are used as the count to specify the number of bits shifting to be done. The remaining bits are ignored.

### Format (RS)



### Condition Code

◆ Unchanged.

### Interrupt Action

◆ None.

### Notes

- ◆ 1. Low-order bits of the register are shifted out and lost.
- 2. Zeros are placed into the left end of the register.
- 3. All 32 bits of the specified register are shifted; that is, the operation is unsigned.

## Shift Left Double Logical (SLDL)

### General Description

- ◆ The entire contents of the double-length operand (two general registers) — even/odd specified by the first address ( $R_1$ ) are shifted left the number of bit positions specified by the second address ( $B_2/D_2$ ). The  $R_3$  field is ignored.

The second address does not refer to a main memory location. The low-order six bits of the second address are used as the count to specify the number of bits of shifting to be done. The remaining bits are ignored.

### Format (RS)



### Condition Code

- ◆ Unchanged.

### Interrupt Action

- ◆ Address error:  
Specification.

### Notes

- ◆ 1. The first address must specify an even-numbered register.
- 2. All 64 bits of the double-length operand are shifted.
- 3. High-order bits are shifted out and lost.
- 4. Zeros are placed into the low-order end of the odd-numbered register.

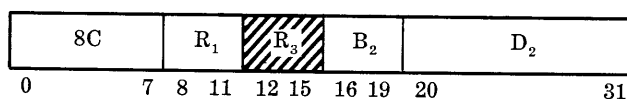
# Shift Right Double Logical (SRDL)

## General Description

◆ The entire contents of the double-length operand (two general registers) — even/odd specified by the first address ( $R_1$ ) are shifted right the number of bit positions specified by the second address ( $B_2/D_2$ ). The  $R_3$  field is ignored.

The second address does not refer to a main memory location. The low-order six bits of the second address are used as the count to specify the number of bits of shifting to be done. The remaining bits are ignored.

## Format (RS)



## Condition Code

◆ Unchanged.

## Interrupt

◆ Address error:  
Specification.

## Notes

- ◆ 1. The first address must specify an even-numbered register.
- ◆ 2. All 64 bits of the double-length operand are shifted.
- ◆ 3. Low-order bits are shifted out and lost.
- ◆ 4. Zeros are placed into the high-order end of the even-numbered register.

## BRANCHING INSTRUCTIONS

### INTRODUCTION

◆ In normal processor operation, instructions are executed in sequential order according to the main memory locations in which they are stored. When branching is performed, a break in this normal sequential execution occurs. Branching instructions provide for referencing another subroutine or repeating a segment of coding or continuing to the next instruction in sequence. When branching occurs, the address specified in the branch instruction replaces the current address in the P counter. The branch address can be specified by an instruction address or it can be obtained from one of the general registers.

The actual branching execution is based on the setting of the condition code or on the contents of a general register as specified in the loop-closing operations.

In a branching operation, the current address in the updated P counter can be stored before the branch address is placed in the P counter. This stored address can be used for linking the new segment of instructions with the segment of instructions from which the branching occurred.

The Execute instruction is listed with the branch instructions, although only a temporary departure from sequential operation is entailed by use of this instruction. The branch address, in this instruction, specifies one instruction to be executed in the instruction sequence. The address in the P counter is not replaced by the branch address and only the instruction located at the address is executed before the sequence is continued based upon the updated P counter.

### SEQUENTIAL EXECUTION

◆ Normally, the P counter instruction address specifies a main memory location from which the next instruction to be executed is fetched. This instruction address is updated in the P counter by the length, in bytes, of the instruction to be executed as indicated by the current P counter. The instruction currently indicated by the P counter is executed and the operation is repeated using the updated P counter to fetch the next instruction.

Instructions can occupy from one halfword (two bytes) up to three halfwords (six bytes). The high-order two bits of the operation code of each instruction designates its length as follows:

00 = halfword instruction (two bytes).

01, 10 = two-halfword instructions (four bytes).

11 = three-halfword instructions (six bytes).

### INSTRUCTION FORMATS

#### RS Format

◆ Branching instructions use the following three instruction formats:

| Op Code |   | R <sub>1</sub> |    | R <sub>3</sub> |    | B <sub>2</sub> |    | D <sub>2</sub> |  |    |
|---------|---|----------------|----|----------------|----|----------------|----|----------------|--|----|
| 0       | 7 | 8              | 11 | 12             | 15 | 16             | 19 | 20             |  | 31 |

#### Description

◆ The contents of the general register specified by B<sub>2</sub> are added to the contents of the D<sub>2</sub> field to obtain the branch address (second operand). The R<sub>1</sub> field specifies the general register that contains the first operand. The R<sub>3</sub> field specifies the general register that contains the third operand.

**RX Format**

| Op Code | R <sub>1</sub> /M | X <sub>2</sub> | B <sub>2</sub> | D <sub>2</sub> |
|---------|-------------------|----------------|----------------|----------------|
| 0       | 7 8               | 11 12          | 15 16          | 19 20          |
|         |                   |                |                | 31             |

*Description*

◆ The contents of the general registers specified by X<sub>2</sub> and B<sub>2</sub> are added to the contents of the D<sub>2</sub> field to obtain the branch address (second operand). The R<sub>1</sub> field specifies the general register which contains the first operand. In a Branch on Condition instruction, the M field is a mask which specifies the condition codes to be tested.

**RR Format**

| Op Code | R <sub>1</sub> /M | R <sub>2</sub> |
|---------|-------------------|----------------|
| 0       | 7 8               | 11 12          |
|         |                   | 15             |

*Description*

◆ The contents of the general register specified by the R<sub>2</sub> field are the branch address (second operand). The R<sub>1</sub> field specifies the general register that contains the first operand. The same register can be specified by R<sub>1</sub> and R<sub>2</sub>. If R<sub>2</sub> is zero, no branching occurs. In a Branch on Condition instruction, the M field is a mask that specifies the condition codes to be tested.

*Notes:*

1. A zero in the X<sub>2</sub> or B<sub>2</sub> field indicates that the corresponding address component is absent.
2. The sequence of operations when using general registers is as follows:
  - a. compute the address.
  - b. store arithmetic or link information.
  - c. replace the P counter with the branch address.

**INTERRUPT ACTION**

◆ Interrupts can occur as a result of an Execute instruction only. The interrupt conditions are as follows:

**Address Error***Addressing*

◆ An address error interrupt occurs when the branch address of an Execute instruction is outside the main memory for the particular installation, or if an Execute instruction is attempted to perform another Execute instruction. The operation is suppressed and the condition code, registers, and main memory are unaltered.

*Specification*

◆ An address error interrupt occurs if the branch address of an Execute instruction is not on a halfword boundary. The operation is suppressed and the condition code, registers, and main memory are unaltered.

# Branch on Condition (BCR) (BC)

## General Description

◆ If the condition code is set to any of the conditions specified by the four-bit mask field ( $M$  or  $M_1$ ), the P counter is replaced by the branch address ( $R_2$  or  $X_2/B_2/D_2$ ). If the four-bit mask field ( $M$  or  $M_1$ ) is not equivalent to the condition code settings, branching does not occur and the next instruction in sequence is executed. The branch is initiated whenever the condition code has a corresponding mask bit set.

## Format (RR)

|          |       |          |
|----------|-------|----------|
| (BCR) 07 | $M_1$ | $R_2$    |
| 0        | 7 8   | 11 12 15 |

## (RX)

|         |     |          |          |       |
|---------|-----|----------|----------|-------|
| (BC) 47 | $M$ | $X_2$    | $B_2$    | $D_2$ |
| 0       | 7 8 | 11 12 15 | 16 19 20 | 31    |

## Condition Code

◆ Unchanged.

## Interrupt Action

◆ None.

## Notes

◆ 1. The four-bit mask in  $M_1$  corresponds, left-to-right, with the four condition codes:

| Instruction Bit | Condition Code |
|-----------------|----------------|
| 8               | 0              |
| 9               | 1              |
| 10              | 2              |
| 11              | 3              |

2. If all mask bits are set ( $M_1 = F_{16}$ ), an unconditional branch is effected.
3. When all mask bits are zero, or if  $R_2$  in the RR format is zero, the instruction is a no-op.
4. When a branch occurs, the leftmost eight-bit portion of the 32-bit P counter (ILC, CC, and mask) is unpredictable. However, the actual condition code and program mask (hardware registers) are unaffected by branching.
5. The contents of the registers specified by the second address are unaltered.

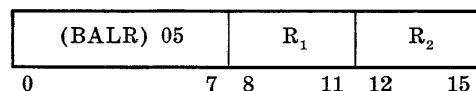


# Branch and Link (BALR) (BAL)

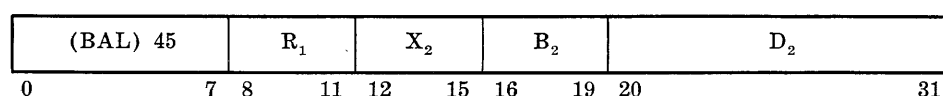
## General Description

◆ The entire 32-bit contents of the P counter are loaded into the general register specified by  $R_1$ . Then, the program branches to the instruction address specified by the branch address ( $R_2$  or  $X_2/B_2/D_2$ ). The instruction length counter, the condition code, the program mask, and the updated instruction address are stored. However, when branching occurs, only the instruction address is replaced.

## Format (RR)



## (RX)



## Condition Code

◆ Unchanged.

## Interrupt Action

◆ None.

## Notes

- ◆ 1. The P counter is stored without branching in the RR format when the  $R_2$  field is zero.
- 2. When a branch occurs, the leftmost eight-bit portion of the 32-bit P counter (ILC, CC, and mask) is unpredictable. However, the actual condition code and program mask (hardware registers) are unaffected by branching.
- 3. The contents of the register specified by the second address are unaltered.

# Branch on Count (BCTR) (BCT)

## General Description

◆ The contents of the general register specified by the  $R_1$  field are algebraically decremented by one. The contents of the register are examined, and if the contents are zero, no branching occurs. If the contents are not zero, the instruction address in the P counter is replaced by the branch address ( $R_2$  or  $X_2/B_2/D_2$ ) and branching occurs.

## Format (RR)

|           |       |          |
|-----------|-------|----------|
| (BCTR) 06 | $R_1$ | $R_2$    |
| 0         | 7 8   | 11 12 15 |

## (RX)

|          |       |          |          |       |
|----------|-------|----------|----------|-------|
| (BCT) 46 | $R_1$ | $X_2$    | $B_2$    | $D_2$ |
| 0        | 7 8   | 11 12 15 | 16 19 20 | 31    |

## Condition Code

◆ Unchanged.

## Interrupt Action

◆ None.

## Notes

- ◆ 1. The subtraction executes as in fixed-point arithmetic with all 32 bits participating.
- 2. An initial count of zero in the  $R_1$  field results in branching, because subtraction occurs before testing the contents of the register. If the value is zero, branching occurs and the result is minus one. To effect a *no branch*, the contents of the  $R_1$  field must be 1.
- 3. The contents of the registers specified by the second address are unaltered.
- 4. When branching occurs, the leftmost eight-bit portion of the 32-bit P counter (ILC, CC, and mask) is unpredictable. However, the actual condition code and program mask (hardware registers) are unaffected by branching.
- 5. In the RR format, if the  $R_2$  field is zero, counting is performed without branching.
- 6. If a negative number appears in  $R_1$ , an overflow condition occurs when this field is decremented. However, this overflow is ignored.
- 7. Overflow from a maximum negative number to a maximum positive number is ignored.

# Branch on Index High (BXH)

## General Description

◆ The operand specified by the third address ( $R_3$ ) is added to the operand specified by the first address ( $R_1$ ) and the sum is algebraically compared with the operand specified by the third address ( $R_3$ ), if  $R_3$  specifies an odd register. If  $R_3$  specifies an even register, the sum is algebraically compared with  $R_3 + 1$ . If the sum is low or equal, branching does not occur and the next instruction is executed. If the sum is high, the instruction address in the P counter is replaced by the branch address ( $B_2/D_2$ ) and branching occurs.

## Format (RS)

|    |       |       |       |       |
|----|-------|-------|-------|-------|
| 86 | $R_1$ | $R_3$ | $B_2$ | $D_2$ |
| 0  | 7 8   | 11 12 | 15 16 | 19 20 |
|    |       |       |       | 31    |

## Condition Code

◆ Unchanged.

## Interrupt Action

◆ None.

## Notes

- ◆ 1. The sum replaces the operand specified by the first address ( $R_1$ ) regardless of the comparison. The sum replaces ( $R_1$ ) after the comparison has been made.
- 2. Overflow is not recognized.
- 3. The contents of the register specified by  $R_3$  or  $R_3 + 1$  are unaltered.
- 4. When a branch occurs, the leftmost eight-bit positions of the 32-bit P counter (ILC, CC, and mask) are unpredictable. However, the actual condition code and program mask (hardware registers) are unaffected by branching.

# Branch on Index Low or Equal (BXLE)

## General Description

◆ The operand specified by the third address ( $R_3$ ) is added to the operand specified by the first address ( $R_1$ ) and the sum is algebraically compared with the operand specified by the third address ( $R_3$ ), if  $R_3$  specifies an odd register. If  $R_3$  specifies an even register, the sum is algebraically compared with  $R_3 + 1$ . If the sum is high, branching does not occur and the next instruction in sequence is executed. If the sum is low or equal, the instruction address in the P counter is replaced by the branch address ( $B_2/D_2$ ) and branching occurs.

## Format (RS)

|    |       |       |       |       |    |    |    |    |    |
|----|-------|-------|-------|-------|----|----|----|----|----|
| 87 | $R_1$ | $R_3$ | $B_2$ | $D_2$ |    |    |    |    |    |
| 0  | 7     | 8     | 11    | 12    | 15 | 16 | 19 | 20 | 31 |

## Condition Code

◆ Unchanged.

## Interrupt Action

◆ None.

## Notes

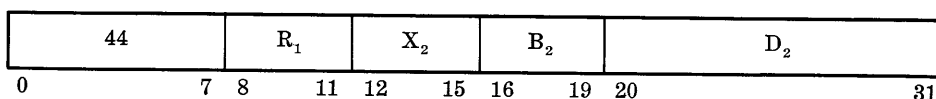
- ◆ 1. The sum replaces the operand specified by the first address ( $R_1$ ) regardless of the comparison. The sum replaces ( $R_1$ ) after the comparison has been made.
- 2. Overflow is not recognized.
- 3. The contents of the register specified by  $R_3$  or  $R_3 + 1$  are unaltered.
- 4. When a branch occurs, the leftmost eight-bit positions of the 32-bit P counter (ILC, CC, and mask) are unpredictable. However, the actual condition code and program mask (hardware registers) are unaffected by branching.

**Execute  
(EX)**

**General Description**

◆ The instruction in the location specified by the second address ( $X_2/B_2/D_2$ ) is modified by the contents of the register specified by the first address ( $R_1$ ). Then, the modified instruction is executed and control is returned to the instruction following the Execute instruction.

**Format  
(RX)**



**Condition Code**

◆ May be set by the instruction being modified and executed.

**Interrupt Action**

◆ Address error:  
Addressing.  
Specification.

**Notes**

- ◆ 1. Bits 8–15 of the subject instruction are “OR”ed with bits 24–31 of the register specified by the first address ( $R_1$ ).
- 2. If  $R_1$  is zero, no modification takes place.
- 3. The ILC is set to two (length of the Execute) and the P counter is set to the address of the instruction following the Execute instruction.
- 4. The contents of  $R_1$  and the subject instruction in main memory are unaltered.
- 5. Interrupts are inhibited until the subject instruction has been completed.
- 6. When the subject instruction is a successful branching instruction, the P counter is updated by the branch address.

## FLOATING-POINT INSTRUCTIONS

### INTRODUCTION

◆ Floating-point arithmetic instructions provide the capability to process operands of large magnitude with precise results.

A floating-point number is made up of three parts: a sign, an exponent and a mantissa. The sign portion applies to the mantissa. The exponent is a power to which the number 16 is raised. The mantissa is a hexadecimal number with an assumed radix point to the left of the high-order digit. The quantity that the floating-point number represents is obtained by multiplying the mantissa by the number 16 raised to the power represented by the exponent.

Four floating-point registers are provided, each of which is 64 bits long. These registers are numbered 0, 2, 4 and 6.

Included in this set are instructions for loading, adding, subtracting, comparing, multiplying, dividing, storing, and controlling signs of short and long operands.

Addition, subtraction, multiplication, and division produce normalized results. Addition and subtraction can also produce unnormalized results. Operands can be normalized, or unnormalized, in any floating-point operation.

Sign control, add, subtract, and compare operation results are indicated in the condition code settings.

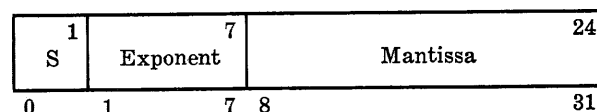
### DATA FORMATS

◆ Floating-point numbers are fixed in length and are either full-word *short* or double-word *long* in format.

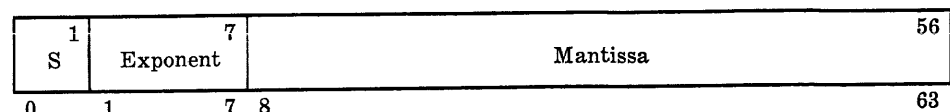
The first bit in both formats is the sign of the mantissa. A 1 bit represents a minus sign and a 0 bit represents a plus sign. The next seven bits represent the exponent. The mantissa contains six hexadecimal digits (short floating-point number) or 14 (long floating-point number) hexadecimal digits.

The short format allows for faster processing and uses less storage. Because floating-point registers are 64 bits long, the rightmost 32 bits are ignored when dealing with short operands. When the short format is specified, all operands and the result are 32 bits long. When using the long format, which provides greater precision, all operands are 64 bits long and require the full register.

Short Floating-Point Number



Long Floating-Point Number



## REPRESENTATION OF NUMBERS

◆ The mantissa is always represented in hexadecimal. An assumed radix point is always immediately to the left of the high-order digit of the mantissa.

The exponent, bits 1 through 7, indicates the power to which the number 16 must be raised. The range of the exponent is from  $-64$  to  $+63$  corresponding to the binary value of 0–127. The power is equal to the binary number minus 64, as shown in following table:

| Exponent          | Decimal Equivalent | Power   |
|-------------------|--------------------|---------|
| $(1\ 111\ 111)_2$ | 127 $-64$          | $= +63$ |
| $(1\ 000\ 111)_2$ | 71 $-64$           | $= +7$  |
| $(0\ 000\ 000)_2$ | 0 $-64$            | $= -64$ |

Because the value  $(64)_{10}$  represents the power zero, this technique is called excess 64 notation.

The sign of a result from addition, subtraction, multiplication, or division with a zero mantissa is positive. A zero sign, zero exponent, and zero mantissa in a floating-point number is called true zero.

## NORMALIZATION

◆ A floating-point number with a mantissa containing a non-zero, high-order, hexadecimal digit is called a normalized number. An unnormalized number has one or more high-order hexadecimal zero digits in the mantissa. To change an unnormalized number into a normalized number, the mantissa is shifted to the left until the high-order digit is non-zero. Then, the exponent is decremented by the number of digits shifted.

Generally, normalization occurs when the intermediate arithmetic result is changed to the final result. However, in multiplication and division operations, normalization occurs before the arithmetic process.

Floating-point operations are performed with, or without, normalization. Most operations are performed in only one way; however, addition and subtraction may be performed either way as specified.

When normalization is not performed, high-order zeros in the result mantissa are not eliminated. Depending on the original operands, the result may, or may not, be normalized.

Initial operands in both normalized and unnormalized operations need not be in normalized form. Because normalization takes place on hexadecimal digits, the three high-order bits of a normalized mantissa can be zero.

## INSTRUCTION FORMATS

### RX Format

◆ The following two instruction formats are used for floating-point operations:

| Op Code | $R_1$ | $X_2$ | $B_2$ | $D_2$ |
|---------|-------|-------|-------|-------|
| 0       | 7 8   | 11 12 | 15 16 | 19 20 |
|         |       |       |       | 31    |

### Description

◆ An address is formed by adding the contents of general registers  $X_2$  and  $B_2$  to the displacement field  $D_2$ . This address specifies a main memory location that contains the second operand in the operation.  $R_1$  designates the floating-point register containing the first operand.

**RR Format**

| Op Code | R <sub>1</sub> | R <sub>2</sub> |
|---------|----------------|----------------|
| 0 7     | 8 11           | 12 15          |

*Description*

◆ In this format, R<sub>1</sub> designates the address of the floating-point register holding the first operand. R<sub>2</sub> is the address of the floating-point register holding the second operand. The first and second operands can be the same and are designated by identical R<sub>1</sub> and R<sub>2</sub> addresses.

*Notes:*

1. Register addresses specified by the R<sub>1</sub> and R<sub>2</sub> fields must be 0, 2, 4, or 6 or an address error (specification) interrupt occurs.
2. A short operand must be located on a word boundary and a long operand must be on a double-word boundary; if not, an address error (specification) interrupt occurs.
3. Floating-point registers are used by floating-point instructions only.
4. A zero in an X<sub>2</sub> or B<sub>2</sub> field shows that there is no address component to enter in forming an address.
5. Except for the instructions Store (long) and Store (short), results of floating-point operations replace the first operand.
6. Except for the storing of the result, the contents of floating-point registers, general registers, and main memory locations used in the operations are not changed.
7. It is possible to designate the same general register to specify both operand locations and address generation. Addresses are generated before execution.

**CONDITION CODE UTILIZATION**

◆ The condition code reflects results of floating-point sign control, add, subtract, and compare instructions. The code is not changed by any other floating-point operation. Decision-making by branch on condition instructions can be done after those instructions that set the code.

For most arithmetic and load instructions, Condition Codes 0, 1, or 2 indicate respectively a zero, or less than, or greater than zero content, of the result. Condition Code 3 is set for overflow of the result in arithmetic instructions only. In comparison instructions, the Condition Codes 0, 1, or 2 show, respectively, that the first operand is either equal to, less than, or greater than the second operand.

Instructions that cause the condition code to be set and the meaning of the setting are as follows:

| Instruction                      | Condition Code Setting |        |        |          |
|----------------------------------|------------------------|--------|--------|----------|
|                                  | 0                      | 1      | 2      | 3        |
| Add Normalized Short/Long        | Zero                   | < Zero | > Zero | Overflow |
| Add Unnormalized Short/Long      | Zero                   | < Zero | > Zero | Overflow |
| Compare Short/Long               | Equal                  | Low    | High   | —        |
| Load and Test Short/Long         | Zero                   | < Zero | > Zero | —        |
| Load Complement Short/Long       | Zero                   | < Zero | > Zero | —        |
| Load Negative Short/Long         | Zero                   | < Zero | —      | —        |
| Load Positive Short/Long         | Zero                   | —      | > Zero | —        |
| Subtract Normalized Short/Long   | Zero                   | < Zero | > Zero | Overflow |
| Subtract Unnormalized Short/Long | Zero                   | < Zero | > Zero | Overflow |



## **INTERRUPT ACTION**

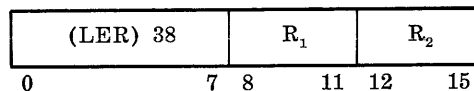
|                           |  |
|---------------------------|--|
| <b>Address Error</b>      | <p>◆ The following interrupt conditions can occur as a result of a floating-point instruction.</p>   |
| <i>Addressing</i>         | <p>◆ An address error interrupt occurs when an address in the RX instruction format specifies a location outside the available main memory. The operation is terminated at the point of error. The result data and the condition code (if affected) are unpredictable.</p>   |
| <i>Specification</i>      | <p>◆ An address error interrupt occurs if a short operand is not located on a word boundary or a long operand is not located on a double-word boundary. An address error interrupt also occurs if a floating-point register other than 0, 2, 4 or 6 is specified. The instruction is suppressed. The condition code, the data in main memory, and the registers remain unchanged. Address restrictions do not apply to the <math>X_2</math>, <math>B_2</math> and <math>D_2</math> components of the instruction.</p>  |
| <i>Protection</i>         | <p>◆ An address error interrupt occurs when the protection key and the storage key of the result location do not match. The operation is suppressed. The condition code, the data in main memory, and the registers remain unchanged. (This interrupt can only occur if the memory protect feature is installed.)</p>  |
| <b>Significance Error</b> | <p>◆ A significance error interrupt occurs when the result mantissa of an add or subtract operation is zero. A program interrupt occurs if the significance error mask bit in the Interrupt Mask Register of the current state is set to 1. The operation is completed, the exponent is unaltered, and the interrupt is taken. If the significance error mask bit is zero, the interrupt is prohibited and the operation is completed by setting the result to true zero (zero sign, zero exponent, and zero mantissa). In either case, the condition code is set to zero.</p> |
| <b>Divide Error</b>       | <p>◆ A divide error interrupt occurs if division by zero is attempted.</p>   |
| <b>Exponent Overflow</b>  | <p>◆ An exponent overflow interrupt occurs when the result exponent overflows and the mantissa is not zero. The operation is terminated and the result data is unpredictable. Addition and subtraction set the condition code to 3. Multiplication and division do not affect the condition code setting.</p>  |
| <b>Exponent Underflow</b> | <p>◆ An exponent underflow interrupt occurs when the result exponent is less than zero and the result mantissa is not zero. The operation is completed by setting the result to true zero (zero sign, zero exponent, and zero mantissa). Addition and subtraction set the condition code to zero. Multiplication and division do not affect the condition code setting.</p>  |

**Load**  
**(LER) (LE) (LDR) (LD)**

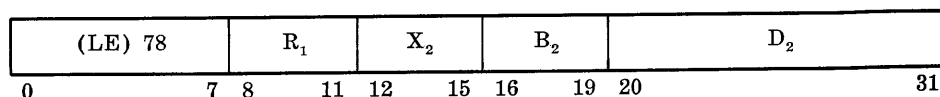
**General Description**

◆ The operand specified by the second address ( $R_2$  or  $X_2/B_2/D_2$ ) is loaded into the floating-point register specified by the first address ( $R_1$ ).

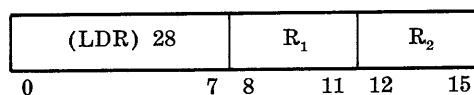
**Format**  
**(RR Short)**



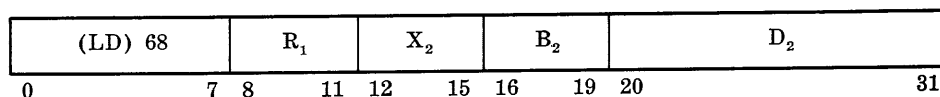
**(RX Short)**



**(RR Long)**



**(RX Long)**



**Condition Code**

◆ Unchanged.

**Interrupt Action**

◆ Address error:  
Addressing (RX format).  
Specification.

**Notes**

- ◆ 1. The operand specified by the second address is unaltered.
- 2. Exponent overflow, underflow, or lost significance cannot occur.
- 3. The low-order half of the register specified by the first address is unaltered when short operands are used.

**Load and Test  
(LTER) (LTDR)**

**General Description**

- ◆ The operand in the floating-point register specified by the second address ( $R_2$ ) is loaded into the floating-point register specified by the first address ( $R_1$ ). The sign and magnitude of the loaded operand determine the condition code.

**Format  
(RR Short)**

|           |       |       |
|-----------|-------|-------|
| (LTER) 32 | $R_1$ | $R_2$ |
| 0 7 8     | 11 12 | 15    |

**(RR Long)**

|           |       |       |
|-----------|-------|-------|
| (LTDR) 22 | $R_1$ | $R_2$ |
| 0 7 8     | 11 12 | 15    |

**Condition Code**

- ◆ 0 — result mantissa is zero.
- 1 — result mantissa is less than zero.
- 2 — result mantissa is greater than zero.
- 3 — not used.

**Interrupt Action**

- ◆ Address error:  
Specification.

**Notes**

- ◆ 1. If  $R_1$  and  $R_2$  are equal, the operation is equivalent to a test without data movement.
- 2. The operand specified by the second address is unaltered.
- 3. Short operands do not alter the low-order half of the register specified by the first address.

# **Load Complement (LCER) (LCDR)**

## **General Description**

- ◆ The operand in the floating-point register specified by the second address ( $R_2$ ) is loaded into the floating-point register specified by the first address ( $R_1$ ) and the sign is changed to the opposite value. The sign and magnitude of the loaded operand determine the condition code.

## **Format (RR Short)**

|           |       |       |
|-----------|-------|-------|
| (LCER) 33 | $R_1$ | $R_2$ |
| 0 7 8     | 11 12 | 15    |

## **(RR Long)**

|           |       |       |
|-----------|-------|-------|
| (LCDR) 23 | $R_1$ | $R_2$ |
| 0 7 8     | 11 12 | 15    |

## **Condition Code**

- ◆ 0 — result mantissa is zero.
- 1 — result mantissa is less than zero.
- 2 — result mantissa is greater than zero.
- 3 — not used.

## **Interrupt Action**

- ◆ Address error:  
Specification.

## **Notes**

- ◆ 1. The exponent and mantissa are unaltered.
- 2. Short operands do not alter the low-order half of the register specified by the first address.

**Load Positive  
(LPER) (LPDR)**

**General Description**

- ◆ The operand in the floating-point register specified by the second address ( $R_2$ ) is loaded into the floating-point register specified by the first address ( $R_1$ ) and the operand sign is made plus.

**Format  
(RR Short)**

|           |       |       |
|-----------|-------|-------|
| (LPER) 30 | $R_1$ | $R_2$ |
| 0 7 8     | 11 12 | 15    |

**(RR Long)**

|           |       |       |
|-----------|-------|-------|
| (LPDR) 20 | $R_1$ | $R_2$ |
| 0 7 8     | 11 12 | 15    |

**Condition Code**

- ◆ 0 — result mantissa is zero.
- 1 — not used.
- 2 — result mantissa is greater than zero.
- 3 — not used.

**Interrupt Action**

- ◆ Address error:  
Specification.

**Notes**

- ◆ 1. The exponent and mantissa are unaltered.
- 2. Short operands do not alter the low-order half of the register specified by the first address.

# **Load Negative (LNER) (LNDR)**

## **General Description**

- ◆ The operand in the floating-point register specified by the second address ( $R_2$ ) is loaded into the floating-point register specified by the first address ( $R_1$ ) and the operand sign is made minus.

## **Format (RR Short)**

|           |       |       |
|-----------|-------|-------|
| (LNER) 31 | $R_1$ | $R_2$ |
| 0 7 8     | 11 12 | 15    |

## **(RR Long)**

|           |       |       |
|-----------|-------|-------|
| (LNDR) 21 | $R_1$ | $R_2$ |
| 0 7 8     | 11 12 | 15    |

## **Condition Code**

- ◆ 0 — result mantissa is zero.
- 1 — result mantissa is less than zero.
- 2 — not used.
- 3 — not used.

## **Interrupt Action**

- ◆ Address error:  
Specification.

## **Notes**

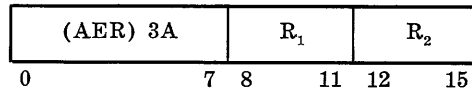
- ◆ 1. The exponent and mantissa are unaltered.
- 2. Short operands do not alter the low-order half of the register specified by the first address.

**Add Normalized**  
**(AER) (AE) (ADR) (AD)**

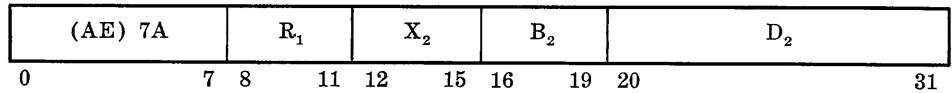
**General Description**

◆ The operand specified by the second address ( $R_2$  or  $X_2/B_2/D_2$ ) is added to the operand in the floating-point register specified by the first address ( $R_1$ ). The *normalized* sum is loaded into the register specified by the first address. The sign and magnitude of the sum determine the condition code.

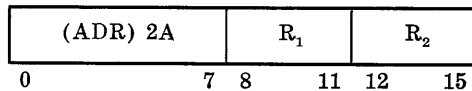
**Format**  
**(RR Short)**



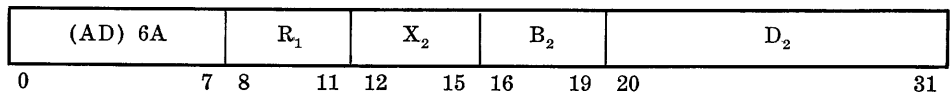
**(RX Short)**



**(RR Long)**



**(RX Long)**



**Condition Code**

- ◆ 0 — result mantissa is zero.  
 1 — result mantissa is less than zero.  
 2 — result mantissa is greater than zero.  
 3 — result exponent overflows.

**Interrupt Action**

- ◆ Address error:  
 Addressing (RX format).  
 Specification.  
 Significance error.  
 Exponent overflow.  
 Exponent underflow.

**Notes**

- ◆ 1. To perform normalized addition, the computer must scale the two operands. Scaling consists of comparing the exponents of the two operands. If they do not agree, the mantissa with the smaller exponent operand is shifted right. Its exponent is increased by one for each digit right-shifted, until the two exponents agree. Then, the mantissas are added algebraically to form an intermediate sum. If an overflow carry occurs, the intermediate sum is right-shifted one digit and its exponent is increased by one. If this causes an overflow, an exponent overflow interrupt condition occurs.

For short operands, the intermediate sum consists of seven hexadecimal digits and a possible carry. The low-order digit is the guard digit which is retained from the mantissa which is shifted right. Only one guard digit participates in the mantissa addition. The guard digit is zero if no shift occurs.

**Notes  
(Cont'd)**

- For long operands, the intermediate sum consists of fourteen hexadecimal digits and a possible carry. No guard digit is retained.
2. After addition, the intermediate sum is left-shifted until all high-order zero hexadecimal digits have been eliminated. The vacated low-order digits are made zero and the exponent is decremented by one for each zero digit shifted. If no left-shift takes place, the intermediate sum is truncated to the proper mantissa length. If the exponent underflows (exceeds  $-64$ ) during normalization, the floating-point number is made true zero and an exponent underflow interrupt occurs.
  3. No normalization is performed when the intermediate sum is zero. The sum mantissa is unaltered and a significance error interrupt occurs. If a significance error interrupt is prohibited by the interrupt mask, the quantity is made true zero and a significance error interrupt does not occur.
  4. Initial operands need not be in normalized form.
  5. The sign of the sum is determined by the rules of algebra. A zero sum is always plus.
  6. Short operands do not alter the low-order halves of the registers specified by the address fields.

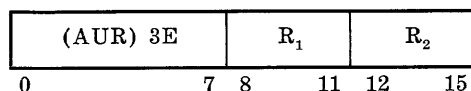


# **Add Unnormalized** (AUR) (AU) (AWR) (AW)

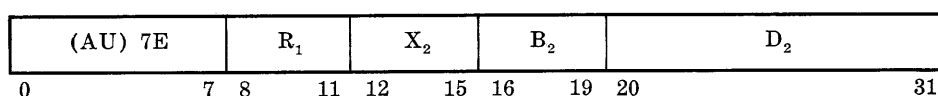
## **General Description**

◆ The operand specified by the second address ( $R_2$  or  $X_2/B_2/D_2$ ) is added to the operand in the floating-point register specified by the first address ( $R_1$ ). The *unnormalized* sum is loaded into the register specified by the first address. The sign and magnitude of the loaded sum determine the condition code.

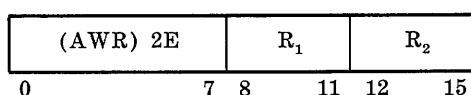
## **Format** (RR Short)



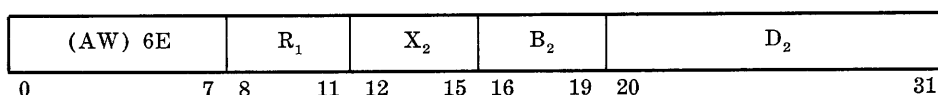
## (RX Short)



## (RR Long)



## (RX Long)



## **Condition Code**

- ◆ 0 — result mantissa is zero.
- 1 — result mantissa is less than zero.
- 2 — result mantissa is greater than zero.
- 3 — result exponent overflows.

## **Interrupt Action**

- ◆ Address error:  
Addressing (RX format).  
Specification.  
Exponent overflow.  
Significance.

## **Notes**

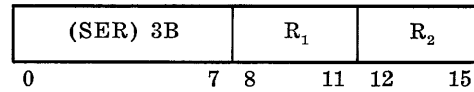
- ◆ 1. The Add Unnormalized is similar to the Add Normalized, except that the sum is not normalized by this instruction and exponent underflow cannot occur.

# **Subtract Normalized** **(SER) (SE) (SDR) (SD)**

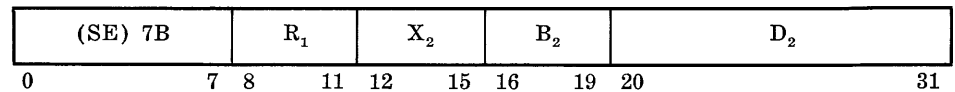
## **General Description**

◆ The operand specified by the second address ( $R_2$  or  $X_2/B_2/D_2$ ) is subtracted from the operand in the floating-point register specified by the first address ( $R_1$ ). The *normalized* difference is loaded into the register specified by the first address. The sign and magnitude of the difference determine the condition code.

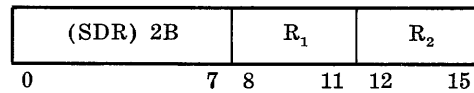
## **Format** **(RR Short)**



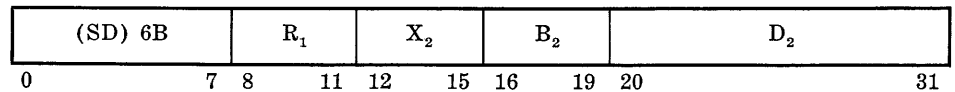
## **(RX Short)**



## **(RR Long)**



## **(RX Long)**



## **Condition Code**

- ◆ 0 — result mantissa is zero.
- 1 — result mantissa is less than zero.
- 2 — result mantissa is greater than zero.
- 3 — result exponent overflows.

## **Interrupt Action**

- ◆ Address error:  
Addressing (RX format).  
Specification.  
Significance error.  
Exponent overflow.  
Exponent underflow.

## **Notes**

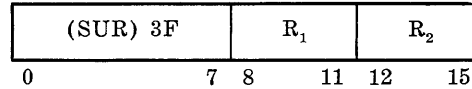
- ◆ 1. The Subtract Normalized is the same as the Add Normalized, except that the sign of the second operand is changed to the opposite value before addition. A zero difference is always positive.

**Subtract  
Unnormalized  
(SUR) (SU) (SWR) (SW)**

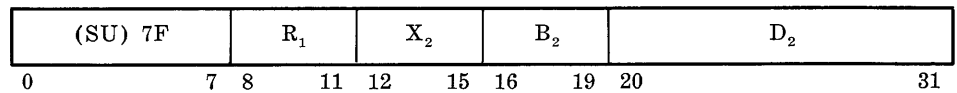
**General Description**

◆ The operand specified by the second address ( $R_2$  or  $X_2/B_2/D_2$ ) is subtracted from the operand in the floating-point register specified by the first address ( $R_1$ ). The *unnormalized* difference is loaded into the register specified by the first address. The sign and magnitude of the difference determine the condition code.

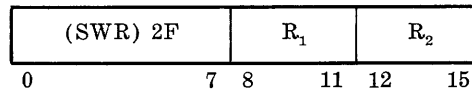
**Format  
(RR Short)**



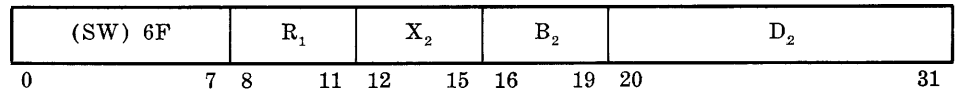
**(RX Short)**



**(RR Long)**



**(RX Long)**



**Condition Code**

- ◆ 0 — result mantissa is zero.  
 1 — result mantissa is less than zero.  
 2 — result mantissa is greater than zero.  
 3 — result exponent overflows.

**Interrupt Action**

- ◆ Address error:  
     Addressing (RX format).  
     Specification.  
     Significance error.  
     Exponent overflow.

**Notes**

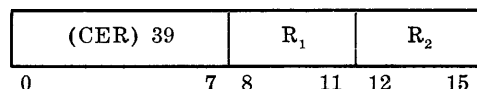
- ◆ 1. Subtract Unnormalized differs from Subtract Normalized only in that the difference is not normalized before it is loaded into the result register.  
 2. Exponent underflow cannot occur.

# **Compare (CER) (CE) (CDR) (CD)**

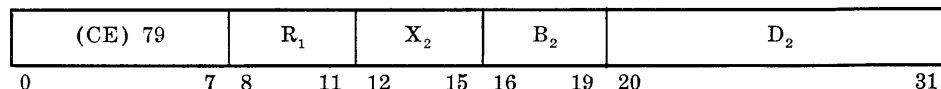
## **General Description**

◆ The operand in the floating-point register specified by the first address ( $R_1$ ) is algebraically compared to the operand specified by the second address ( $R_2$  or  $X_2/B_2/D_2$ ). The result determines the condition code.

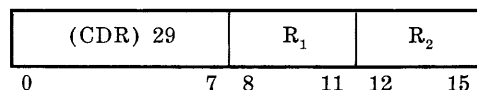
## **Format (RR Short)**



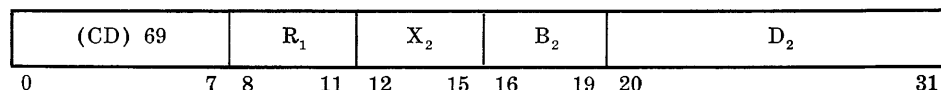
## **(RX Short)**



## **(RR Long)**



## **(RX Long)**



## **Condition Code**

- ◆ 0 — operands are equal.
- 1 — operand specified by the first address is less than the one specified by the second address.
- 2 — operand specified by the first address is greater than the one specified by the second address.
- 3 — not used.

## **Interrupt Action**

- ◆ Address error:  
Addressing (RX format).  
Specification.

## **Notes**

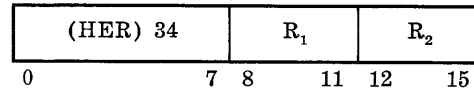
- ◆ 1. Comparison takes into account the sign, exponent, and mantissa of each number. Exponent inequality is not decisive for magnitude determination since the mantissas may have different numbers of leading zeros. The operands are scaled, as in Subtract Normalized, and if the mantissa of each operand is zero, the numbers are considered equal regardless of the sign and exponent.
- 2. Both operands are unaltered.

**Halve  
(HER) (HDR)**

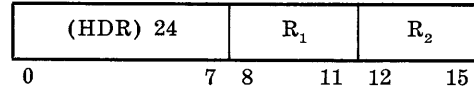
**General Description**

- ◆ The operand in the floating-point register specified by the second address ( $R_2$ ) is divided by two. The quotient is loaded into the floating-point register specified by the first address ( $R_1$ ).

**Format  
(RR Short)**



**(RR Long)**



**Condition Code**

- ◆ Unchanged.

**Interrupt Action**

- ◆ Address error:  
Specification.

**Notes**

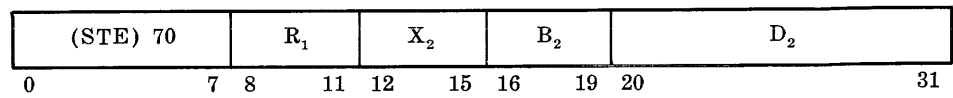
- ◆ 1. The difference between the Halve instruction and a Divide instruction with a divisor of two, is that no normalization and no zero mantissa testing takes place. The sign and exponent are unaltered and the mantissa is shifted right one bit.
- 2. Short operands do not alter the low-order half of the result register.

**Store  
(STE) (STD)**

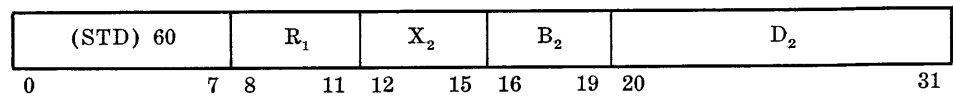
**General Description**

◆ The contents of the floating-point register specified by the first address ( $R_1$ ) are stored in the main memory location specified by the second address ( $X_2/B_2/D_2$ ).

**Format  
(RX Short)**



**(RX Long)**



**Condition Code**

◆ Unchanged.

**Interrupt Action**

◆ Address error:  
Addressing.  
Specification.  
Protection.

**Notes**

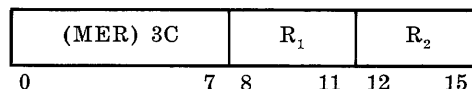
- ◆ 1. The first operand is unaltered.
- 2. Short operands do not alter the low-order half of the register specified by the second address.

**Multiply**  
**(MER) (ME)**  
**(MDR) (MD)**

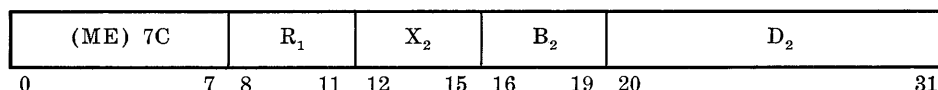
**General Description**

◆ The operand in the floating-point register specified by the first address ( $R_1$ ) is multiplied by the operand specified by the second address ( $R_2$  or  $X_2/B_2/D_3$ ). The *normalized* product is loaded into the register specified by the first address.

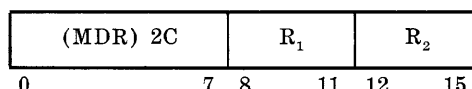
**Format**  
**(RR Short)**



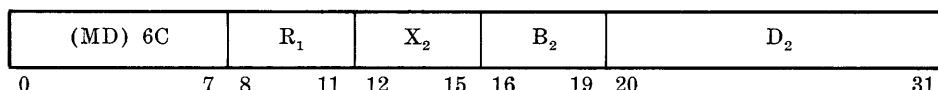
**(RX Short)**



**(RR Long)**



**(RX Long)**



**Condition Code**

◆ Unchanged.

**Interrupt Action**

◆ Address error:  
Addressing (RX format).  
Specification.  
Exponent overflow.  
Exponent underflow.

**Notes**

- ◆ 1. The exponents of the two operands are added, and the sum is reduced by 64 to form an intermediate exponent. The mantissas are normalized as described in the Add Normalize instruction, and multiplied to form an intermediate mantissa. The intermediate mantissa is then normalized (reducing its exponent by one for each digit left shifted) to form the final product.
- 2. The sign of the product is determined by the rules of algebra.
- 3. If the product mantissa is zero, the final product is made true zero.
- 4. If the final product exponent is greater than 127, an exponent overflow interrupt occurs.
- 5. If final product exponent is less than zero, an exponent underflow interrupt occurs.
- 6. For short operands, the low-order half of the register specified by the first address *is used* in the calculation of the intermediate mantissa. The product mantissa has the full 14 digits as in the long format and the two low-order digits are always zero.
- 7. The least significant digit of the double word results of a floating point Multiply (Long) may not be the same on the 70/55 processor as the 70/35 and 70/45 since the algorithm for this instruction is different on the 70/55. Final product digits above the least significant are identical on all processors.

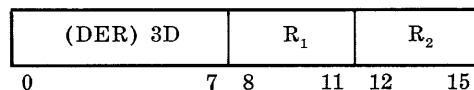
# Divide (DER) (DE) (DDR) (DD)

## General Description

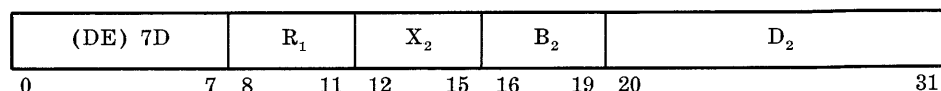
◆ The operand (dividend) in the floating-point register specified by the first address ( $R_1$ ) is divided by the operand divisor specified by the second address ( $R_2$  or  $X_2/B_2/D_2$ ). The *normalized* quotient is stored in the register specified by the first address. The remainder is not retained.

## Format

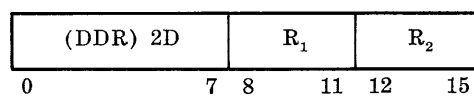
(RX Short)



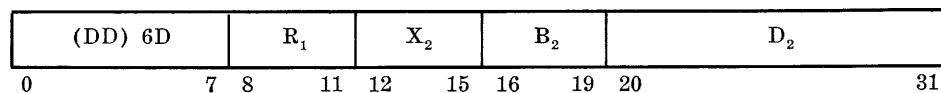
## (RX Short)



## (RR Long)



## (RX Long)



## Condition Code

◆ Unchanged.

## Interrupt Action

◆ Address error:  
 Addressing (RX format).  
 Specification.  
 Exponent overflow.  
 Exponent underflow.  
 Divide error.

## Notes

- ◆ 1. The exponents of the two operands are subtracted and the difference is increased by 64 to form an intermediate exponent. The mantissas are normalized as described in the Subtract Normalize instruction, and divided to form the mantissa of the intermediate quotient. The intermediate exponent and mantissa are normalized to form a final quotient.
- 2. If the dividend (first operand) is zero, the quotient is made true zero.
- 3. If the divisor (second operand) is zero, a divide error interrupt occurs.
- 4. The sign of the quotient is determined by the rules of algebra.
- 5. If the final quotient exponent is less than zero, the final quotient is made true zero and an exponent underflow interrupt occurs.
- 6. If the final quotient exponent exceeds 127, an exponent overflow interrupt occurs.
- 7. For short operands, the low-order halves of the registers are unaltered.



**OPTIONAL  
FEATURES**  
**FEATURE 5001  
MEMORY PROTECT**

**Operational  
Characteristics**

◆ Data in memory can be protected from destruction by the erroneous storing of information during program execution through the optional Memory Protect feature. Feature 5001-35 is applicable to the 70/35 Processor; feature 5001-45 is applicable to the 70/45 Processor; feature 5001-55 is applicable to the 70/55 Processor.

◆ Memory protection is accomplished by a four-bit storage key associated with each block of 2,048 bytes of main memory. Whenever data is to be stored in main memory during the execution of an instruction, the four-bit protection key in the Interrupt Status register for the current program state is compared with the four-bit storage key. During a channel-to-memory data transfer, the protection key (as specified in the channel address word) is compared with the storage key. If the storage and protection keys are equal, or either one is zero, the storage of data is completed.

If the storage and protection keys do not match (neither is zero), the execution of an instruction that stores data into memory is suppressed or terminated. An address error (protection) interrupt occurs, and the protected memory remains unaltered. If the storage and protection keys mismatch during a channel-to-memory data transfer, the data transfer is terminated and a channel termination interrupt occurs. The protected memory is unaltered and the indication of mismatch is stored in the input/output channel registers in scratch-pad memory for the specified channel.

The storage key can be changed by the privileged instruction Set Storage Key and can be inspected by the privileged instruction Insert Storage Key.

When the Memory Protect feature is not installed and the protection key is non zero, an address error (specification) interrupt occurs.

**FEATURE 5002  
ELAPSED TIME  
CLOCK**

**Operational  
Characteristics**

◆ The elapsed time clock is an optional feature available on the 70/35, 70/45, and 70/55 Processors. Feature 5002-35 is applicable to the 70/35 Processor; feature 5002-45 is applicable to the 70/45 Processor; feature 5002-55 is applicable to the 70/55 Processor.

◆ The elapsed time clock occupies a full word beginning at main memory location 80. The word is treated as a signed binary operand and follows the rules of fixed-point arithmetic.

The clock count is performed by decrementing bit positions 21 and 23 every 1/60th of a second (60 cycle processor) or by decrementing bit positions 21 and 23 every 1/50th of a second (50 cycle processor). In either case, the effect is equivalent to reducing the elapsed time clock by one in bit position 23 every 1/300th of a second (every 3.3 milliseconds). When the clock goes from positive to negative, an elapsed time clock interrupt occurs.

Normally, an updated elapsed time clock is available after the completion of each instruction execution. However, when input/output data transmission approaches the limit of main memory capability, or a Read Direct instruction time is excessive, elapsed time clock updating can be skipped.

When an elapsed time clock interrupt occurs, the clock may have been decremented several times before the interrupt takes effect, depending on the execution time of the current instruction.

**FEATURE 5003  
DIRECT CONTROL**

**Operational  
Characteristics**

◆ The Direct Control feature enables one 70/35, 70/45, or 70/55 processor program to directly signal the programs of from one to five other processors over an interface independent of the input/output channels. The processors directly connected by this feature may be remotely located up to 500 cable feet from the transmitting processor. Feature 5003-35 is applicable to the 70/35 Processor; feature 5003-45 is applicable to the 70/45 Processor; feature 5003-55 is applicable to the 70/55 Processor.

◆ Two additional privileged instructions are provided with this option, Write Direct and Read Direct, which initiate the transfer of one byte of control information between processor memories, and which signal the opposite unit (by external interrupt) upon execution of an instruction.

This feature can also initiate initial program loading in a remote processor which is in a stopped state. In this case, the Load Unit Switches on the console of the processor being signaled specify the device from which the loading is to occur and the information byte is ignored.

**FEATURE 5015  
SELECTOR  
CHANNEL\***

◆ This feature is applicable to the 70/45 Processor. It provides two selector channels with four input/output trunks (two trunks per channel).

**FEATURE 5016  
SELECTOR  
CHANNEL\***

◆ This feature is applicable to the 70/45 Processor. It provides three selector channels with six input/output trunks (two trunks per channel).

**FEATURE 5020  
SELECTOR  
CHANNEL\*\***

◆ This feature is applicable to the 70/55 Processor. It provides two selector channels and four input/output trunks.

**FEATURE 5022  
SELECTOR  
CHANNEL\*\***

◆ This feature is applicable to the 70/55 Processor. It provides four selector channels and six input/output trunks.

**FEATURE 5024  
SELECTOR  
CHANNEL\*\***

◆ This feature is applicable to the 70/55 Processor. It provides six selector channels and twelve input/output trunks.

**FEATURE 5030  
SELECTOR  
CHANNEL\*\*\***

◆ This feature is applicable to the 70/35 Processor. It provides one selector channel and two input/output trunks.

**FEATURE 5031  
SELECTOR  
CHANNEL\*\*\***

◆ This feature is applicable to the 70/35 Processor. It provides two selector channels and four input/output trunks.

**EMULATOR OPTIONS**

◆ Object code programs for the RCA 301 and 501 systems and IBM 1410 and 1401 (including 1440 and 1460) systems can be executed on the Model 70/35 and 70/45 systems through the optional Emulator features. The feature numbers and applicable processors are listed on the following page.

\* Only one feature (5015 or 5016) is permitted on a system.

\*\* Only one feature (5020, 5022 or 5024) is permitted on a system.

\*\*\* Only one feature (5030 or 5031) is permitted on a system.

|   |  |
|---|--|
| <b>Operational<br/>Characteristics</b>        | <p>◆ Using the facilities of the Model 70/35 and 70/45 Processors and associated peripheral devices, the Emulator features permit the running of RCA 301 and 501 and IBM 1400 series object-code programs on the 70/35-45 systems without modification or reprogramming.</p> <p>A 70/45 system provided with the facility for emulating one of the specified systems may be further enhanced to emulate any one of the remaining specified computers. (The 70/35 may only be enhanced with the 301 and 1401 Emulator feature.) However, not more than two Emulator features may be contained in a 70/35 or 70/45 system.</p> <p>While reprogramming of programs is not required, certain conditions must be considered before emulation is attempted. Programs to be emulated must have been written in accordance with normal programming standards of the subject computer, must not utilize or be affected by non-standard "RPQ" or "PQR" features installed in the subject computer, and must be emulated with comparable 70/35 or 70/45 equipment complement with equivalent standard or optional features as the subject computer. In addition, programs with time dependency coding must be carefully reviewed and modified where necessary.</p> <p>Emulated programs may be inefficient, inaccurate, or may not function unless they are compatible with timing factors for both the emulator system and the 70/35-45 input/output operations.</p> <p>Detailed functional descriptions and operating characteristics of these emulator features may be found in the specific Emulator Reference Manuals.</p> |
| <b>Feature 5005-35<br/>301 Emulator</b>       | ◆ This feature is applicable to the 70/35 Processor.   |
| <b>Feature 5005-45<br/>301 Emulator</b>       | ◆ This feature is applicable to the 70/45 Processor.   |
| <b>Feature 5006-35<br/>1401 Emulator</b>      | ◆ This feature is applicable to the 70/35 Processor.   |
| <b>Feature 5006-45<br/>1401 Emulator</b>      | ◆ This feature is applicable to the 70/45 Processor.   |
| <b>Feature 5007-45<br/>501 Emulator</b>       | ◆ This feature is applicable to the 70/45 Processor.   |
| <b>Feature 5026-45<br/>1410 Emulator</b>      | ◆ This feature is applicable to the 70/45 Processor.   |
| <b>Feature 5036-45<br/>301/501 Emulator</b>   | ◆ This feature is applicable to the 70/45 Processor.   |
| <b>Feature 5046-45<br/>1410/1401 Emulator</b> | ◆ This feature is applicable to the 70/45 Processor.   |



## APPENDICES

# APPENDIX A — SUMMARY

## Privileged

| Instruction        | Op <sub>(16)</sub> | Mnemonic | Format | Interrupt Action  | Condition Code   |  |
|--------------------|--------------------|----------|--------|---|--|--|
| Check Channel      | 9F                 | CKC      | SI     | 1. Privileged operation.  | 0 — I/O chan. avail.<br>1 — Interrupt pending in selector channel.<br>2 — Selector chan. busy or int. pending or multiplex chan. operating in burst mode.<br>3 — Inoperable. |  |
| Diagnose           | 83                 | DIG      | SI     | 1. Privileged operation.  | Unaltered.   |  |
| Halt Device        | 9E                 | HDV      | SI     | 1. Privileged operation.  | 0 — Not busy.<br>1 — Standard device byte stored in scratch-pad memory.<br>2 — Termination accepted.<br>3 — Inoperable.  |  |
| Idle               | 80                 | IDL      | SI     | 1. Privileged operation.  | Unchanged.   |  |
| Insert Storage Key | 09                 | ISK      | RR     | 1. Privileged operation.<br>2. Operation code trap (if feature not installed).<br>3. Address error. | Unchanged.   |  |
| Load Scratch Pad   | D8                 | LSP      | SS     | 1. Privileged operation.<br>2. Address error.   | Unchanged.   |  |
| Program Control    | 82                 | PC       | SI     | 1. Privileged operation.<br>2. Address error.   | CC of state being terminated is stored in P counter.<br>CC of state being initiated used to set CC indicators.   |  |
| Read Direct        | 85                 | RDD      | SI     | 1. Privileged operation.<br>2. Operation code trap (if feature not installed).<br>3. Address error. | Unchanged.   |  |
| Set Storage Key    | 08                 | SSK      | RR     | 1. Privileged operation.<br>2. Operation code trap (if feature not installed).<br>3. Address error. | Unchanged.   |  |
| Start Device       | 9C                 | SDV      | SI     | 1. Privileged operation.  | 0 — I/O operation started and channel proceeding.<br>1 — Status bits stored in scratch-pad.<br>2 — Busy or interrupt pending.<br>3 — Inoperable.                             |  |
| Store Scratch-Pad  | D0                 | SSP      | SS     | 1. Privileged operation.<br>2. Address error.   | Unchanged.   |  |
| Test Device        | 9D                 | TDV      | SI     | 1. Privileged operation.  | 0 — Available.<br>1 — Standard device byte stored in scratch-pad.<br>2 — Busy or interrupt pending.<br>3 — Inoperable.   |  |
| Write Direct       | 84                 | WRD      | SI     | 1. Privileged operation.<br>2. Operation code trap (if feature not installed).<br>3. Address error. | Unchanged.   |  |

# OF INSTRUCTIONS

## Instructions

|  | Timing ( $\mu$ sec)<br>(Average and Includes Staticizing) |  |                 | Page<br>Ref. |
|--|---|--|-----------------|--------------|
|  | 70/35   | 70/45  | 70/55           |              |
|  |   | Multiplexor = 5.52<br>Selector = 6.48                                    | 2.70            | 99           |
|  |   | 4.56   | 2.16            | 91           |
|  |   | Multiplexor = 10.32 + CRT<br>Burst = 5.52 + CRT<br>Selector = 6.00 + CRT | 7.14 + CRT      | 95           |
|  | 9.60  | 6.00   | 3.66            | 90           |
|  |   | 5.28   | 3.00            | 100          |
|  | 24.48 + 7.68R<br>(Note 3)                                 | 7.20 + 2.88R   | 3.60 + 0.96R    | 86           |
|  | 36.48<br>(Note 4)   | 7.44   | 3.66            | 88           |
|  | 8.64 + ED   | To be supplied.  | To be supplied. | 103          |
|  |   | 5.28   | 3.36            | 101          |
|  |   | Multiplexor = 33.36 + CRT<br>Selector = 27.60 + CRT                      | 14.46 + CRT     | 92           |
|  | 23.52 + 7.68R<br>(Note 3)                                 | 7.20 + 2.88R   | 3.60 + 1.20R    | 87           |
|  |   | Multiplexor = 8.40 + CRT<br>Selector = 8.88 + CRT                        | 7.14 + CRT      | 97           |
|  | 8.16  | To be supplied.  | To be supplied. | 102          |

**Legend:** CRT — channel response time (two microseconds average).  
R — number of registers specified.  
ED — external delay.

## SUMMARY OF

### Processor State

| Instruction      | Op <sub>(16)</sub> | Mnemonic | Format | Interrupt Action | Condition Code   |  |
|------------------|--------------------|----------|--------|------------------|--|--|
| Set Program Mask | 04                 | SPM      | RR     | None.            | CC set according to GR bits 2, 3 specified by R <sub>1</sub> . |  |
| Supervisor Call  | 0A                 | SVC      | RR     | None.            | Unchanged.   |  |

### Fixed-Point

|                    |    |     |    |   |  |  |
|--------------------|----|-----|----|---|--|--|
| Add Halfword       | 4A | AH  | RX | 1. Fixed-Point overflow.<br>2. Address error.           | 0 — Sum is zero.<br>1 — Sum is less than zero.<br>2 — Sum is greater than zero.<br>3 — Overflow.                                 |  |
| Add Logical        | 5E | AL  | RX | 1. Address error.                                       | 0 — Sum is zero & no carry.<br>1 — Sum is not zero & no carry.<br>2 — Sum is zero with carry.<br>3 — Sum is not zero with carry. |  |
|                    | 1E | ALR | RR |   |  |  |
| Add Word           | 5A | A   | RX | 1. Fixed-Point overflow.<br>2. Address error.           | 0 — Sum is zero.<br>1 — Sum is less than zero.<br>2 — Sum is greater than zero.<br>3 — Overflow.                                 |  |
|                    | 1A | AR  | RR |   |  |  |
| Compare Halfword   | 49 | CH  | RX | 1. Address error.                                       | 0 — Operands equal.<br>1 — First operand low.<br>2 — First operand high.<br>3 — Not used.  |  |
| Compare Word       | 59 | C   | RX | 1. Address error.                                       | 0 — Operands equal.<br>1 — First operand low.<br>2 — First operand high.<br>3 — Not used.  |  |
|                    | 19 | CR  | RR |   |  |  |
| Convert to Binary  | 4F | CVB | RX | 1. Address error.<br>2. Data error.<br>3. Divide error. | Unchanged.   |  |
| Convert to Decimal | 4E | CVD | RX | 1. Address error.                                       | Unchanged.   |  |
| Divide             | 5D | D   | RX | 1. Address error.                                       | Unchanged.   |  |
|                    | 1D | DR  | RR | 2. Divide error.  |  |  |
| Load Complement    | 13 | LCR | RR | 1. Fixed-Point overflow.                                | 0 — Result is zero.<br>1 — Result is less than zero.<br>2 — Result is greater than zero.<br>3 — Overflow.                        |  |
| Load Halfword      | 48 | LH  | RX | 1. Address error.                                       | Unchanged.   |  |
| Load Multiple      | 98 | LM  | RS | 1. Address error.                                       | Unchanged.   |  |



## INSTRUCTIONS (Cont'd)

## Control Instructions

|  | Timing ( $\mu$ sec)<br>(Average and Includes Staticizing) |       |       | Page<br>Ref. |
|--|---|-------|-------|--------------|
|  | 70/35   | 70/45 | 70/55 |              |
|  | 11.52   | 2.88  | 1.80  | 106          |
|  | 12.48   | 2.88  | 2.04  | 105          |

## Instructions

|  |   |                |               |     |
|--|---|----------------|---------------|-----|
|  | 20.48   | 7.92           | 3.98          | 119 |
|  | 19.68   | 8.40           | 2.58          | 120 |
|  | 13.44   | 4.80           | 1.92          |     |
|  | 19.00   | 8.88           | 2.58          | 118 |
|  | 13.76   | 5.28           | 1.92          |     |
|  | 19.04   | 7.44           | 2.58          | 125 |
|  | 19.04   | 8.40           | 2.58          | 124 |
|  | 12.80   | 4.80           | 1.92          |     |
|  | 43.20 + 18.24BY   | 91.20          | 5.34 to 26.34 | 129 |
|  | 60.96 + 3.36 bi ( $0 \leq bi \leq 16$ )<br>30.24 + 5.28 bi ( $17 \leq bi \leq 32$ ) | 68.88 to 91.92 | 5.70 to 23.82 | 130 |
|  | 211.00  | 94.89          | 19.86         | 128 |
|  | 204.00  | 90.81          | 19.20         |     |
|  | 11.84   | 5.28           | 1.92          | 114 |
|  | 16.32   | 7.92           | 2.58          | 112 |
|  | 7.68 + 7.2R   | 6.00 + 2.88R   | 2.10 + 0.84R  | 117 |

**Legend:** R — number of registers specified.  
 BY — number of significant bytes in a decimal number ( $1 \leq BY \leq 8$ ).  
 bi — number of significant bits of a binary number ( $0 \leq bi \leq 32$ ).

# SUMMARY OF Fixed-Point

| Instruction        | Op <sub>(16)</sub> | Mnemonic | Format | Interrupt Action                              | Condition Code  |  |
|--------------------|--------------------|----------|--------|---|---|--|
| Load Negative      | 11                 | LNR      | RR     | None.   | 0 — Result is zero.<br>1 — Result is less than zero.<br>2 — Not used.<br>3 — Not used.                    |  |
| Load Positive      | 10                 | LPR      | RR     | 1. Fixed-Point overflow.                      | 0 — Result is zero.<br>1 — Not used.<br>2 — Result greater than zero.<br>3 — Overflow.                    |  |
| Load and Test      | 12                 | LTR      | RR     | None.   | 0 — Result is zero.<br>1 — Result is less than zero.<br>2 — Result is greater than zero.<br>3 — Not used. |  |
| Load Word          | 58                 | L        | RX     | 1. Address error.                             | Unchanged.  |  |
|                    | 18                 | LR       | RR     |   |   |  |
| Multiply Halfword  | 4C                 | MH       | RX     | 1. Address error.                             | Unchanged.  |  |
| Multiply Word      | 5C                 | M        | RX     | 1. Address error.                             | Unchanged.  |  |
|                    | 1C                 | MR       | RR     |   |   |  |
| Shift Left Double  | 8F                 | SLDA     | RS     | 1. Fixed-Point overflow.<br>2. Address error. | 0 — Result is zero.<br>1 — Result is less than zero.<br>2 — Result is greater than zero.<br>3 — Overflow. |  |
| Shift Right Double | 8E                 | SRDA     | RS     | 1. Address error.                             | 0 — Result is zero.<br>1 — Result is less than zero.<br>2 — Result is greater than zero.<br>3 — Not used. |  |
| Shift Left Single  | 8B                 | SLA      | RS     | 1. Fixed-Point overflow.                      | 0 — Result is zero.<br>1 — Result is less than zero.<br>2 — Result is greater than zero.<br>3 — Overflow. |  |
| Shift Right Single | 8A                 | SRA      | RS     | None.   | 0 — Result is zero.<br>1 — Result is less than zero.<br>2 — Result is greater than zero.<br>3 — Not used. |  |

**INSTRUCTIONS (Cont'd)****Instructions (Cont'd)**

|  | Timing ( $\mu\text{sec}$ )<br>(Average and Includes Staticizing)                     |   |  | Page<br>Ref. |
|--|--|---|--|--------------|
|  | 70/35  | 70/45   | 70/55                                  |              |
|  | 12.80  | 6.24  | 1.92                                   | 116          |
|  | 12.32  | 6.24  | 1.92                                   | 115          |
|  | 11.36  | 5.28  | 1.98                                   | 113          |
|  | 14.40  | 8.88  | 2.46                                   | 111          |
|  | 8.16   | 2.88  | 1.98                                   |              |
|  | 72.00  | 35.40   | 12.28                                  | 127          |
|  | 131.00   | 65.64   | 12.78                                  | 126          |
|  | 125.00   | 62.52   | 12.12                                  |              |
|  | 52.16 + .96N + 7.68NU +<br>.96J (NL $\neq$ 0)<br>51.68 + .96N + 7.68NU<br>(NL = 0)   | Under 16 = 11.04 + 0.96 (N)<br>16 to 31 = 15.12 + 0.96 (N-16)<br>32 to 47 = 19.20 + 0.96 (N-32)<br>48 to 63 = 23.28 + 0.96 (N-48) | 2.10 + 0.72 (P + Q) +<br>0.72S (N)     | 136          |
|  | 56.96 + .96N + 10.56NU +<br>.96J (NL $\neq$ 0)<br>54.96 + .96N + 10.56NU<br>(NL = 0) | Under 16 = 9.36 + 0.96 (N)<br>16 to 31 = 12.48 + 0.96 (N-16)<br>32 to 47 = 15.60 + 0.96 (N-32)<br>48 to 63 = 18.72 + 0.96 (N-48)  | 2.10 + 0.72 (P + Q + M) +<br>0.72S (N) | 137          |
|  | 20.96 + .48I + .48N (N $\neq$ 0)<br>20.48 (N = 0)                                    | Under 16 = 10.08 + 0.48 (N)<br>16 to 31 = 13.20 + 0.48 (N-16)<br>32 to 47 = 16.32 + 0.48 (N-32)<br>48 to 63 = 19.44 + 0.48 (N-48) | 2.10 + 0.36 (P + Q) +<br>0.36S (N)     | 134          |
|  | 21.92 + .48J + .48N (N $\neq$ 0)<br>20.48 (N = 0)                                    | Under 16 = 8.16 + 0.48 (N)<br>16 to 31 = 10.32 + 0.48 (N-16)<br>32 to 47 = 12.48 + 0.48 (N-32)<br>48 to 63 = 12.48 + 0.48 (N-48)  | 2.10 + 0.36 (P + Q + M) +<br>0.36S (N) | 135          |

**Legend:**

- I — equals 1 when N is odd; equals 0 when N is even.
- J — equals 0 when N is odd; equals 1 when N is even.
- M — number of two-bit shifts.
- N — total number of bits shifted.
- P — number of four-bit shifts.
- Q — number of one-bit shifts.
- NL — lower 3 bits of N (Module 8 of N).
- NU — upper 3 bits of N (Module 8 count of N).
- S (N) — 1 if N = 0; S (N) = 0 if N  $\neq$  0.

## SUMMARY OF Fixed-Point

| Instruction       | Op <sub>(16)</sub> | Mnemonic | Format | Interrupt Action                              | Condition Code   |  |
|-------------------|--------------------|----------|--------|---|--|--|
| Store Halfword    | 40                 | STH      | RX     | 1. Address error.                             | Unchanged.   |  |
| Store Multiple    | 90                 | STM      | RS     | 1. Address error.                             | Unchanged.   |  |
| Store Word        | 50                 | ST       | RX     | 1. Address error.                             | Unchanged.   |  |
| Subtract Halfword | 4B                 | SH       | RX     | 1. Fixed-Point overflow.<br>2. Address error. | 0 — Diff. is zero.<br>1 — Diff. less than zero.<br>2 — Diff. greater than zero.<br>3 — Overflow.               |  |
| Subtract Logical  | 5F                 | SL       | RX     | 1. Address error.                             | 0 — Not used.<br>1 — Diff. not zero; no carry.<br>2 — Diff. zero with carry.<br>3 — Diff. not zero with carry. |  |
|                   | 1F                 | SLR      | RR     |   |  |  |
| Subtract Word     | 5B                 | S        | RX     | 1. Fixed-Point overflow.<br>2. Address error. | 0 — Diff. is zero.<br>1 — Diff. less than zero.<br>2 — Diff. greater than zero.<br>3 — Overflow.               |  |
|                   | 1B                 | SR       | RR     |   |  |  |

## Decimal Arithmetic

|                  |    |     |    |   |   |  |
|------------------|----|-----|----|---|---|--|
| Add Decimal      | FA | AP  | SS | 1. Address error.<br>2. Data error.<br>3. Decimal overflow.     | 0 — Sum is zero.<br>1 — Sum is less than zero.<br>2 — Sum is greater than zero.<br>3 — Overflow.                              |  |
| Compare Decimal  | F9 | CP  | SS | 1. Address error.<br>2. Data error.                             | 0 — Fields algeb. equal.<br>1 — 1st operand algeb. less than 2nd operand.<br>2 — 1st operand algeb. greater than 2nd operand. |  |
| Divide Decimal   | FD | DP  | SS | 1. Address error.<br>2. Data error.<br>3. Decimal divide error. | Unchanged.  |  |
| Move with Offset | F1 | MVO | SS | 1. Address error.   | Unchanged.  |  |
| Multiply Decimal | FC | MP  | SS | 1. Address error.<br>2. Data error.                             | Unchanged.  |  |

**INSTRUCTIONS (Cont'd)****Instructions (Cont'd)**

|  | Timing ( $\mu\text{sec}$ )<br>(Average and Includes Staticizing) |                |                | Page<br>Ref. |
|--|--|----------------|----------------|--------------|
|  | 70/35  | 70/45          | 70/55          |              |
|  | 11.52  | 5.04           | 4.38           | 132          |
|  | $7.68 + 7.2R$  | $6.00 + 2.88R$ | $2.10 + 1.20R$ | 133          |
|  | 17.76  | 7.44           | 2.70           | 131          |
|  | 20.96  | 7.92           | 2.58           | 122          |
|  | 20.16  | 8.40           | 2.58           | 123          |
|  | 13.92  | 4.80           | 1.92           |              |
|  | 20.48  | 8.88           | 2.58           | 121          |
|  | 14.24  | 5.28           | 1.92           |              |

**Instructions**

|  |   |   |  |     |
|--|---|---|--|-----|
|  | $39.36 + 2.76L_1 + [1.92 + 3.84 (L_2 - L_1)] Z$     | $15.36 + 1.8L_1 + 0.42L_2$<br>(Note 1)              | $5.40 + 1.92W_1 + 0.96W_2 + 0.48L_1$<br>(Note 1)                           | 142 |
|  | $35.52 + 2.76L_1 + [1.92 + 3.84 (L_2 - L_1)] Z$     | $16.80 + 1.08L_1 + 0.42L_2$<br>(Note 1)             | $5.40 + 0.96W_2 + 1.08W_1 + 0.48L_1$<br>(Note 1)                           | 145 |
|  | $13.44 + 26.4L_2 (L_1 - L_2) + 71.52L_1 - 75.36L_2$ | $26.33 + 36.71L_1 - 35.14L_2 + 5.40L_2 (L_1 - L_2)$ | $11.28 + 1.2W_1 + 0.36L_1 + 0.72S + 0.60W_2$                               | 147 |
|  | $18.24 + 3.36L_1 + 1.44L_2$                         | $11.04 + 1.92L_1 + 0.96L_2$                         | $4.92 + 1.80W_1 + 0.60W_2 + 0.72 (L_1 + L_2)$                              | 150 |
|  | $42.72 + 13.30L_2 (L_1 - L_2) + 17.28L_1 - 7.2L_2$  | $28.49 + 16.96L_1 - 14.35L_2 + 2.34L_2 (L_1 - L_2)$ | $8.88 + 1.20W_1 + 1.08W_2 + 5.16L_2 + 8.88S + 3.12SL_2 + 0.72 (L_1 - L_2)$ | 146 |

**Legend:**

- $L_1$  — number of bytes in first operand field.
- $L_2$  — number of bytes in second operand field.
- $R$  — number of registers specified.
- $S = (L_1 - L_2) \div 4$ . If result is a mixed number, next higher integer is used.
- $W_1$  — total number of words in first operand field including partial words.
- $W_2$  — total number of words in second operand field including partial words.
- $Z$  — equals 0 when  $L_2 < L_1$ ; equals 1 when  $L_2 > L_1$ .

## SUMMARY OF Decimal Arithmetic

| Instruction      | Op <sub>(16)</sub> | Mnemonic | Format | Interrupt Action  | Condition Code  |
|------------------|--------------------|----------|--------|---|---|
| Pack             | F2                 | PACK     | SS     | 1. Address error.   | Unchanged.  |
| Subtract Decimal | FB                 | SP       | SS     | 1. Address error.<br>2. Data error.<br>3. Decimal overflow. | 0 — Diff. is zero.<br>1 — Diff. is less than zero.<br>2 — Diff. is greater than zero.<br>3 — Overflow.    |
| Unpack           | F3                 | UNPK     | SS     | 1. Address error.   | Unchanged.  |
| Zero and Add     | F8                 | ZAP      | SS     | 1. Address error.<br>2. Data error.<br>3. Decimal overflow. | 0 — Result is zero.<br>1 — Result is less than zero.<br>2 — Result is greater than zero.<br>3 — Overflow. |

## Logical

|                 |    |     |    |                                     |   |
|-----------------|----|-----|----|-------------------------------------|---|
| And             | 54 | N   | RX | 1. Address error.                   | 0 — Result is zero.<br>1 — Result is not zero.<br>2 — Not used.<br>3 — Not used.  |
|                 | D4 | NC  | SS |                                     |   |
|                 | 94 | NI  | SI |                                     |   |
|                 | 14 | NR  | RR |                                     |   |
| Compare Logical | 55 | CL  | RX | 1. Address error.                   | 0 — Operands equal.<br>1 — 1st operand less than 2nd operand.<br>2 — 1st operand greater than 2nd operand.<br>3 — Not used.   |
|                 | D5 | CLC | SS |                                     |   |
|                 | 95 | CLI | SI |                                     |   |
|                 | 15 | CLR | RR |                                     |   |
| Edit            | DE | ED  | SS | 1. Address error.<br>2. Data error. | 0 — Indicates zero source field whether or not signif. is established.<br>1 — Non-zero result field with signif. established to indicate less than zero.<br>2 — Non-zero result field with no signif. established to indicate greater than zero.<br>3 — Not used. |

**INSTRUCTIONS (Cont'd)****Instructions (Cont'd)**

|  | Timing ( $\mu$ sec)<br>(Average and Includes Staticizing) |   |  | Page<br>Ref. |
|--|---|---|--|--------------|
|  | 70/35   | 70/45                                   | 70/55  |              |
|  | $12.00 + 2.4L_1 + 2.88L_2$                                | $8.88 + 1.92L_1 + 0.96L_2$              | $4.56 + 1.80W_1 + 0.60W_2 + 0.72L_1 + 0.36L_2$   | 148          |
|  | $39.36 + 2.76L_1 + [1.92 + 3.84(L_2 - L_1)] Z$            | $15.36 + 1.80L_1 + 0.42L_2$<br>(Note 1) | $5.40 + 0.96W_2 + 1.92W_1 + 0.48L_1$<br>(Note 1) | 143          |
|  | $18.72 + 3.84L_1 + .24L_2$                                | $9.90 + 0.96L_1 + 0.90L_2$              | $4.80 + 1.80W_1 + 0.60W_2 + 0.36L_1 + 0.72L_2$   | 149          |
|  | $39.36 + 3.12L_1 + [1.92 + 3.84(L_2 - L_1)] Z$            | $15.48 + 1.08L_1 + 0.42L_2$<br>(Note 1) | $6.96 + 0.96W_1 + 0.96W_2 + 0.48L_1$<br>(Note 1) | 144          |

**Instructions**

|  |                                  |   |  |     |
|--|----------------------------------|---|--|-----|
|  | 20.16                            | 8.40                                    | 2.58   | 158 |
|  | $15.48 + 3.94L$                  | $8.95 + 2.22L$                          | $3.84 + 1.80W_1 + 0.96W_2 + 0.48L$                     |     |
|  | 10.08                            | 6.96                                    | 3.18   |     |
|  | 13.92                            | 5.28                                    | 1.92   |     |
|  | 18.40                            | 8.40                                    | 2.58   | 157 |
|  | $12.96 + 3.36L$                  | $8.96 + 1.44B$<br>(Note 2)              | $3.24 + 0.96W_1 + 0.96W_2 + 0.48B$<br>(Note 2)         |     |
|  | 9.28                             | 6.0                                     | 2.46   |     |
|  | 12.16                            | 4.8                                     | 1.92   |     |
|  | $14.40 + 6.72L_1 - .48F + 2.88K$ | $10.56 + 3L_1 + 1.92L_2 - 0.12F - 0.6K$ | $3.72 + 1.80W_1 + 0.60W_2 + 0.36L_1 + 0.96L_2 + 0.36K$ | 167 |

**Legend:** B — total number of bytes processed. This condition occurs if instruction terminates before the L count is exhausted.  
 F — total number of field separating symbols in pattern field.  
 K — number of control characters in pattern field.  
 L — total number of bytes specified by L field.  
 $L_1$  — number of bytes in first operand field.  
 $L_2$  — number of bytes in second operand field.  
 $W_1$  — total number of words in first operand field including partial words.  
 $W_2$  — total number of words in second operand field including partial words.  
 Z — equals 0 when  $L_2 < L_1$ ; equals 1 when  $L_2 > L_1$ .

# SUMMARY OF Logic

| Instruction               | Op <sub>(16)</sub> | Mnemonic | Format | Interrupt Action                    | Condition Code  |  |
|---------------------------|--------------------|----------|--------|-------------------------------------|---|--|
| Edit and Mark             | DF                 | EDMK     | SS     | 1. Address error.<br>2. Data error. | 0 — Indicates zero source field whether or not signif. is established.<br>1 — Non-zero result field with signif. established to indicate less than zero.<br>2 — Non-zero result field with no signif. established to indicate greater than zero.<br>3 — Not used. |  |
| Exclusive Or              | 57                 | X        | RX     | 1. Address error.                   | 0 — Result is zero.<br>1 — Result is not zero.<br>2 — Not used.<br>3 — Not used.  |  |
|                           | D7                 | XC       | SS     |                                     |   |  |
|                           | 97                 | XI       | SI     |                                     |   |  |
|                           | 17                 | XR       | RR     |                                     |   |  |
| Insert Character          | 43                 | IC       | RX     | 1. Address error.                   | Unchanged.  |  |
| Load Address              | 41                 | LA       | RX     | None.                               | Unchanged.  |  |
| Move                      | D2                 | MVC      | SS     | 1. Address error.                   | Unchanged.  |  |
|                           | 92                 | MVI      | SI     |                                     |   |  |
| Move Numerics             | D1                 | MVN      | SS     | 1. Address error.                   | Unchanged.  |  |
| Move Zones                | D3                 | MVZ      | SS     | 1. Address error.                   | Unchanged.  |  |
| Or                        | 56                 | O        | RX     | 1. Address error.                   | 0 — Result is zero.<br>1 — Result is not zero.<br>2 — Not used.<br>3 — Not used.  |  |
|                           | D6                 | OC       | SS     |                                     |   |  |
|                           | 96                 | OI       | SI     |                                     |   |  |
|                           | 16                 | OR       | RR     |                                     |   |  |
| Shift Left Single Logical | 89                 | SLL      | RS     | None.                               | Unchanged.  |  |



## INSTRUCTIONS (Cont'd)

## Instructions (Cont'd)

|  | Timing ( $\mu$ sec)<br>(Average and Includes Staticizing)   |  |  | Page<br>Ref. |
|--|---|--|--|--------------|
|  | 70/35   | 70/45  | 70/55  |              |
|  | $18.76 + 6.72L_1 - .48F + 2.88K$                            | $13.44 + 3L_1 + 1.92L_2 - 0.12F - 0.6K$  | $6.00 + 1.80W_1 + 0.60W_2 + 0.36L_1 + 0.96L_2 + 0.36K$ | 170          |
|  | 20.64   | 8.40   | 2.58   | 160          |
|  | $15.48 + 3.94L$   | $8.95 + 2.22L$   | $3.84 + 1.80W_1 + 0.96W_2 + 0.48L$                     |              |
|  | 10.57   | 6.96   | 3.18   |              |
|  | 14.40   | 5.28   | 1.92   |              |
|  | 12.00   | 5.52   | 2.70   | 162          |
|  | 19.20   | 7.92   | 2.10   | 164          |
|  | $13.92 + 1.92L$   | $8.94 + 1.44L$   | $5.76 + 0.84W_1 + 0.96W_2 + 0.36L$                     | 154          |
|  | 8.64  | 5.04   | 3.18   |              |
|  | $10.56 + 4.8L$  | $9.90 + 2.22L$   | $3.84 + 1.80W_1 + 0.96W_2 + 0.36L$                     | 155          |
|  | $10.56 + 4.32L$   | $9.90 + 2.22L$   | $3.84 + 1.80W_1 + 0.96W_2 + 0.36L$                     | 156          |
|  | 20.16   | 8.40   | 2.58   | 159          |
|  | $15.48 + 3.94L$   | $8.95 + 2.22L$   | $3.84 + 1.80W_1 + 0.96W_2 + 0.48L$                     |              |
|  | 10.08   | 6.96   | 3.18   |              |
|  | 13.92   | 5.28   | 1.92   |              |
|  | $18.24 + .48J + .48N$ ( $N \neq 0$ )<br>$20.48$ ( $N = 0$ ) | Under 16 = $7.92 + 0.48$ ( $N$ )<br>16 to 31 = $11.04 + 0.48$ ( $N-16$ )<br>32 to 47 = $14.16 + 0.48$ ( $N-32$ )<br>48 to 63 = $17.28 + 0.48$ ( $N-48$ ) | $2.10 + 0.36$ ( $P + Q$ ) +<br>$0.36S$ ( $N$ )         | 172          |

**Legend:**

- F — total number of fields separating symbols in pattern field.
- K — number of control characters in pattern field.
- L — total number of bytes specified by L field.
- $L_1$  — number of bytes in first operand field.
- $L_2$  — number of bytes in second operand field.
- N — number of bits shifted.
- P — number of four-bit shifts.
- Q — number of one-bit shifts.
- $W_1$  — total number of words in first operand field including partial words.
- $W_2$  — total number of words in second operand field including partial words.
- S ( $N$ ) — 1 if  $N = 0$ ; S ( $N$ ) = 0 if  $N \neq 0$ .

# SUMMARY OF

## Logical

| Instruction                | Op <sub>(16)</sub> | Mnemonic | Format | Interrupt Action  | Condition Code   |  |
|----------------------------|--------------------|----------|--------|-------------------|--|--|
| Shift Right Single Logical | 88                 | SRL      | RS     | None.             | Unchanged.   |  |
| Shift Left Double Logical  | 8D                 | SLDL     | RS     | 1. Address Error. | Unchanged.   |  |
| Shift Right Double Logical | 8C                 | SRDL     | RS     | 1. Address Error. | Unchanged.   |  |
| Store Character            | 42                 | STC      | RX     | 1. Address Error. | Unchanged.   |  |
| Test Under Mask            | 91                 | TM       | SI     | 1. Address Error. | 0 — Selected bits all zero, or mask all zero.<br>1 — Selected bits mixed zero and one.<br>2 — Not used.<br>3 — Selected bits all one.      |  |
| Translate                  | DC                 | TR       | SS     | 1. Address Error. | Unchanged.   |  |
| Translate and Test         | DD                 | TRT      | SS     | 1. Address Error. | 0 — All accessed function bytes all zeros.<br>1 — Non-zero function byte encountered.<br>2 — Last function byte non-zero.<br>3 — Not used. |  |

## Branching

|                 |    |      |    |       |            |  |
|-----------------|----|------|----|-------|------------|--|
| Branch and Link | 45 | BAL  | RX | None. | Unchanged. |  |
|                 | 05 | BALR | RR |       |            |  |

**INSTRUCTIONS (Cont'd)****Instructions (Cont'd)**

|  | Timing ( $\mu$ sec)<br>(Average and Includes Staticizing)                            |  |  | Page<br>Ref. |
|--|--|--|--|--------------|
|  | 70/35  | 70/45  | 70/55  |              |
|  | 18.72 + .48J + .48N (N $\neq$ 0)<br>17.28 (N = 0)                                    | Under 16 = 8.88 + 0.48 (N)<br>16 to 31 = 11.04 + 0.48 (N-16)<br>32 to 47 = 13.20 + 0.48 (N-32)<br>48 to 63 = 13.20 + 0.48 (N-48) | 2.10 + 0.36 (P + Q + M) +<br>0.36S (N)         | 173          |
|  | 49.44 + .96N + 8.64NU +<br>.48J (NL $\neq$ 0)<br>48.48 + .96N + 8.64NU<br>(NL = 0)   | Under 16 = 7.68 + 0.96 (N)<br>16 to 31 = 11.76 + 0.96 (N-16)<br>32 to 47 = 15.84 + 0.96 (N-32)<br>48 to 63 = 19.92 + 0.96 (N-48) | 2.10 + 0.72 (P + Q) +<br>0.72S (N)             | 174          |
|  | 53.76 + .96N + 10.56NU +<br>.96J (NL $\neq$ 0)<br>51.36 + .96N + 10.56NU<br>(NL = 0) | Under 16 = 7.44 + 0.96 (N)<br>16 to 31 = 10.56 + 0.96 (N-16)<br>32 to 47 = 13.68 + 0.96 (N-32)<br>48 to 63 = 16.80 + 0.96 (N-48) | 2.10 + 0.72 (P + Q + M) +<br>0.72S (N)         | 175          |
|  | 11.52  | 5.04   | 3.18   | 163          |
|  | 9.28   | 6.48   | 2.82   | 161          |
|  | 11.04 + 4.8L   | 6.24 + 5.04L   | 3.24 + 1.20W <sub>1</sub> + 2.88L <sub>1</sub> | 165          |
|  | 1.92 + 4.8B (CC = 0)<br>19.30 + 4.8B (CC = 1 or 2)                                   | 11.04 + 4.08B  | 4.68 + 2.52B                                   | 166          |

**Instructions**

|  |                                     |                                   |                                   |     |
|--|-------------------------------------|-----------------------------------|-----------------------------------|-----|
|  | 17.28                               | 5.52                              | 2.70                              | 179 |
|  | Branch = 12.48<br>No Branch = 11.52 | Branch = 4.80<br>No Branch = 3.84 | Branch = 2.52<br>No Branch = 2.04 |     |

**Legend:**

- B — total number of bytes processed. This condition occurs if instruction terminates before L count is exhausted.
- L — total number of bytes specified by L field.
- L<sub>1</sub> — number of bytes in first operand field.
- M — number of two-bit shifts.
- N — number of bits shifted.
- P — number of four-bit shifts.
- Q — number of one-bit shifts.
- W<sub>1</sub> — total number of words in first operand field including partial words.
- CC — condition code.
- NL — lower 3 bits of N (Module 8 of N).
- NU — upper 3 bits of N (Module 8 count of N).
- S (N) — 1 if N = 0; S (N) = 0 if N  $\neq$  0.

## SUMMARY OF Branching

| Instruction                  | Op <sub>(16)</sub> | Mnemonic | Format | Interrupt Action  | Condition Code   |  |
|------------------------------|--------------------|----------|--------|-------------------|--|--|
| Branch on Condition          | 47                 | BC       | RX     | None.             | Unchanged.   |  |
|                              | 07                 | BCR      | RR     |                   |  |  |
| Branch on Count              | 46                 | BCT      | RX     | None.             | Unchanged.   |  |
|                              | 06                 | BCTR     | RR     |                   |  |  |
| Branch on Index High         | 86                 | BXH      | RS     | None.             | Unchanged.   |  |
| Branch on Index Low or Equal | 87                 | BXLE     | RS     | None.             | Unchanged.   |  |
| Execute                      | 44                 | EX       | RX     | 1. Address error. | May be set by instruction being modified and executed. |  |

## Floating-Point

|                          |    |     |    |  |   |  |
|--------------------------|----|-----|----|--|---|--|
| Add Normalized (Long)    | 6A | AD  | RX | 1. Address error.<br>2. Significance error.<br>3. Exponent overflow.<br>4. Exponent underflow. | 0 — Result mantissa zero.<br>1 — Result mantissa less than zero.<br>2 — Result mantissa greater than zero.<br>3 — Result exponent overflow. |  |
|                          | 2A | ADR | RR |  |   |  |
| Add Normalized (Short)   | 7A | AE  | RX |  |   |  |
|                          | 3A | AER | RR |  |   |  |
| Add Unnormalized (Long)  | 6E | AW  | RX | 1. Address error.<br>2. Significance error.<br>3. Exponent overflow.                           | 0 — Result mantissa zero.<br>1 — Result mantissa less than zero.<br>2 — Result mantissa greater than zero.<br>3 — Result exponent overflow. |  |
|                          | 2E | AWR | RR |  |   |  |
| Add Unnormalized (Short) | 7E | AU  | RX |  |   |  |
|                          | 3E | AUR | RR |  |   |  |
| Compare (Long)           | 69 | CD  | RX | 1. Address error.  | 0 — Operands equal.<br>1 — Operand specified by 1st address low.<br>2 — Operand specified by 1st address high.<br>3 — Not used.             |  |
|                          | 29 | CDR | RR |  |   |  |
| Compare (Short)          | 79 | CE  | RX |  |   |  |
|                          | 39 | CER | RR |  |   |  |
| Divide (Long)            | 6D | DD  | RX | 1. Address error.<br>2. Exponent overflow.<br>3. Exponent underflow.<br>4. Divide error.       | Unchanged.  |  |
|                          | 2D | DDR | RR |  |   |  |
| Divide (Short)           | 7D | DE  | RX |  |   |  |
|                          | 3D | DER | RR |  |   |  |

**INSTRUCTIONS (Cont'd)****Instructions (Cont'd)**

|  | Timing ( $\mu$ sec)<br>(Average and Includes Staticizing) |                                     |                                   | Page<br>Ref. |
|--|---|-------------------------------------|-----------------------------------|--------------|
|  | 70/35   | 70/45                               | 70/55                             |              |
|  | Branch = 10.56<br>No Branch = 9.60                        | Branch = 4.56<br>No Branch = 4.56   | Branch = 2.10<br>No Branch = 1.74 | 178          |
|  | Branch = 6.72<br>No Branch = 4.80                         | Branch = 3.84<br>No Branch = 3.36   | Branch = 1.98<br>No Branch = 1.62 |              |
|  | Branch = 17.76<br>No Branch = 16.32                       | Branch = 7.92<br>No Branch = 6.96   | Branch = 2.58<br>No Branch = 2.22 | 180          |
|  | Branch = 12.96<br>No Branch = 11.52 C = 12.00             | Branch = 5.76<br>No Branch = 5.28   | Branch = 2.40<br>No Branch = 1.92 |              |
|  | Branch = 24.48<br>No Branch = 23.04                       | Branch = 11.60<br>No Branch = 11.12 | Branch = 3.72<br>No Branch = 3.36 | 181          |
|  | Branch = 24.00<br>No Branch = 23.52                       | Branch = 11.60<br>No Branch = 11.60 | Branch = 3.72<br>No Branch = 3.36 | 182          |
|  | 18.24 + EX  | 6.96 + EX                           | 3.90 + EX                         | 183          |

**Arithmetic Instructions**

|  |         |        |       |     |
|--|---------|--------|-------|-----|
|  | 73.62   | 27.69  | 9.95  | 193 |
|  | 68.34   | 22.63  | 8.57  |     |
|  | 46.33   | 19.20  | 7.46  |     |
|  | 42.01   | 16.08  | 6.32  |     |
|  | 71.19   | 26.81  | 9.82  | 195 |
|  | 65.91   | 21.77  | 8.44  |     |
|  | 44.95   | 18.96  | 6.59  |     |
|  | 40.63   | 15.84  | 6.25  |     |
|  | 61.66   | 23.52  | 7.20  | 198 |
|  | 56.38   | 18.48  | 5.82  |     |
|  | 38.62   | 15.36  | 6.57  |     |
|  | 34.32   | 12.24  | 5.43  |     |
|  | 1239.86 | 280.27 | 75.29 | 202 |
|  | 1234.58 | 275.68 | 73.91 |     |
|  | 410.89  | 83.00  | 22.68 |     |
|  | 406.57  | 79.88  | 21.54 |     |

*Legend:* C — counting only is performed.  
EX — object instruction execution time.

## SUMMARY OF Floating-Point

| Instruction                 | Op <sub>(16)</sub> | Mnemonic | Format | Interrupt Action   | Condition Code  |  |
|-----------------------------|--------------------|----------|--------|--|---|--|
| Halve (Long)                | 24                 | HDR      | RR     | 1. Address error.  | Unchanged.  |  |
| Halve (Short)               | 34                 | HER      | RR     |  |   |  |
| Load Complement (Long)      | 23                 | LCDR     | RR     | 1. Address error.  | 0 — Result mantissa zero.<br>1 — Result mantissa less than zero.<br>2 — Result mantissa greater than zero.<br>3 — Not used.                 |  |
| Load Complement (Short)     | 33                 | LCER     | RR     |  |   |  |
| Load (Long)                 | 68                 | LD       | RX     | 1. Address error.  | Unchanged.  |  |
|                             | 28                 | LDR      | RR     |  |   |  |
| Load (Short)                | 78                 | LE       | RX     |  |   |  |
|                             | 38                 | LER      | RR     |  |   |  |
| Load Negative (Long)        | 21                 | LNDR     | RR     | 1. Address error.  | 0 — Result mantissa zero.<br>1 — Result mantissa less than zero.<br>2 — Not used.<br>3 — Not used.  |  |
| Load Negative (Short)       | 31                 | LNDR     | RR     |  |   |  |
| Load Positive (Long)        | 20                 | LPDR     | RR     | 1. Address error.  | 0 — Result mantissa zero.<br>1 — Not used.<br>2 — Result mantissa greater than zero.<br>3 — Not used.                                       |  |
| Load Positive (Short)       | 30                 | LPER     | RR     |  |   |  |
| Load and Test (Long)        | 22                 | LTDR     | RR     | 1. Address error.  | 0 — Result mantissa zero.<br>1 — Result mantissa less than zero.<br>2 — Result mantissa greater than zero.<br>3 — Not used.                 |  |
| Load and Test (Short)       | 32                 | LTER     | RR     |  |   |  |
| Multiply (Long)             | 6C                 | MD       | RX     | 1. Address error.<br>2. Exponent overflow.<br>3. Exponent underflow.                           | Unchanged.  |  |
|                             | 2C                 | MDR      | RR     |  |   |  |
| Multiply (Short)            | 7C                 | ME       | RX     |  |   |  |
|                             | 3C                 | MER      | RR     |  |   |  |
| Store (Long)                | 60                 | STD      | RX     | 1. Address error.  | Unchanged.  |  |
| Store (Short)               | 70                 | STE      | RX     |  |   |  |
| Subtract Normalized (Long)  | 6B                 | SD       | RX     | 1. Address error.<br>2. Significance error.<br>3. Exponent overflow.<br>4. Exponent underflow. | 0 — Result mantissa zero.<br>1 — Result mantissa less than zero.<br>2 — Result mantissa greater than zero.<br>3 — Result exponent overflow. |  |
|                             | 2B                 | SDR      | RR     |  |   |  |
| Subtract Normalized (Short) | 7B                 | SE       | RX     |  |   |  |
|                             | 3B                 | SER      | RR     |  |   |  |

## INSTRUCTIONS (Cont'd)

## Arithmetic Instructions (Cont'd)

|  | Timing ( $\mu$ sec)<br>(Average and Includes Staticizing) |        |       | Page<br>Ref. |
|--|---|--------|-------|--------------|
|  | 70/35   | 70/45  | 70/55 |              |
|  | 20.16   | 8.16   | 2.40  | 199          |
|  | 14.40   | 6.00   | 1.80  |              |
|  | 23.76   | 8.16   | 2.58  | 190          |
|  | 16.56   | 6.00   | 1.98  |              |
|  | 17.28   | 13.68  | 4.02  | 188          |
|  | 17.76   | 8.64   | 2.58  |              |
|  | 16.32   | 9.84   | 2.46  |              |
|  | 12.00   | 6.72   | 1.98  |              |
|  | 22.80   | 7.68   | 2.56  | 192          |
|  | 15.60   | 5.52   | 1.98  |              |
|  | 23.28   | 7.68   | 2.56  | 191          |
|  | 16.08   | 5.52   | 1.98  |              |
|  | 22.32   | 8.16   | 2.58  | 189          |
|  | 15.12   | 6.00   | 1.98  |              |
|  | 494.11  | 186.55 | 41.45 | 201          |
|  | 488.83  | 181.51 | 40.06 |              |
|  | 168.06  | 49.42  | 17.24 |              |
|  | 163.74  | 46.40  | 16.10 |              |
|  | 24.96   | 11.28  | 4.50  | 200          |
|  | 18.24   | 8.40   | 3.30  |              |
|  | 73.62   | 27.69  | 9.95  | 196          |
|  | 69.30   | 22.63  | 8.57  |              |
|  | 47.29   | 19.20  | 7.46  |              |
|  | 42.97   | 16.08  | 6.32  |              |

# SUMMARY OF Floating-Point

| Instruction                         | Op <sub>(16)</sub> | Mnemonic | Format | Interrupt Action   | Condition Code  |  |
|-------------------------------------|--------------------|----------|--------|--|---|--|
| Subtract<br>Unnormalized<br>(Long)  | 6F                 | SW       | RX     | 1. Address error.<br>2. Significance error.<br>3. Exponent overflow. | 0 — Result mantissa zero.<br>1 — Result mantissa less than zero.<br>2 — Result mantissa greater than zero.<br>3 — Result exponent overflow. |  |
|                                     | 2F                 | SWR      | RR     |  |   |  |
| Subtract<br>Unnormalized<br>(Short) | 7F                 | SU       | RX     |  |   |  |
|                                     | 3F                 | SUR      | RR     |  |   |  |



**INSTRUCTIONS (Cont'd)****Arithmetic Instructions (Cont'd)**

|  | Timing ( $\mu\text{sec}$ )<br>(Average and Includes Staticizing) |       |       | Page<br>Ref. |
|--|--|-------|-------|--------------|
|  | 70/35  | 70/45 | 70/55 |              |
|  | 72.15  | 26.81 | 9.82  | 197          |
|  | 66.87  | 21.77 | 8.44  |              |
|  | 41.76  | 18.96 | 6.59  |              |
|  | 41.59  | 15.84 | 6.25  |              |

- otes: 1. Time for  $L_1 > L_2$  and no End Around Carry. Additional time must be added if  $L_2 > L_1$  or End Around Carry.
2. If the two fields are equal  $B = L$  since all bytes must be examined. If the fields are unequal the instruction is terminated upon examining the first pair of unequal bytes. In this case, B is less than L.
3. Each 127 words stored or loaded requires an extra 0.96 microseconds to effect wrap around.
4. If Debug Mode,  $19.20 + EX$ .
5. Indexing after base addressing (RX format only) requires an additional 1.44 microseconds on the 70/35, no additional time on the 70/45 and .36 microseconds on the 70/55.

## APPENDIX B

### LIST OF PROGRAM INTERRUPTS

| Priority | Condition                        | State Initiated | Explanation  | Timing (If Interrupt Taken) |             |             |
|----------|----------------------------------|-----------------|--|-----------------------------|-------------|-------------|
|          |                                  |                 |  | 70/35                       | 70/45       | 70/55       |
| 1        | Power Failure                    | 4               | Power failure in processor or memory.  | 50.88                       | 11.64       | 7.32        |
| 2        | Machine Check                    | 4               | Parity error or equipment malfunction.   | 52.80                       | 11.64       | 7.32        |
| 3        | External Signal 1                | 3               | Signal received on one of the six external lines associated with the direct-control feature. | 54.72 (Note 1)              | 11.64       | 7.32        |
| 4        | External Signal 2                | 3               |  | 56.64 (Note 1)              | 11.64       | 7.32        |
| 5        | External Signal 3                | 3               |  | 58.56 (Note 1)              | 11.64       | 7.32        |
| 6        | External Signal 4                | 3               |  | 60.48 (Note 1)              | 11.64       | 7.32        |
| 7        | External Signal 5                | 3               |  | 62.40 (Note 1)              | 11.64       | 7.32        |
| 8        | External Signal 6                | 3               |  | 64.32 (Note 1)              | 11.64       | 7.32        |
| 9        | Not Specified                    |                 |  |                             |             |             |
| 10       | Selector 1 Terminate             | 3               | A device on the associated selector or multiplexor channel has terminated.                   | 68.16 (Note 2)              | 18.86 + CRT | 18.36 + CRT |
| 11       | Selector 2 Terminate             | 3               |  | 70.08 (Note 2)              | 18.86 + CRT | 18.36 + CRT |
| 12       | Selector 3 Terminate 70/45 70/55 | 3               |  |                             | 18.86 + CRT | 18.36 + CRT |
| 13       | Selector 4 Terminate 70/55       | 3               |  |                             |             | 18.36 + CRT |
| 14       | Selector 5 Terminate 70/55       | 3               |  |                             |             | 18.36 + CRT |
| 15       | Selector 6 Terminate 70/55       | 3               |  |                             |             | 18.36 + CRT |
| 16       | Multiplexor Terminate            | 3               |  | 79.68 (Note 2)              | 25.90 + CRT | 19.80 + CRT |
| 17       | Elapsed Time Clock               | 3               | Elapsed time count has expired.  | 54.72 (Notes 1 & 3)         | 13.08       | 6.60        |
| 18       | Console Request                  | 3               | Manual request for interrupt by the operator.  | 56.64 (Note 1)              | 13.08       | 6.60        |
| 19       | Not Specified                    |                 |  |                             |             |             |
| 20       | Not Specified                    |                 |  |                             |             |             |
| 21       | Supervisor Call                  | 3               | Result of execution of Supervisor Call instruction to utilize programmed routines.           | 67.40 (Note 1)              | 13.08       | 6.60        |
| 22       | Privileged Operation             | 3               | Privileged instruction attempted in non-privileged mode.                                     | 69.32 (Note 1)              | 13.08       | 6.60        |
| 23       | Op-Code Trap                     | 3               | Op Code attempted which is invalid for this model.   | 71.24 (Note 1)              | 13.08       | 6.60        |
| 24       | Address Error                    | 3               | Invalid address, specification, or memory protect violation.                                 | 73.16 (Note 1)              | 13.08       | 6.60        |

## APPENDIX B

### LIST OF PROGRAM INTERRUPTS (Cont'd)

| Priority              | Condition            | State Initiated | Explanation  | Timing (If Interrupt Taken) |       |       |
|-----------------------|----------------------|-----------------|--|-----------------------------|-------|-------|
|                       |                      |                 |  | 70/35                       | 70/45 | 70/55 |
| 25                    | Data Error           | 3               | Sign of operand incorrect in decimal arithmetic and editing, or incorrect field overlap. | 75.08 (Note 1)              | 13.08 | 6.24  |
| 26                    | Exponent Overflow    | 3               | Result characteristic of floating-point operation is greater than 127.                   | 77.00 (Note 1)              | 13.08 | 6.24  |
| 27                    | Divide Error         | 3               | Rules pertaining to Divide instruction have been violated.                               | 78.92 (Note 1)              | 13.08 | 6.24  |
| 28                    | Significance Error   | 3               | Result of floating-point or subtract has zero fraction.                                  | 80.84 (Note 1)              | 13.08 | 6.24  |
| 29                    | Exponent Underflow   | 3               | Result characteristic of floating-point operation is less than zero.                     | 82.76 (Note 1)              | 13.08 | 6.24  |
| 30                    | Decimal Overflow     | 3               | Result field is too small to contain the result of a decimal operation.                  | 84.68 (Note 1)              | 13.08 | 6.24  |
| 31                    | Fixed-Point Overflow | 3               | High-order carry or high-order significant bits lost in fixed-point operation.           | 86.60 (Note 1)              | 13.08 | 6.24  |
| 32                    | Test Mode            | 3               | Allows program control over processor during program testing.                            | 94.24 (Note 1)              | 13.08 | 6.24  |
| Priorities 1 thru 16  |                      |                 |  | 14.40                       | 5.76  | 2.04  |
| Priorities 17 thru 32 |                      |                 |  | 18.72                       | 5.76  | 2.04  |

Note 1. Entry to Interrupt processing is delayed until the end of the instruction currently being executed.

Note 2. Entry to Interrupt processing is delayed 29.76 microseconds plus the time required to reach the end of the instruction currently being executed.

Note 3. When a timer update request exists, add 6.72 microseconds. When a timer update request exists and Timer overflow occurs as a result of the update, add 7.68 microseconds.

# APPENDIX C

## INPUT/ OUTPUT SERVICE REQUEST

| Operation   | Timing Per Byte (microseconds) |        |               |
|---|--------------------------------|--------|---------------|
|   | 70/35                          | 70/45  | 70/55         |
| <i>Selector Channel</i>                           |                                |        |               |
| a. Normal Service                                 | Note 3                         | 2.40   |               |
| Scratch-Pad Read and Write                        |                                |        | 1.20          |
| Main Memory Read or Write (normal)                |                                |        | 1.56          |
| Less than 4 byte data move Read or Write (normal) |                                |        | 1.68          |
| b. Data chaining with no Transfer In Channel      | Note 3                         | 9.60   | 1.92 (Note 1) |
| c. Data chaining with Transfer In Channel         | Note 3                         | 13.92  | 2.04 (Note 1) |
| d. End Service                                    | Note 3                         | Note 3 |               |
| Normal  |                                |        | 2.40          |
| Data Chaining, Command Chaining                   |                                |        | 4.32          |
| (1) For Status Modifier, add                      |                                |        | .96           |
| (2) For each Transfer In Channel, add             |                                |        | 2.04          |
| (3) For Incomplete Read (Note 2), add             |                                |        | .96           |
| <i>Multiplexor Channel</i>                        |                                |        |               |
| a. Normal Service                                 | Note 3                         | 14.40  | 4.80          |
| b. Data Chaining with no Transfer In Channel      | Note 3                         | 27.36  |               |
| c. Data Chaining with Transfer In Channel         | Note 3                         | 31.68  |               |
| d. Burst/Catch-up (per byte, after first byte)    | Note 3                         |        | 1.68          |
| e. End Service                                    | Note 3                         | Note 3 |               |
| No chaining, no burst mode                        |                                |        | 4.68          |
| Data chaining, burst mode                         |                                |        | 7.68          |
| Data chaining, no burst mode                      |                                |        | 9.24          |
| Command chaining, burst mode                      |                                |        | 7.80          |
| Command chaining, no burst mode                   |                                |        | 8.76          |
| (1) For Status Modifier, add                      |                                |        | .48           |
| (2) For each Transfer In Channel, add             |                                |        | 1.80          |

Note 1. Plus any one of the times listed in item a.

Note 2. If a Read terminates while characters are still contained in the Scratch-Pad Assembly Word, a special path must be taken to move these characters to Main Memory when END is received.

Note 3. To be supplied.



## APPENDIX E

### AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE (ASCII) (Extended to 8 Bits)

|      | ← 4321 → |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|------|----------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 76X5 | 0000     | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
| 0000 | NUL      | SOH  | STX  | ETX  | EOT  | ENQ  | ACK  | BEL  | BS   | HT   | LF   | VT   | FF   | CR   | SO   | SI   |
| 0001 | DLE      | DC1  | DC2  | DC3  | DC4  | NAK  | SYN  | ETB  | CAN  | EM   | SS   | ESC  | FS   | GS   | RS   | US   |
| 0010 |          |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| 0011 |          |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| 0100 | SP       | !    | "    | #    | \$   | %    | &    | '    | (    | )    | *    | +    | ,    | -    | .    | /    |
| 0101 | 0        | 1    | 2    | 3    | 4    | 5    | 6    | 7    | 8    | 9    | :    | ;    | <    | =    | >    | ?    |
| 0110 |          |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| 0111 |          |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| 1000 |          |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| 1001 |          |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| 1010 | `        | A    | B    | C    | D    | E    | F    | G    | H    | I    | J    | K    | L    | M    | N    | O    |
| 1011 | P        | Q    | R    | S    | T    | U    | V    | W    | X    | Y    | Z    | [    | ~    | ]    | ^    | _    |
| 1100 |          |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| 1101 |          |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| 1110 | @        | a    | b    | c    | d    | e    | f    | g    | h    | i    | j    | k    | l    | m    | n    | o    |
| 1111 | p        | q    | r    | s    | t    | u    | v    | w    | x    | y    | z    | {    | ␣    | }    |      | DEL  |

Bit Positions: 7 6 X 5 4 3 2 1

Significance: 2<sup>7</sup> 2<sup>6</sup> 2<sup>5</sup> 2<sup>4</sup> 2<sup>3</sup> 2<sup>2</sup> 2<sup>1</sup> 2<sup>0</sup>

#### Control Characters:

NUL — Null  
 SOH — Start of Heading (CC)  
 STX — Start of Text (CC)  
 ETX — End of Text (CC)  
 EOT — End of Transmission (CC)  
 ENQ — Enquiry (CC)  
 ACK — Acknowledge (CC)  
 BEL — Bell (audible or attention signal)  
 BS — Backspace (FE)  
 HT — Horizontal Tabulation  
      (punch card skip) (FE)  
 LF — Line Feed (FE)  
 VT — Vertical Tabulation (FE)  
 FF — Form Feed (FE)  
 CR — Carriage Return (FE)  
 SO — Shift Out  
 SI — Shift In  
 DLE — Data Link Escape (CC)  
 DC1 — Device Control 1  
 DC2 — Device Control 2  
 DC3 — Device Control 3

DC4 — Device Control 4 (stop)  
 NAK — Negative Acknowledge (CC)  
 SYN — Synchronous Idle (CC)  
 ETB — End of Transmission Block (CC)  
 CAN — Cancel  
 EM — End of Medium  
 SS — Start of Special Sequence  
 ESC — Escape  
 FS — File Separator (IS)  
 GS — Group Separator (IS)  
 RS — Record Separator (IS)  
 US — Unit Separator (IS)  
 DEL — Delete

SP — Space (normally non-printing)

(CC) — Communication Control  
 (FE) — Format Effector  
 (IS) — Information Separator

## APPENDIX F

### CHARACTER CODES

| Decimal | Hexadecimal | EBCDIC    | Character Set<br>Punch<br>Combination | Printer<br>Graphics |
|---------|-------------|-----------|---------------------------------------|---------------------|
| 0       | 00          | 0000 0000 | 12,0,9,8,1                            |                     |
| 1       | 01          | 0000 0001 | 12,9,1                                |                     |
| 2       | 02          | 0000 0010 | 12,9,2                                |                     |
| 3       | 03          | 0000 0011 | 12,9,3                                |                     |
| 4       | 04          | 0000 0100 | 12,9,4                                |                     |
| 5       | 05          | 0000 0101 | 12,9,5                                |                     |
| 6       | 06          | 0000 0110 | 12,9,6                                |                     |
| 7       | 07          | 0000 0111 | 12,9,7                                |                     |
| 8       | 08          | 0000 1000 | 12,9,8                                |                     |
| 9       | 09          | 0000 1001 | 12,9,8,1                              |                     |
| 10      | 0A          | 0000 1010 | 12,9,8,2                              |                     |
| 11      | 0B          | 0000 1011 | 12,9,8,3                              |                     |
| 12      | 0C          | 0000 1100 | 12,9,8,4                              |                     |
| 13      | 0D          | 0000 1101 | 12,9,8,5                              |                     |
| 14      | 0E          | 0000 1110 | 12,9,8,6                              |                     |
| 15      | 0F          | 0000 1111 | 12,9,8,7                              |                     |
| 16      | 10          | 0001 0000 | 12,11,9,8,1                           |                     |
| 17      | 11          | 0001 0001 | 11,9,1                                |                     |
| 18      | 12          | 0001 0010 | 11,9,2                                |                     |
| 19      | 13          | 0001 0011 | 11,9,3                                |                     |
| 20      | 14          | 0001 0100 | 11,9,4                                |                     |
| 21      | 15          | 0001 0101 | 11,9,5                                |                     |
| 22      | 16          | 0001 0110 | 11,9,6                                |                     |
| 23      | 17          | 0001 0111 | 11,9,7                                |                     |
| 24      | 18          | 0001 1000 | 11,9,8                                |                     |
| 25      | 19          | 0001 1001 | 11,9,8,1                              |                     |
| 26      | 1A          | 0001 1010 | 11,9,8,2                              |                     |
| 27      | 1B          | 0001 1011 | 11,9,8,3                              |                     |
| 28      | 1C          | 0001 1100 | 11,9,8,4                              |                     |
| 29      | 1D          | 0001 1101 | 11,9,8,5                              |                     |
| 30      | 1E          | 0001 1110 | 11,9,8,6                              |                     |
| 31      | 1F          | 0001 1111 | 11,9,8,7                              |                     |
| 32      | 20          | 0010 0000 | 11,0,9,8,1                            |                     |
| 33      | 21          | 0010 0001 | 0,9,1                                 |                     |
| 34      | 22          | 0010 0010 | 0,9,2                                 |                     |
| 35      | 23          | 0010 0011 | 0,9,3                                 |                     |
| 36      | 24          | 0010 0100 | 0,9,4                                 |                     |
| 37      | 25          | 0010 0101 | 0,9,5                                 |                     |
| 38      | 26          | 0010 0110 | 0,9,6                                 |                     |
| 39      | 27          | 0010 0111 | 0,9,7                                 |                     |
| 40      | 28          | 0010 1000 | 0,9,8                                 |                     |
| 41      | 29          | 0010 1001 | 0,9,8,1                               |                     |
| 42      | 2A          | 0010 1010 | 0,9,8,2                               |                     |
| 43      | 2B          | 0010 1011 | 0,9,8,3                               |                     |
| 44      | 2C          | 0010 1100 | 0,9,8,4                               |                     |
| 45      | 2D          | 0010 1101 | 0,9,8,5                               |                     |
| 46      | 2E          | 0010 1110 | 0,9,8,6                               |                     |
| 47      | 2F          | 0010 1111 | 0,9,8,7                               |                     |
| 48      | 30          | 0011 0000 | 12,11,0,9,8,1                         |                     |
| 49      | 31          | 0011 0001 | 9,1                                   |                     |
| 50      | 32          | 0011 0010 | 9,2                                   |                     |
| 51      | 33          | 0011 0011 | 9,3                                   |                     |
| 52      | 34          | 0011 0100 | 9,4                                   |                     |
| 53      | 35          | 0011 0101 | 9,5                                   |                     |
| 54      | 36          | 0011 0110 | 9,6                                   |                     |

## CHARACTER CODES (Cont.)

| Decimal | Hexadecimal | EBCDIC    | Character Set<br>Punch<br>Combination | Printer<br>Graphics  |
|---------|-------------|-----------|---------------------------------------|--|
| 55      | 37          | 0011 0111 | 9,7                                   | Space  |
| 56      | 38          | 0011 1000 | 9,8                                   |  |
| 57      | 39          | 0011 1001 | 9,8,1                                 |  |
| 58      | 3A          | 0011 1010 | 9,8,2                                 |  |
| 59      | 3B          | 0011 1011 | 9,8,3                                 |  |
| 60      | 3C          | 0011 1100 | 9,8,4                                 |  |
| 61      | 3D          | 0011 1101 | 9,8,5                                 |  |
| 62      | 3E          | 0011 1110 | 9,8,6                                 |  |
| 63      | 3F          | 0011 1111 | 9,8,7                                 |  |
| 64      | 40          | 0100 0000 |                                       |  |
| 65      | 41          | 0100 0001 | 12,0,9,1                              | ¢ (cents)<br>· (period)<br>< (less than)<br>( (open parenthesis)<br>+ (plus)<br>  (vertical)<br>& (ampersand)                                |
| 66      | 42          | 0100 0010 | 12,0,9,2                              |  |
| 67      | 43          | 0100 0011 | 12,0,9,3                              |  |
| 68      | 44          | 0100 0100 | 12,0,9,4                              |  |
| 69      | 45          | 0100 0101 | 12,0,9,5                              |  |
| 70      | 46          | 0100 0110 | 12,0,9,6                              |  |
| 71      | 47          | 0100 0111 | 12,0,9,7                              |  |
| 72      | 48          | 0100 1000 | 12,0,9,8                              |  |
| 73      | 49          | 0100 1001 | 12,8,1                                |  |
| 74      | 4A          | 0100 1010 | 12,8,2                                |  |
| 75      | 4B          | 0100 1011 | 12,8,3                                | ! (exclamation)<br>\$ (dollar sign)<br>* (asterisk)<br>) (close parenthesis)<br>; (semicolon)<br>¬ (logical NOT)<br>- (minus)<br>/ (virgule) |
| 76      | 4C          | 0100 1100 | 12,8,4                                |  |
| 77      | 4D          | 0100 1101 | 12,8,5                                |  |
| 78      | 4E          | 0100 1110 | 12,8,6                                |  |
| 79      | 4F          | 0100 1111 | 12,8,7                                |  |
| 80      | 50          | 0101 0000 | 12                                    |  |
| 81      | 51          | 0101 0001 | 12,11,9,1                             |  |
| 82      | 52          | 0101 0010 | 12,11,9,2                             |  |
| 83      | 53          | 0101 0011 | 12,11,9,3                             |  |
| 84      | 54          | 0101 0100 | 12,11,9,4                             |  |
| 85      | 55          | 0101 0101 | 12,11,9,5                             | ^ (logical AND)<br>, (comma)<br>% (percent)<br>(underline)   |
| 86      | 56          | 0101 0110 | 12,11,9,6                             |  |
| 87      | 57          | 0101 0111 | 12,11,9,7                             |  |
| 88      | 58          | 0101 1000 | 12,11,9,8                             |  |
| 89      | 59          | 0101 1001 | 11,8,1                                |  |
| 90      | 5A          | 0101 1010 | 11,8,2                                |  |
| 91      | 5B          | 0101 1011 | 11,8,3                                |  |
| 92      | 5C          | 0101 1100 | 11,8,4                                |  |
| 93      | 5D          | 0101 1101 | 11,8,5                                |  |
| 94      | 5E          | 0101 1110 | 11,8,6                                |  |
| 95      | 5F          | 0101 1111 | 11,8,7                                |  |
| 96      | 60          | 0110 0000 | 11                                    |  |
| 97      | 61          | 0110 0001 | 0,1                                   |  |
| 98      | 62          | 0110 0010 | 11,0,9,2                              |  |
| 99      | 63          | 0110 0011 | 11,0,9,3                              |  |
| 100     | 64          | 0110 0100 | 11,0,9,4                              |  |
| 101     | 65          | 0110 0101 | 11,0,9,5                              |  |
| 102     | 66          | 0110 0110 | 11,0,9,6                              |  |
| 103     | 67          | 0110 0111 | 11,0,9,7                              |  |
| 104     | 68          | 0110 1000 | 11,0,9,8                              |  |
| 105     | 69          | 0110 1001 | 0,8,1                                 |  |
| 106     | 6A          | 0110 1010 | 12,11                                 |  |
| 107     | 6B          | 0110 1011 | 0,8,3                                 |  |
| 108     | 6C          | 0110 1100 | 0,8,4                                 |  |
| 109     | 6D          | 0110 1101 | 0,8,5                                 |  |



## CHARACTER CODES (Cont.)

| Decimal | Hexadecimal | EBCDIC    | Character Set<br>Punch<br>Combination | Printer<br>Graphics |
|---------|-------------|-----------|---------------------------------------|---------------------|
| 110     | 6E          | 0110 1110 | 0,8,6                                 | > (greater than)    |
| 111     | 6F          | 0110 1111 | 0,8,7                                 | ? (question mark)   |
| 112     | 70          | 0111 0000 | 12,11,0                               |                     |
| 113     | 71          | 0111 0001 | 12,11,0,9,1                           |                     |
| 114     | 72          | 0111 0010 | 12,11,0,9,2                           |                     |
| 115     | 73          | 0111 0011 | 12,11,0,9,3                           |                     |
| 116     | 74          | 0111 0100 | 12,11,0,9,4                           |                     |
| 117     | 75          | 0111 0101 | 12,11,0,9,5                           |                     |
| 118     | 76          | 0111 0110 | 12,11,0,9,6                           |                     |
| 119     | 77          | 0111 0111 | 12,11,0,9,7                           |                     |
| 120     | 78          | 0111 1000 | 12,11,0,9,8                           |                     |
| 121     | 79          | 0111 1001 | 8,1                                   |                     |
| 122     | 7A          | 0111 1010 | 8,2                                   | : (colon)           |
| 123     | 7B          | 0111 1011 | 8,3                                   | # (number sign)     |
| 124     | 7C          | 0111 1100 | 8,4                                   | @ (at the rate of)  |
| 125     | 7D          | 0111 1101 | 8,5                                   | ' (apostrophe)      |
| 126     | 7E          | 0111 1110 | 8,6                                   | = (equals)          |
| 127     | 7F          | 0111 1111 | 8,7                                   | " (quote)           |
| 128     | 80          | 1000 0000 | 12,0,8,1                              |                     |
| 129     | 81          | 1000 0001 | 12,0,1                                |                     |
| 130     | 82          | 1000 0010 | 12,0,2                                |                     |
| 131     | 83          | 1000 0011 | 12,0,3                                |                     |
| 132     | 84          | 1000 0100 | 12,0,4                                |                     |
| 133     | 85          | 1000 0101 | 12,0,5                                |                     |
| 134     | 86          | 1000 0110 | 12,0,6                                |                     |
| 135     | 87          | 1000 0111 | 12,0,7                                |                     |
| 136     | 88          | 1000 1000 | 12,0,8                                |                     |
| 137     | 89          | 1000 1001 | 12,0,9                                |                     |
| 138     | 8A          | 1000 1010 | 12,0,8,2                              |                     |
| 139     | 8B          | 1000 1011 | 12,0,8,3                              |                     |
| 140     | 8C          | 1000 1100 | 12,0,8,4                              |                     |
| 141     | 8D          | 1000 1101 | 12,0,8,5                              |                     |
| 142     | 8E          | 1000 1110 | 12,0,8,6                              |                     |
| 143     | 8F          | 1000 1111 | 12,0,8,7                              |                     |
| 144     | 90          | 1001 0000 | 12,11,8,1                             |                     |
| 145     | 91          | 1001 0001 | 12,11,1                               |                     |
| 146     | 92          | 1001 0010 | 12,11,2                               |                     |
| 147     | 93          | 1001 0011 | 12,11,3                               |                     |
| 148     | 94          | 1001 0100 | 12,11,4                               |                     |
| 149     | 95          | 1001 0101 | 12,11,5                               |                     |
| 150     | 96          | 1001 0110 | 12,11,6                               |                     |
| 151     | 97          | 1001 0111 | 12,11,7                               |                     |
| 152     | 98          | 1001 1000 | 12,11,8                               |                     |
| 153     | 99          | 1001 1001 | 12,11,9                               |                     |
| 154     | 9A          | 1001 1010 | 12,11,8,2                             |                     |
| 155     | 9B          | 1001 1011 | 12,11,8,3                             |                     |
| 156     | 9C          | 1001 1100 | 12,11,8,4                             |                     |
| 157     | 9D          | 1001 1101 | 12,11,8,5                             |                     |
| 158     | 9E          | 1001 1110 | 12,11,8,6                             |                     |
| 159     | 9F          | 1001 1111 | 12,11,8,7                             |                     |
| 160     | A0          | 1010 0000 | 11,0,8,1                              |                     |
| 161     | A1          | 1010 0001 | 11,0,1                                |                     |
| 162     | A2          | 1010 0010 | 11,0,2                                |                     |
| 163     | A3          | 1010 0011 | 11,0,3                                |                     |
| 164     | A4          | 1010 0100 | 11,0,4                                |                     |

## CHARACTER CODES (Cont.)

| Decimal | Hexadecimal | EBCDIC    | Character Set<br>Punch<br>Combination | Printer<br>Graphics |
|---------|-------------|-----------|---------------------------------------|---------------------|
| 165     | A5          | 1010 0101 | 11,0,5                                |                     |
| 166     | A6          | 1010 0110 | 11,0,6                                |                     |
| 167     | A7          | 1010 0111 | 11,0,7                                |                     |
| 168     | A8          | 1010 1000 | 11,0,8                                |                     |
| 169     | A9          | 1010 1001 | 11,0,9                                |                     |
| 170     | AA          | 1010 1010 | 11,0,8,2                              |                     |
| 171     | AB          | 1010 1011 | 11,0,8,3                              |                     |
| 172     | AC          | 1010 1100 | 11,0,8,4                              |                     |
| 173     | AD          | 1010 1101 | 11,0,8,5                              |                     |
| 174     | AE          | 1010 1110 | 11,0,8,6                              |                     |
| 175     | AF          | 1010 1111 | 11,0,8,7                              |                     |
| 176     | B0          | 1011 0000 | 12,11,0,8,1                           |                     |
| 177     | B1          | 1011 0001 | 12,11,0,1                             |                     |
| 178     | B2          | 1011 0010 | 12,11,0,2                             |                     |
| 179     | B3          | 1011 0011 | 12,11,0,3                             |                     |
| 180     | B4          | 1011 0100 | 12,11,0,4                             |                     |
| 181     | B5          | 1011 0101 | 12,11,0,5                             |                     |
| 182     | B6          | 1011 0110 | 12,11,0,6                             |                     |
| 183     | B7          | 1011 0111 | 12,11,0,7                             |                     |
| 184     | B8          | 1011 1000 | 12,11,0,8                             |                     |
| 185     | B9          | 1011 1001 | 12,11,0,9                             |                     |
| 186     | BA          | 1011 1010 | 12,11,0,8,2                           |                     |
| 187     | BB          | 1011 1011 | 12,11,0,8,3                           |                     |
| 188     | BC          | 1011 1100 | 12,11,0,8,4                           |                     |
| 189     | BD          | 1011 1101 | 12,11,0,8,5                           |                     |
| 190     | BE          | 1011 1110 | 12,11,0,8,6                           |                     |
| 191     | BF          | 1011 1111 | 12,11,0,8,7                           |                     |
| 192     | C0          | 1100 0000 | 12,0                                  |                     |
| 193     | C1          | 1100 0001 | 12,1                                  | A                   |
| 194     | C2          | 1100 0010 | 12,2                                  | B                   |
| 195     | C3          | 1100 0011 | 12,3                                  | C                   |
| 196     | C4          | 1100 0100 | 12,4                                  | D                   |
| 197     | C5          | 1100 0101 | 12,5                                  | E                   |
| 198     | C6          | 1100 0110 | 12,6                                  | F                   |
| 199     | C7          | 1100 0111 | 12,7                                  | G                   |
| 200     | C8          | 1100 1000 | 12,8                                  | H                   |
| 201     | C9          | 1100 1001 | 12,9                                  | I                   |
| 202     | CA          | 1100 1010 | 12,0,9,8,2                            |                     |
| 203     | CB          | 1100 1011 | 12,0,9,8,3                            |                     |
| 204     | CC          | 1100 1100 | 12,0,9,8,4                            |                     |
| 205     | CD          | 1100 1101 | 12,0,9,8,5                            |                     |
| 206     | CE          | 1100 1110 | 12,0,9,8,6                            |                     |
| 207     | CF          | 1100 1111 | 12,0,9,8,7                            |                     |
| 208     | D0          | 1101 0000 | 11,0                                  |                     |
| 209     | D1          | 1101 0001 | 11,1                                  | J                   |
| 210     | D2          | 1101 0010 | 11,2                                  | K                   |
| 211     | D3          | 1101 0011 | 11,3                                  | L                   |
| 212     | D4          | 1101 0100 | 11,4                                  | M                   |
| 213     | D5          | 1101 0101 | 11,5                                  | N                   |
| 214     | D6          | 1101 0110 | 11,6                                  | O                   |
| 215     | D7          | 1101 0111 | 11,7                                  | P                   |
| 216     | D8          | 1101 1000 | 11,8                                  | Q                   |
| 217     | D9          | 1101 1001 | 11,9                                  | R                   |
| 218     | DA          | 1101 1010 | 12,11,9,8,2                           |                     |
| 219     | DB          | 1101 1011 | 12,11,9,8,3                           |                     |

## CHARACTER CODES (Cont.)

| Decimal | Hexadecimal | EBCDIC    | Character Set<br>Punch<br>Combination | Printer<br>Graphics   |
|---------|-------------|-----------|---------------------------------------|---|
| 220     | DC          | 1101 1100 | 12,11,9,8,4                           | Blank   |
| 221     | DD          | 1101 1101 | 12,11,9,8,5                           |   |
| 222     | DE          | 1101 1110 | 12,11,9,8,6                           |   |
| 223     | DF          | 1101 1111 | 12,11,9,8,7                           |   |
| 224     | E0          | 1110 0000 | 0,8,2                                 |   |
| 225     | E1          | 1110 0001 | 11,0,9,1                              |   |
| 226     | E2          | 1110 0010 | 0,2                                   |   |
| 227     | E3          | 1110 0011 | 0,3                                   |   |
| 228     | E4          | 1110 0100 | 0,4                                   |   |
| 229     | E5          | 1110 0101 | 0,5                                   |   |
| 230     | E6          | 1110 0110 | 0,6                                   |   |
| 231     | E7          | 1110 0111 | 0,7                                   |   |
| 232     | E8          | 1110 1000 | 0,8                                   |   |
| 233     | E9          | 1110 1001 | 0,9                                   |   |
| 234     | EA          | 1110 1010 | 11,0,9,8,2                            |   |
| 235     | EB          | 1110 1011 | 11,0,9,8,3                            |   |
| 236     | EC          | 1110 1100 | 11,0,9,8,4                            | 0<br>1<br>2<br>3<br>4<br>5<br>6<br>7<br>8<br>9<br><br>⬢ (lozenge) |
| 237     | ED          | 1110 1101 | 11,0,9,8,5                            |   |
| 238     | EE          | 1110 1110 | 11,0,9,8,6                            |   |
| 239     | EF          | 1110 1111 | 11,0,9,8,7                            |   |
| 240     | F0          | 1111 0000 | 0                                     |   |
| 241     | F1          | 1111 0001 | 1                                     |   |
| 242     | F2          | 1111 0010 | 2                                     |   |
| 243     | F3          | 1111 0011 | 3                                     |   |
| 244     | F4          | 1111 0100 | 4                                     |   |
| 245     | F5          | 1111 0101 | 5                                     |   |
| 246     | F6          | 1111 0110 | 6                                     |   |
| 247     | F7          | 1111 0111 | 7                                     |   |
| 248     | F8          | 1111 1000 | 8                                     |   |
| 249     | F9          | 1111 1001 | 9                                     |   |
| 250     | FA          | 1111 1010 | 12,11,0,9,8,2                         |   |
| 251     | FB          | 1111 1011 | 12,11,0,9,8,3                         |   |
| 252     | FC          | 1111 1100 | 12,11,0,9,8,4                         |   |
| 253     | FD          | 1111 1101 | 12,11,0,9,8,5                         |   |
| 254     | FE          | 1111 1110 | 12,11,0,9,8,6                         |   |
| 255     | FF          | 1111 1111 | 12,11,0,9,8,7                         |   |

# **APPENDIX G** **POWERS OF TWO TABLE**

| $2^n$             | $n$ | $2^{-n}$  |
|-------------------|-----|---|
| 1                 | 0   | 1.0   |
| 2                 | 1   | 0.5   |
| 4                 | 2   | 0.25  |
| 8                 | 3   | 0.125   |
| 16                | 4   | 0.062 5   |
| 32                | 5   | 0.031 25  |
| 64                | 6   | 0.015 625   |
| 128               | 7   | 0.007 812 5   |
| 256               | 8   | 0.003 906 25  |
| 512               | 9   | 0.001 953 125   |
| 1 024             | 10  | 0.000 976 562 5   |
| 2 048             | 11  | 0.000 488 281 25  |
| 4 096             | 12  | 0.000 244 140 625                                       |
| 8 192             | 13  | 0.000 122 070 312 5                                     |
| 16 384            | 14  | 0.000 061 035 156 25                                    |
| 32 768            | 15  | 0.000 030 517 578 125                                   |
| 65 536            | 16  | 0.000 015 258 789 062 5                                 |
| 131 072           | 17  | 0.000 007 629 394 531 25                                |
| 262 144           | 18  | 0.000 003 814 697 265 625                               |
| 524 288           | 19  | 0.000 001 907 348 632 812 5                             |
| 1 048 576         | 20  | 0.000 000 953 674 316 406 25                            |
| 2 097 152         | 21  | 0.000 000 476 837 158 203 125                           |
| 4 194 304         | 22  | 0.000 000 238 418 579 101 562 5                         |
| 8 388 608         | 23  | 0.000 000 119 209 289 550 781 25                        |
| 16 777 216        | 24  | 0.000 000 059 604 644 775 390 625                       |
| 33 554 432        | 25  | 0.000 000 029 802 322 387 695 312 5                     |
| 67 108 864        | 26  | 0.000 000 014 901 161 193 847 656 25                    |
| 134 217 728       | 27  | 0.000 000 007 450 580 596 923 828 125                   |
| 268 435 456       | 28  | 0.000 000 003 725 290 298 461 914 062 5                 |
| 536 870 912       | 29  | 0.000 000 001 862 645 149 230 957 031 45                |
| 1 073 741 824     | 30  | 0.000 000 000 931 322 574 615 478 515 625               |
| 2 147 483 648     | 31  | 0.000 000 000 465 661 287 307 739 257 812 5             |
| 4 294 967 296     | 32  | 0.000 000 000 232 830 643 653 869 628 906 25            |
| 8 589 934 592     | 33  | 0.000 000 000 116 415 321 826 934 814 453 125           |
| 17 179 869 184    | 34  | 0.000 000 000 058 207 660 913 467 407 226 562 5         |
| 34 359 738 368    | 35  | 0.000 000 000 029 103 830 456 733 703 613 281 25        |
| 68 719 476 736    | 36  | 0.000 000 000 014 551 915 228 366 851 806 640 625       |
| 137 438 953 472   | 37  | 0.000 000 000 007 275 957 614 183 425 903 320 312 5     |
| 274 877 906 944   | 38  | 0.000 000 000 003 637 978 807 091 712 951 660 156 25    |
| 549 755 813 888   | 39  | 0.000 000 000 001 818 989 403 545 856 475 830 078 125   |
| 1 099 511 627 776 | 40  | 0.000 000 000 000 909 494 701 772 928 237 915 039 062 5 |

## APPENDIX H

### HEXADECIMAL-DECIMAL NUMBER CONVERSION

#### General

◆ This Appendix contains the necessary reference information for the conversion of decimal numbers to hexadecimal numbers and the conversion of binary numbers to decimal or hexadecimal.

*Example #1*  $(0011\ 1010)_2 = (3A)_{16} = (58)_{10}$

*Example #2*  $(FC)_{16} = (1111\ 1100)_2 = (252)_{10}$

In the conversion of a hexadecimal number to its decimal value the marks (0-F) represent a multiplier and their position (reading right to left) within the hexadecimal number represent the exponent of the base. Each mark is multiplied by the base raised to the appropriate power and the summation of their product is the decimal value of the number.

*Example #3*  $(36F)_{16} = 3 (16^2) + 6 (16^1) + 15 (16^0)$   
F

$(36F)_{16} = 3 (256) + 6 (16) + 15 (1) = (879)_{10}$

To convert hexadecimal to binary substitute the binary equivalent of the hexadecimal mark into its appropriate position as follows:

$$(3\ 6\ F)_{16} = (0011\ 0110\ 1111)_2$$

◆ The table in this Appendix provides for direct conversion of decimal and hexadecimal numbers in these ranges:

#### Hexadecimal- Decimal Number Conversion Table

| <i>Hexadecimal</i> | <i>Decimal</i>   |
|--------------------|------------------|
| 00000 to 01FFF     | 000000 to 008191 |

For numbers outside the range of the table, add the following values to the table figures:

| <i>Hexadecimal</i> | <i>Decimal</i> |
|--------------------|----------------|
| 3000               | 12288          |
| 4000               | 16384          |
| 5000               | 20480          |
| 6000               | 24576          |
| 7000               | 28672          |
| 8000               | 32768          |
| 9000               | 36864          |
| A000               | 40960          |
| B000               | 45056          |
| C000               | 49152          |
| D000               | 53248          |
| E000               | 57344          |
| F000               | 61440          |

## HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 0000 | 000000 | 000001 | 000002 | 000003 | 000004 | 000005 | 000006 | 000007 | 000008 | 000009 | 000010 | 000011 | 000012 | 000013 | 000014 | 000015 |
| 0001 | 000016 | 000017 | 000018 | 000019 | 000020 | 000021 | 000022 | 000023 | 000024 | 000025 | 000026 | 000027 | 000028 | 000029 | 000030 | 000031 |
| 0002 | 000032 | 000033 | 000034 | 000035 | 000036 | 000037 | 000038 | 000039 | 000040 | 000041 | 000042 | 000043 | 000044 | 000045 | 000046 | 000047 |
| 0003 | 000048 | 000049 | 000050 | 000051 | 000052 | 000053 | 000054 | 000055 | 000056 | 000057 | 000058 | 000059 | 000060 | 000061 | 000062 | 000063 |
| 0004 | 000064 | 000065 | 000066 | 000067 | 000068 | 000069 | 000070 | 000071 | 000072 | 000073 | 000074 | 000075 | 000076 | 000077 | 000078 | 000079 |
| 0005 | 000080 | 000081 | 000082 | 000083 | 000084 | 000085 | 000086 | 000087 | 000088 | 000089 | 000090 | 000091 | 000092 | 000093 | 000094 | 000095 |
| 0006 | 000096 | 000097 | 000098 | 000099 | 000100 | 000101 | 000102 | 000103 | 000104 | 000105 | 000106 | 000107 | 000108 | 000109 | 000110 | 000111 |
| 0007 | 000112 | 000113 | 000114 | 000115 | 000116 | 000117 | 000118 | 000119 | 000120 | 000121 | 000122 | 000123 | 000124 | 000125 | 000126 | 000127 |
| 0008 | 000128 | 000129 | 000130 | 000131 | 000132 | 000133 | 000134 | 000135 | 000136 | 000137 | 000138 | 000139 | 000140 | 000141 | 000142 | 000143 |
| 0009 | 000144 | 000145 | 000146 | 000147 | 000148 | 000149 | 000150 | 000151 | 000152 | 000153 | 000154 | 000155 | 000156 | 000157 | 000158 | 000159 |
| 000A | 000160 | 000161 | 000162 | 000163 | 000164 | 000165 | 000166 | 000167 | 000168 | 000169 | 000170 | 000171 | 000172 | 000173 | 000174 | 000175 |
| 000B | 000176 | 000177 | 000178 | 000179 | 000180 | 000181 | 000182 | 000183 | 000184 | 000185 | 000186 | 000187 | 000188 | 000189 | 000190 | 000191 |
| 000C | 000192 | 000193 | 000194 | 000195 | 000196 | 000197 | 000198 | 000199 | 000200 | 000201 | 000202 | 000203 | 000204 | 000205 | 000206 | 000207 |
| 000D | 000208 | 000209 | 000210 | 000211 | 000212 | 000213 | 000214 | 000215 | 000216 | 000217 | 000218 | 000219 | 000220 | 000221 | 000222 | 000223 |
| 000E | 000224 | 000225 | 000226 | 000227 | 000228 | 000229 | 000230 | 000231 | 000232 | 000233 | 000234 | 000235 | 000236 | 000237 | 000238 | 000239 |
| 000F | 000240 | 000241 | 000242 | 000243 | 000244 | 000245 | 000246 | 000247 | 000248 | 000249 | 000250 | 000251 | 000252 | 000253 | 000254 | 000255 |

| 0    | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |        |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 0010 | 000256 | 000257 | 000258 | 000259 | 000260 | 000261 | 000262 | 000263 | 000264 | 000265 | 000266 | 000267 | 000268 | 000269 | 000270 | 000271 |
| 0011 | 000272 | 000273 | 000274 | 000275 | 000276 | 000277 | 000278 | 000279 | 000280 | 000281 | 000282 | 000283 | 000284 | 000285 | 000286 | 000287 |
| 0012 | 000288 | 000289 | 000290 | 000291 | 000292 | 000293 | 000294 | 000295 | 000296 | 000297 | 000298 | 000299 | 000300 | 000301 | 000302 | 000303 |
| 0013 | 000304 | 000305 | 000306 | 000307 | 000308 | 000309 | 000310 | 000311 | 000312 | 000313 | 000314 | 000315 | 000316 | 000317 | 000318 | 000319 |
| 0014 | 000320 | 000321 | 000322 | 000323 | 000324 | 000325 | 000326 | 000327 | 000328 | 000329 | 000330 | 000331 | 000332 | 000333 | 000334 | 000335 |
| 0015 | 000336 | 000337 | 000338 | 000339 | 000340 | 000341 | 000342 | 000343 | 000344 | 000345 | 000346 | 000347 | 000348 | 000349 | 000350 | 000351 |
| 0016 | 000352 | 000353 | 000354 | 000355 | 000356 | 000357 | 000358 | 000359 | 000360 | 000361 | 000362 | 000363 | 000364 | 000365 | 000366 | 000367 |
| 0017 | 000368 | 000369 | 000370 | 000371 | 000372 | 000373 | 000374 | 000375 | 000376 | 000377 | 000378 | 000379 | 000380 | 000381 | 000382 | 000383 |
| 0018 | 000384 | 000385 | 000386 | 000387 | 000388 | 000389 | 000390 | 000391 | 000392 | 000393 | 000394 | 000395 | 000396 | 000397 | 000398 | 000399 |
| 0019 | 000400 | 000401 | 000402 | 000403 | 000404 | 000405 | 000406 | 000407 | 000408 | 000409 | 000410 | 000411 | 000412 | 000413 | 000414 | 000415 |
| 001A | 000416 | 000417 | 000418 | 000419 | 000420 | 000421 | 000422 | 000423 | 000424 | 000425 | 000426 | 000427 | 000428 | 000429 | 000430 | 000431 |
| 001B | 000432 | 000433 | 000434 | 000435 | 000436 | 000437 | 000438 | 000439 | 000440 | 000441 | 000442 | 000443 | 000444 | 000445 | 000446 | 000447 |
| 001C | 000448 | 000449 | 000450 | 000451 | 000452 | 000453 | 000454 | 000455 | 000456 | 000457 | 000458 | 000459 | 000460 | 000461 | 000462 | 000463 |
| 001D | 000464 | 000465 | 000466 | 000467 | 000468 | 000469 | 000470 | 000471 | 000472 | 000473 | 000474 | 000475 | 000476 | 000477 | 000478 | 000479 |
| 001E | 000480 | 000481 | 000482 | 000483 | 000484 | 000485 | 000486 | 000487 | 000488 | 000489 | 000490 | 000491 | 000492 | 000493 | 000494 | 000495 |
| 001F | 000496 | 000497 | 000498 | 000499 | 000500 | 000501 | 000502 | 000503 | 000504 | 000505 | 000506 | 000507 | 000508 | 000509 | 000510 | 000511 |

| 0    | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |        |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 0020 | 000512 | 000513 | 000514 | 000515 | 000516 | 000517 | 000518 | 000519 | 000520 | 000521 | 000522 | 000523 | 000524 | 000525 | 000526 | 000527 |
| 0021 | 000528 | 000529 | 000530 | 000531 | 000532 | 000533 | 000534 | 000535 | 000536 | 000537 | 000538 | 000539 | 000540 | 000541 | 000542 | 000543 |
| 0022 | 000544 | 000545 | 000546 | 000547 | 000548 | 000549 | 000550 | 000551 | 000552 | 000553 | 000554 | 000555 | 000556 | 000557 | 000558 | 000559 |
| 0023 | 000560 | 000561 | 000562 | 000563 | 000564 | 000565 | 000566 | 000567 | 000568 | 000569 | 000570 | 000571 | 000572 | 000573 | 000574 | 000575 |
| 0024 | 000576 | 000577 | 000578 | 000579 | 000580 | 000581 | 000582 | 000583 | 000584 | 000585 | 000586 | 000587 | 000588 | 000589 | 000590 | 000591 |
| 0025 | 000592 | 000593 | 000594 | 000595 | 000596 | 000597 | 000598 | 000599 | 000600 | 000601 | 000602 | 000603 | 000604 | 000605 | 000606 | 000607 |
| 0026 | 000608 | 000609 | 000610 | 000611 | 000612 | 000613 | 000614 | 000615 | 000616 | 000617 | 000618 | 000619 | 000620 | 000621 | 000622 | 000623 |
| 0027 | 000624 | 000625 | 000626 | 000627 | 000628 | 000629 | 000630 | 000631 | 000632 | 000633 | 000634 | 000635 | 000636 | 000637 | 000638 | 000639 |
| 0028 | 000640 | 000641 | 000642 | 000643 | 000644 | 000645 | 000646 | 000647 | 000648 | 000649 | 000650 | 000651 | 000652 | 000653 | 000654 | 000655 |
| 0029 | 000656 | 000657 | 000658 | 000659 | 000660 | 000661 | 000662 | 000663 | 000664 | 000665 | 000666 | 000667 | 000668 | 000669 | 000670 | 000671 |
| 002A | 000672 | 000673 | 000674 | 000675 | 000676 | 000677 | 000678 | 000679 | 000680 | 000681 | 000682 | 000683 | 000684 | 000685 | 000686 | 000687 |
| 002B | 000688 | 000689 | 000690 | 000691 | 000692 | 000693 | 000694 | 000695 | 000696 | 000697 | 000698 | 000699 | 000700 | 000701 | 000702 | 000703 |
| 002C | 000704 | 000705 | 000706 | 000707 | 000708 | 000709 | 000710 | 000711 | 000712 | 000713 | 000714 | 000715 | 000716 | 000717 | 000718 | 000719 |
| 002D | 000720 | 000721 | 000722 | 000723 | 000724 | 000725 | 000726 | 000727 | 000728 | 000729 | 000730 | 000731 | 000732 | 000733 | 000734 | 000735 |
| 002E | 000736 | 000737 | 000738 | 000739 | 000740 | 000741 | 000742 | 000743 | 000744 | 000745 | 000746 | 000747 | 000748 | 000749 | 000750 | 000751 |
| 002F | 000752 | 000753 | 000754 | 000755 | 000756 | 000757 | 000758 | 000759 | 000760 | 000761 | 000762 | 000763 | 000764 | 000765 | 000766 | 000767 |

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 0030 | 000768 | 000769 | 000770 | 000771 | 000772 | 000773 | 000774 | 000775 | 000776 | 000777 | 000778 | 000779 | 000780 | 000781 | 000782 | 000783 |
| 0031 | 000784 | 000785 | 000786 | 000787 | 000788 | 000789 | 000790 | 000791 | 000792 | 000793 | 000794 | 000795 | 000796 | 000797 | 000798 | 000799 |
| 0032 | 000800 | 000801 | 000802 | 000803 | 000804 | 000805 | 000806 | 000807 | 000808 | 000809 | 000810 | 000811 | 000812 | 000813 | 000814 | 000815 |
| 0033 | 000816 | 000817 | 000818 | 000819 | 000820 | 000821 | 000822 | 000823 | 000824 | 000825 | 000826 | 000827 | 000828 | 000829 | 000830 | 000831 |
| 0034 | 000832 | 000833 | 000834 | 000835 | 000836 | 000837 | 000838 | 000839 | 000840 | 000841 | 000842 | 000843 | 000844 | 000845 | 000846 | 000847 |
| 0035 | 000848 | 000849 | 000850 | 000851 | 000852 | 000853 | 000854 | 000855 | 000856 | 000857 | 000858 | 000859 | 000860 | 000861 | 000862 | 000863 |
| 0036 | 000864 | 000865 | 000866 | 000867 | 000868 | 000869 | 000870 | 000871 | 000872 | 000873 | 000874 | 000875 | 000876 | 000877 | 000878 | 000879 |
| 0037 | 000880 | 000881 | 000882 | 000883 | 000884 | 000885 | 000886 | 000887 | 000888 | 000889 | 000890 | 000891 | 000892 | 000893 | 000894 | 000895 |
| 0038 | 000896 | 000897 | 000898 | 000899 | 000900 | 000901 | 000902 | 000903 | 000904 | 000905 | 000906 | 000907 | 000908 | 000909 | 000910 | 000911 |
| 0039 | 000912 | 000913 | 000914 | 000915 | 000916 | 000917 | 000918 | 000919 | 000920 | 000921 | 000922 | 000923 | 000924 | 000925 | 000926 | 000927 |
| 003A | 000928 | 000929 | 000930 | 000931 | 000932 | 000933 | 000934 | 000935 | 000936 | 000937 | 000938 | 000939 | 000940 | 000941 | 000942 | 000943 |
| 003B | 000944 | 000945 | 000946 | 000947 | 000948 | 000949 | 000950 | 000951 | 000952 | 000953 | 000954 | 000955 | 000956 | 000957 | 000958 | 000959 |
| 003C | 000960 | 000961 | 000962 | 000963 | 000964 | 000965 | 000966 | 000967 | 000968 | 000969 | 000970 | 000971 | 000972 | 000973 | 000974 | 000975 |
| 003D | 000976 | 000977 | 000978 | 000979 | 000980 | 000981 | 000982 | 000983 | 000984 | 000985 | 000986 | 000987 | 000988 | 000989 | 000990 | 000991 |
| 003E | 000992 | 000993 | 000994 | 000995 | 000996 | 000997 | 000998 | 000999 | 001000 | 001001 | 001002 | 001003 | 001004 | 001005 | 001006 | 001007 |
| 003F | 001008 | 001009 | 001010 | 001011 | 001012 | 001013 | 001014 | 001015 | 001016 | 001017 | 001018 | 001019 | 001020 | 001021 | 001022 | 001023 |

## HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE (Cont'd)

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 0050 | 001280 | 001281 | 001282 | 001283 | 001284 | 001285 | 001286 | 001287 | 001288 | 001289 | 001290 | 001291 | 001292 | 001293 | 001294 | 001295 |
| 0051 | 001296 | 001297 | 001298 | 001299 | 001300 | 001301 | 001302 | 001303 | 001304 | 001305 | 001306 | 001307 | 001308 | 001309 | 001310 | 001311 |
| 0052 | 001312 | 001313 | 001314 | 001315 | 001316 | 001317 | 001318 | 001319 | 001320 | 001321 | 001322 | 001323 | 001324 | 001325 | 001326 | 001327 |
| 0053 | 001328 | 001329 | 001330 | 001331 | 001332 | 001333 | 001334 | 001335 | 001336 | 001337 | 001338 | 001339 | 001340 | 001341 | 001342 | 001343 |
| 0054 | 001344 | 001345 | 001346 | 001347 | 001348 | 001349 | 001350 | 001351 | 001352 | 001353 | 001354 | 001355 | 001356 | 001357 | 001358 | 001359 |
| 0055 | 001360 | 001361 | 001362 | 001363 | 001364 | 001365 | 001366 | 001367 | 001368 | 001369 | 001370 | 001371 | 001372 | 001373 | 001374 | 001375 |
| 0056 | 001376 | 001377 | 001378 | 001379 | 001380 | 001381 | 001382 | 001383 | 001384 | 001385 | 001386 | 001387 | 001388 | 001389 | 001390 | 001391 |
| 0057 | 001392 | 001393 | 001394 | 001395 | 001396 | 001397 | 001398 | 001399 | 001400 | 001401 | 001402 | 001403 | 001404 | 001405 | 001406 | 001407 |
| 0058 | 001408 | 001409 | 001410 | 001411 | 001412 | 001413 | 001414 | 001415 | 001416 | 001417 | 001418 | 001419 | 001420 | 001421 | 001422 | 001423 |
| 0059 | 001424 | 001425 | 001426 | 001427 | 001428 | 001429 | 001430 | 001431 | 001432 | 001433 | 001434 | 001435 | 001436 | 001437 | 001438 | 001439 |
| 005A | 001440 | 001441 | 001442 | 001443 | 001444 | 001445 | 001446 | 001447 | 001448 | 001449 | 001450 | 001451 | 001452 | 001453 | 001454 | 001455 |
| 005B | 001456 | 001457 | 001458 | 001459 | 001460 | 001461 | 001462 | 001463 | 001464 | 001465 | 001466 | 001467 | 001468 | 001469 | 001470 | 001471 |
| 005C | 001472 | 001473 | 001474 | 001475 | 001476 | 001477 | 001478 | 001479 | 001480 | 001481 | 001482 | 001483 | 001484 | 001485 | 001486 | 001487 |
| 005D | 001488 | 001489 | 001490 | 001491 | 001492 | 001493 | 001494 | 001495 | 001496 | 001497 | 001498 | 001499 | 001500 | 001501 | 001502 | 001503 |
| 005E | 001504 | 001505 | 001506 | 001507 | 001508 | 001509 | 001510 | 001511 | 001512 | 001513 | 001514 | 001515 | 001516 | 001517 | 001518 | 001519 |
| 005F | 001520 | 001521 | 001522 | 001523 | 001524 | 001525 | 001526 | 001527 | 001528 | 001529 | 001530 | 001531 | 001532 | 001533 | 001534 | 001535 |

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 0060 | 001536 | 001537 | 001538 | 001539 | 001540 | 001541 | 001542 | 001543 | 001544 | 001545 | 001546 | 001547 | 001548 | 001549 | 001550 | 001551 |
| 0061 | 001552 | 001553 | 001554 | 001555 | 001556 | 001557 | 001558 | 001559 | 001560 | 001561 | 001562 | 001563 | 001564 | 001565 | 001566 | 001567 |
| 0062 | 001568 | 001569 | 001570 | 001571 | 001572 | 001573 | 001574 | 001575 | 001576 | 001577 | 001578 | 001579 | 001580 | 001581 | 001582 | 001583 |
| 0063 | 001584 | 001585 | 001586 | 001587 | 001588 | 001589 | 001590 | 001591 | 001592 | 001593 | 001594 | 001595 | 001596 | 001597 | 001598 | 001599 |
| 0064 | 001600 | 001601 | 001602 | 001603 | 001604 | 001605 | 001606 | 001607 | 001608 | 001609 | 001610 | 001611 | 001612 | 001613 | 001614 | 001615 |
| 0065 | 001616 | 001617 | 001618 | 001619 | 001620 | 001621 | 001622 | 001623 | 001624 | 001625 | 001626 | 001627 | 001628 | 001629 | 001630 | 001631 |
| 0066 | 001632 | 001633 | 001634 | 001635 | 001636 | 001637 | 001638 | 001639 | 001640 | 001641 | 001642 | 001643 | 001644 | 001645 | 001646 | 001647 |
| 0067 | 001648 | 001649 | 001650 | 001651 | 001652 | 001653 | 001654 | 001655 | 001656 | 001657 | 001658 | 001659 | 001660 | 001661 | 001662 | 001663 |
| 0068 | 001664 | 001665 | 001666 | 001667 | 001668 | 001669 | 001670 | 001671 | 001672 | 001673 | 001674 | 001675 | 001676 | 001677 | 001678 | 001679 |
| 0069 | 001680 | 001681 | 001682 | 001683 | 001684 | 001685 | 001686 | 001687 | 001688 | 001689 | 001690 | 001691 | 001692 | 001693 | 001694 | 001695 |
| 006A | 001696 | 001697 | 001698 | 001699 | 001700 | 001701 | 001702 | 001703 | 001704 | 001705 | 001706 | 001707 | 001708 | 001709 | 001710 | 001711 |
| 006B | 001712 | 001713 | 001714 | 001715 | 001716 | 001717 | 001718 | 001719 | 001720 | 001721 | 001722 | 001723 | 001724 | 001725 | 001726 | 001727 |
| 006C | 001728 | 001729 | 001730 | 001731 | 001732 | 001733 | 001734 | 001735 | 001736 | 001737 | 001738 | 001739 | 001740 | 001741 | 001742 | 001743 |
| 006D | 001744 | 001745 | 001746 | 001747 | 001748 | 001749 | 001750 | 001751 | 001752 | 001753 | 001754 | 001755 | 001756 | 001757 | 001758 | 001759 |
| 006E | 001760 | 001761 | 001762 | 001763 | 001764 | 001765 | 001766 | 001767 | 001768 | 001769 | 001770 | 001771 | 001772 | 001773 | 001774 | 001775 |
| 006F | 001776 | 001777 | 001778 | 001779 | 001780 | 001781 | 001782 | 001783 | 001784 | 001785 | 001786 | 001787 | 001788 | 001789 | 001790 | 001791 |

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 0070 | 001792 | 001793 | 001794 | 001795 | 001796 | 001797 | 001798 | 001799 | 001800 | 001801 | 001802 | 001803 | 001804 | 001805 | 001806 | 001807 |
| 0071 | 001808 | 001809 | 001810 | 001811 | 001812 | 001813 | 001814 | 001815 | 001816 | 001817 | 001818 | 001819 | 001820 | 001821 | 001822 | 001823 |
| 0072 | 001824 | 001825 | 001826 | 001827 | 001828 | 001829 | 001830 | 001831 | 001832 | 001833 | 001834 | 001835 | 001836 | 001837 | 001838 | 001839 |
| 0073 | 001840 | 001841 | 001842 | 001843 | 001844 | 001845 | 001846 | 001847 | 001848 | 001849 | 001850 | 001851 | 001852 | 001853 | 001854 | 001855 |
| 0074 | 001856 | 001857 | 001858 | 001859 | 001860 | 001861 | 001862 | 001863 | 001864 | 001865 | 001866 | 001867 | 001868 | 001869 | 001870 | 001871 |
| 0075 | 001872 | 001873 | 001874 | 001875 | 001876 | 001877 | 001878 | 001879 | 001880 | 001881 | 001882 | 001883 | 001884 | 001885 | 001886 | 001887 |
| 0076 | 001888 | 001889 | 001890 | 001891 | 001892 | 001893 | 001894 | 001895 | 001896 | 001897 | 001898 | 001899 | 001900 | 001901 | 001902 | 001903 |
| 0077 | 001904 | 001905 | 001906 | 001907 | 001908 | 001909 | 001910 | 001911 | 001912 | 001913 | 001914 | 001915 | 001916 | 001917 | 001918 | 001919 |
| 0078 | 001920 | 001921 | 001922 | 001923 | 001924 | 001925 | 001926 | 001927 | 001928 | 001929 | 001930 | 001931 | 001932 | 001933 | 001934 | 001935 |
| 0079 | 001936 | 001937 | 001938 | 001939 | 001940 | 001941 | 001942 | 001943 | 001944 | 001945 | 001946 | 001947 | 001948 | 001949 | 001950 | 001951 |
| 007A | 001952 | 001953 | 001954 | 001955 | 001956 | 001957 | 001958 | 001959 | 001960 | 001961 | 001962 | 001963 | 001964 | 001965 | 001966 | 001967 |
| 007B | 001968 | 001969 | 001970 | 001971 | 001972 | 001973 | 001974 | 001975 | 001976 | 001977 | 001978 | 001979 | 001980 | 001981 | 001982 | 001983 |
| 007C | 001984 | 001985 | 001986 | 001987 | 001988 | 001989 | 001990 | 001991 | 001992 | 001993 | 001994 | 001995 | 001996 | 001997 | 001998 | 001999 |
| 007D | 002000 | 002001 | 002002 | 002003 | 002004 | 002005 | 002006 | 002007 | 002008 | 002009 | 002010 | 002011 | 002012 | 002013 | 002014 | 002015 |
| 007E | 002016 | 002017 | 002018 | 002019 | 002020 | 002021 | 002022 | 002023 | 002024 | 002025 | 002026 | 002027 | 002028 | 002029 | 002030 | 002031 |
| 007F | 002032 | 002033 | 002034 | 002035 | 002036 | 002037 | 002038 | 002039 | 002040 | 002041 | 002042 | 002043 | 002044 | 002045 | 002046 | 002047 |

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 0080 | 002048 | 002049 | 002050 | 002051 | 002052 | 002053 | 002054 | 002055 | 002056 | 002057 | 002058 | 002059 | 002060 | 002061 | 002062 | 002063 |
| 0081 | 002064 | 002065 | 002066 | 002067 | 002068 | 002069 | 002070 | 002071 | 002072 | 002073 | 002074 | 002075 | 002076 | 002077 | 002078 | 002079 |
| 0082 | 002080 | 002081 | 002082 | 002083 | 002084 | 002085 | 002086 | 002087 | 002088 | 002089 | 002090 | 002091 | 002092 | 002093 | 002094 | 002095 |
| 0083 | 002096 | 002097 | 002098 | 002099 | 002100 | 002101 | 002102 | 002103 | 002104 | 002105 | 002106 | 002107 | 002108 | 002109 | 002110 | 002111 |
| 0084 | 002112 | 002113 | 002114 | 002115 | 002116 | 002117 | 002118 | 002119 | 002120 | 002121 | 002122 | 002123 | 002124 | 002125 | 002126 | 002127 |
| 0085 | 002128 | 002129 | 002130 | 002131 | 002132 | 002133 | 002134 | 002135 | 002136 | 002137 | 002138 | 002139 | 002140 | 002141 | 002142 | 002143 |
| 0086 | 002144 | 002145 | 002146 | 002147 | 002148 | 002149 | 002150 | 002151 | 002152 | 002153 | 002154 | 002155 | 002156 | 002157 | 002158 | 002159 |
| 0087 | 002160 | 002161 | 002162 | 002163 | 002164 | 002165 | 002166 | 002167 | 002168 | 002169 | 002170 | 002171 | 002172 | 002173 | 002174 | 002175 |
| 0088 | 002176 | 002177 | 002178 | 002179 | 002180 | 002181 | 002182 | 002183 | 002184 | 002185 | 002186 | 002187 | 002188 | 002189 | 002190 | 002191 |
| 0089 | 002192 | 002193 | 002194 | 002195 | 002196 | 002197 | 002198 | 002199 | 002200 | 002201 | 002202 | 002203 | 002204 | 002205 | 002206 | 002207 |
| 008A | 002208 | 002209 | 002210 | 002211 | 002212 | 002213 | 002214 | 002215 | 002216 | 002217 | 002218 | 002219 | 002220 | 002221 | 002222 | 002223 |
| 008B | 002224 | 002225 | 002226 | 002227 | 002228 | 002229 | 002230 | 002231 | 002232 | 002233 | 002234 | 002235 | 002236 | 002237 | 002238 | 002239 |
| 008C | 002240 | 002241 | 002242 | 002243 | 002244 | 002245 | 002246 | 002247 | 002248 | 002249 | 002250 | 002251 | 002252 | 002253 | 002254 | 002255 |
| 008D | 002256 | 002257 | 002258 | 002259 | 002260 | 002261 | 002262 | 002263 | 002264 | 002265 | 002266 | 002267 | 002268 | 002269 | 002270 | 002271 |
| 008E | 002272 | 002273 | 002274 | 002275 | 002276 | 002277 | 002278 | 002279 | 002280 | 002281 | 002282 | 002283 | 002284 | 002285 | 002286 | 002287 |
| 008F | 002288 | 002289 | 002290 | 002291 | 002292 | 002293 | 002294 | 002295 | 002296 | 002297 | 002298 | 002299 | 002300 | 002301 | 002302 | 002303 |

## HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE (Cont'd)

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 00A0 | 002560 | 002561 | 002562 | 002563 | 002564 | 002565 | 002566 | 002567 | 002568 | 002569 | 002570 | 002571 | 002572 | 002573 | 002574 | 002575 |
| 00A1 | 002576 | 002577 | 002578 | 002579 | 002580 | 002581 | 002582 | 002583 | 002584 | 002585 | 002586 | 002587 | 002588 | 002589 | 002590 | 002591 |
| 00A2 | 002592 | 002593 | 002594 | 002595 | 002596 | 002597 | 002598 | 002599 | 002600 | 002601 | 002602 | 002603 | 002604 | 002605 | 002606 | 002607 |
| 00A3 | 002608 | 002609 | 002610 | 002611 | 002612 | 002613 | 002614 | 002615 | 002616 | 002617 | 002618 | 002619 | 002620 | 002621 | 002622 | 002623 |
| 00A4 | 002624 | 002625 | 002626 | 002627 | 002628 | 002629 | 002630 | 002631 | 002632 | 002633 | 002634 | 002635 | 002636 | 002637 | 002638 | 002639 |
| 00A5 | 002640 | 002641 | 002642 | 002643 | 002644 | 002645 | 002646 | 002647 | 002648 | 002649 | 002650 | 002651 | 002652 | 002653 | 002654 | 002655 |
| 00A6 | 002656 | 002657 | 002658 | 002659 | 002660 | 002661 | 002662 | 002663 | 002664 | 002665 | 002666 | 002667 | 002668 | 002669 | 002670 | 002671 |
| 00A7 | 002672 | 002673 | 002674 | 002675 | 002676 | 002677 | 002678 | 002679 | 002680 | 002681 | 002682 | 002683 | 002684 | 002685 | 002686 | 002687 |
| 00A8 | 002688 | 002689 | 002690 | 002691 | 002692 | 002693 | 002694 | 002695 | 002696 | 002697 | 002698 | 002699 | 002700 | 002701 | 002702 | 002703 |
| 00A9 | 002704 | 002705 | 002706 | 002707 | 002708 | 002709 | 002710 | 002711 | 002712 | 002713 | 002714 | 002715 | 002716 | 002717 | 002718 | 002719 |
| 00AA | 002720 | 002721 | 002722 | 002723 | 002724 | 002725 | 002726 | 002727 | 002728 | 002729 | 002730 | 002731 | 002732 | 002733 | 002734 | 002735 |
| 00AB | 002736 | 002737 | 002738 | 002739 | 002740 | 002741 | 002742 | 002743 | 002744 | 002745 | 002746 | 002747 | 002748 | 002749 | 002750 | 002751 |
| 00AC | 002752 | 002753 | 002754 | 002755 | 002756 | 002757 | 002758 | 002759 | 002760 | 002761 | 002762 | 002763 | 002764 | 002765 | 002766 | 002767 |
| 00AD | 002768 | 002769 | 002770 | 002771 | 002772 | 002773 | 002774 | 002775 | 002776 | 002777 | 002778 | 002779 | 002780 | 002781 | 002782 | 002783 |
| 00AE | 002784 | 002785 | 002786 | 002787 | 002788 | 002789 | 002790 | 002791 | 002792 | 002793 | 002794 | 002795 | 002796 | 002797 | 002798 | 002799 |
| 00AF | 002800 | 002801 | 002802 | 002803 | 002804 | 002805 | 002806 | 002807 | 002808 | 002809 | 002810 | 002811 | 002812 | 002813 | 002814 | 002815 |

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 00B0 | 002816 | 002817 | 002818 | 002819 | 002820 | 002821 | 002822 | 002823 | 002824 | 002825 | 002826 | 002827 | 002828 | 002829 | 002830 | 002831 |
| 00B1 | 002832 | 002833 | 002834 | 002835 | 002836 | 002837 | 002838 | 002839 | 002840 | 002841 | 002842 | 002843 | 002844 | 002845 | 002846 | 002847 |
| 00B2 | 002848 | 002849 | 002850 | 002851 | 002852 | 002853 | 002854 | 002855 | 002856 | 002857 | 002858 | 002859 | 002860 | 002861 | 002862 | 002863 |
| 00B3 | 002864 | 002865 | 002866 | 002867 | 002868 | 002869 | 002870 | 002871 | 002872 | 002873 | 002874 | 002875 | 002876 | 002877 | 002878 | 002879 |
| 00B4 | 002880 | 002881 | 002882 | 002883 | 002884 | 002885 | 002886 | 002887 | 002888 | 002889 | 002890 | 002891 | 002892 | 002893 | 002894 | 002895 |
| 00B5 | 002896 | 002897 | 002898 | 002899 | 002900 | 002901 | 002902 | 002903 | 002904 | 002905 | 002906 | 002907 | 002908 | 002909 | 002910 | 002911 |
| 00B6 | 002912 | 002913 | 002914 | 002915 | 002916 | 002917 | 002918 | 002919 | 002920 | 002921 | 002922 | 002923 | 002924 | 002925 | 002926 | 002927 |
| 00B7 | 002928 | 002929 | 002930 | 002931 | 002932 | 002933 | 002934 | 002935 | 002936 | 002937 | 002938 | 002939 | 002940 | 002941 | 002942 | 002943 |
| 00B8 | 002944 | 002945 | 002946 | 002947 | 002948 | 002949 | 002950 | 002951 | 002952 | 002953 | 002954 | 002955 | 002956 | 002957 | 002958 | 002959 |
| 00B9 | 002960 | 002961 | 002962 | 002963 | 002964 | 002965 | 002966 | 002967 | 002968 | 002969 | 002970 | 002971 | 002972 | 002973 | 002974 | 002975 |
| 00BA | 002976 | 002977 | 002978 | 002979 | 002980 | 002981 | 002982 | 002983 | 002984 | 002985 | 002986 | 002987 | 002988 | 002989 | 002990 | 002991 |
| 00BB | 002992 | 002993 | 002994 | 002995 | 002996 | 002997 | 002998 | 002999 | 003000 | 003001 | 003002 | 003003 | 003004 | 003005 | 003006 | 003007 |
| 00BC | 003008 | 003009 | 003010 | 003011 | 003012 | 003013 | 003014 | 003015 | 003016 | 003017 | 003018 | 003019 | 003020 | 003021 | 003022 | 003023 |
| 00BD | 003024 | 003025 | 003026 | 003027 | 003028 | 003029 | 003030 | 003031 | 003032 | 003033 | 003034 | 003035 | 003036 | 003037 | 003038 | 003039 |
| 00BE | 003040 | 003041 | 003042 | 003043 | 003044 | 003045 | 003046 | 003047 | 003048 | 003049 | 003050 | 003051 | 003052 | 003053 | 003054 | 003055 |
| 00BF | 003056 | 003057 | 003058 | 003059 | 003060 | 003061 | 003062 | 003063 | 003064 | 003065 | 003066 | 003067 | 003068 | 003069 | 003070 | 003071 |

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 00C0 | 003072 | 003073 | 003074 | 003075 | 003076 | 003077 | 003078 | 003079 | 003080 | 003081 | 003082 | 003083 | 003084 | 003085 | 003086 | 003087 |
| 00C1 | 003088 | 003089 | 003090 | 003091 | 003092 | 003093 | 003094 | 003095 | 003096 | 003097 | 003098 | 003099 | 003100 | 003101 | 003102 | 003103 |
| 00C2 | 003104 | 003105 | 003106 | 003107 | 003108 | 003109 | 003110 | 003111 | 003112 | 003113 | 003114 | 003115 | 003116 | 003117 | 003118 | 003119 |
| 00C3 | 003120 | 003121 | 003122 | 003123 | 003124 | 003125 | 003126 | 003127 | 003128 | 003129 | 003130 | 003131 | 003132 | 003133 | 003134 | 003135 |
| 00C4 | 003136 | 003137 | 003138 | 003139 | 003140 | 003141 | 003142 | 003143 | 003144 | 003145 | 003146 | 003147 | 003148 | 003149 | 003150 | 003151 |
| 00C5 | 003152 | 003153 | 003154 | 003155 | 003156 | 003157 | 003158 | 003159 | 003160 | 003161 | 003162 | 003163 | 003164 | 003165 | 003166 | 003167 |
| 00C6 | 003168 | 003169 | 003170 | 003171 | 003172 | 003173 | 003174 | 003175 | 003176 | 003177 | 003178 | 003179 | 003180 | 003181 | 003182 | 003183 |
| 00C7 | 003184 | 003185 | 003186 | 003187 | 003188 | 003189 | 003190 | 003191 | 003192 | 003193 | 003194 | 003195 | 003196 | 003197 | 003198 | 003199 |
| 00C8 | 003200 | 003201 | 003202 | 003203 | 003204 | 003205 | 003206 | 003207 | 003208 | 003209 | 003210 | 003211 | 003212 | 003213 | 003214 | 003215 |
| 00C9 | 003216 | 003217 | 003218 | 003219 | 003220 | 003221 | 003222 | 003223 | 003224 | 003225 | 003226 | 003227 | 003228 | 003229 | 003230 | 003231 |
| 00CA | 003232 | 003233 | 003234 | 003235 | 003236 | 003237 | 003238 | 003239 | 003240 | 003241 | 003242 | 003243 | 003244 | 003245 | 003246 | 003247 |
| 00CB | 003248 | 003249 | 003250 | 003251 | 003252 | 003253 | 003254 | 003255 | 003256 | 003257 | 003258 | 003259 | 003260 | 003261 | 003262 | 003263 |
| 00CC | 003264 | 003265 | 003266 | 003267 | 003268 | 003269 | 003270 | 003271 | 003272 | 003273 | 003274 | 003275 | 003276 | 003277 | 003278 | 003279 |
| 00CD | 003280 | 003281 | 003282 | 003283 | 003284 | 003285 | 003286 | 003287 | 003288 | 003289 | 003290 | 003291 | 003292 | 003293 | 003294 | 003295 |
| 00CE | 003296 | 003297 | 003298 | 003299 | 003300 | 003301 | 003302 | 003303 | 003304 | 003305 | 003306 | 003307 | 003308 | 003309 | 003310 | 003311 |
| 00CF | 003312 | 003313 | 003314 | 003315 | 003316 | 003317 | 003318 | 003319 | 003320 | 003321 | 003322 | 003323 | 003324 | 003325 | 003326 | 003327 |

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 00D0 | 003328 | 003329 | 003330 | 003331 | 003332 | 003333 | 003334 | 003335 | 003336 | 003337 | 003338 | 003339 | 003340 | 003341 | 003342 | 003343 |
| 00D1 | 003344 | 003345 | 003346 | 003347 | 003348 | 003349 | 003350 | 003351 | 003352 | 003353 | 003354 | 003355 | 003356 | 003357 | 003358 | 003359 |
| 00D2 | 003360 | 003361 | 003362 | 003363 | 003364 | 003365 | 003366 | 003367 | 003368 | 003369 | 003370 | 003371 | 003372 | 003373 | 003374 | 003375 |
| 00D3 | 003376 | 003377 | 003378 | 003379 | 003380 | 003381 | 003382 | 003383 | 003384 | 003385 | 003386 | 003387 | 003388 | 003389 | 003390 | 003391 |
| 00D4 | 003392 | 003393 | 003394 | 003395 | 003396 | 003397 | 003398 | 003399 | 003400 | 003401 | 003402 | 003403 | 003404 | 003405 | 003406 | 003407 |
| 00D5 | 003408 | 003409 | 003410 | 003411 | 003412 | 003413 | 003414 | 003415 | 003416 | 003417 | 003418 | 003419 | 003420 | 003421 | 003422 | 003423 |
| 00D6 | 003424 | 003425 | 003426 | 003427 | 003428 | 003429 | 003430 | 003431 | 003432 | 003433 | 003434 | 003435 | 003436 | 003437 | 003438 | 003439 |
| 00D7 | 003440 | 003441 | 003442 | 003443 | 003444 | 003445 | 003446 | 003447 | 003448 | 003449 | 003450 | 003451 | 003452 | 003453 | 003454 | 003455 |
| 00D8 | 003456 | 003457 | 003458 | 003459 | 003460 | 003461 | 003462 | 003463 | 003464 | 003465 | 003466 | 003467 | 003468 | 003469 | 003470 | 003471 |
| 00D9 | 003472 | 003473 | 003474 | 003475 | 003476 | 003477 | 003478 | 003479 | 003480 | 003481 | 003482 | 003483 | 003484 | 003485 | 003486 | 003487 |
| 00DA | 003488 | 003489 | 003490 | 003491 | 003492 | 003493 | 003494 | 003495 | 003496 | 003497 | 003498 | 003499 | 003500 | 003501 | 003502 | 003503 |
| 00DB | 003504 | 003505 | 003506 | 003507 | 003508 | 003509 | 003510 | 003511 | 003512 | 003513 | 003514 | 003515 | 003516 | 003517 | 003518 | 003519 |
| 00DC | 003520 | 003521 | 003522 | 003523 | 003524 | 003525 | 003526 | 003527 | 003528 | 003529 | 003530 | 003531 | 003532 | 003533 | 003534 | 003535 |
| 00DD | 003536 | 003537 | 003538 | 003539 | 003540 | 003541 | 003542 | 003543 | 003544 | 003545 | 003546 | 003547 | 003548 | 003549 | 003550 | 003551 |
| 00DE | 003552 | 003553 | 003554 | 003555 | 003556 | 003557 | 003558 | 003559 | 003560 | 003561 | 003562 | 003563 | 003564 | 003565 | 003566 | 003567 |
| 00DF | 003568 | 003569 | 003570 | 003571 | 003572 | 003573 | 003574 | 003575 | 003576 | 003577 | 003578 | 003579 | 003580 | 003581 | 003582 | 003583 |



## HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE (Cont'd)

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 00F0 | 003840 | 003841 | 003842 | 003843 | 003844 | 003845 | 003846 | 003847 | 003848 | 003849 | 003850 | 003851 | 003852 | 003853 | 003854 | 003855 |
| 00F1 | 003856 | 003857 | 003858 | 003859 | 003860 | 003861 | 003862 | 003863 | 003864 | 003865 | 003866 | 003867 | 003868 | 003869 | 003870 | 003871 |
| 00F2 | 003872 | 003873 | 003874 | 003875 | 003876 | 003877 | 003878 | 003879 | 003880 | 003881 | 003882 | 003883 | 003884 | 003885 | 003886 | 003887 |
| 00F3 | 003888 | 003889 | 003890 | 003891 | 003892 | 003893 | 003894 | 003895 | 003896 | 003897 | 003898 | 003899 | 003900 | 003901 | 003902 | 003903 |
| 00F4 | 003904 | 003905 | 003906 | 003907 | 003908 | 003909 | 003910 | 003911 | 003912 | 003913 | 003914 | 003915 | 003916 | 003917 | 003918 | 003919 |
| 00F5 | 003920 | 003921 | 003922 | 003923 | 003924 | 003925 | 003926 | 003927 | 003928 | 003929 | 003930 | 003931 | 003932 | 003933 | 003934 | 003935 |
| 00F6 | 003936 | 003937 | 003938 | 003939 | 003940 | 003941 | 003942 | 003943 | 003944 | 003945 | 003946 | 003947 | 003948 | 003949 | 003950 | 003951 |
| 00F7 | 003952 | 003953 | 003954 | 003955 | 003956 | 003957 | 003958 | 003959 | 003960 | 003961 | 003962 | 003963 | 003964 | 003965 | 003966 | 003967 |
| 00F8 | 003968 | 003969 | 003970 | 003971 | 003972 | 003973 | 003974 | 003975 | 003976 | 003977 | 003978 | 003979 | 003980 | 003981 | 003982 | 003983 |
| 00F9 | 003984 | 003985 | 003986 | 003987 | 003988 | 003989 | 003990 | 003991 | 003992 | 003993 | 003994 | 003995 | 003996 | 003997 | 003998 | 003999 |
| 00FA | 004000 | 004001 | 004002 | 004003 | 004004 | 004005 | 004006 | 004007 | 004008 | 004009 | 004010 | 004011 | 004012 | 004013 | 004014 | 004015 |
| 00FB | 004016 | 004017 | 004018 | 004019 | 004020 | 004021 | 004022 | 004023 | 004024 | 004025 | 004026 | 004027 | 004028 | 004029 | 004030 | 004031 |
| 00FC | 004032 | 004033 | 004034 | 004035 | 004036 | 004037 | 004038 | 004039 | 004040 | 004041 | 004042 | 004043 | 004044 | 004045 | 004046 | 004047 |
| 00FD | 004048 | 004049 | 004050 | 004051 | 004052 | 004053 | 004054 | 004055 | 004056 | 004057 | 004058 | 004059 | 004060 | 004061 | 004062 | 004063 |
| 00FE | 004064 | 004065 | 004066 | 004067 | 004068 | 004069 | 004070 | 004071 | 004072 | 004073 | 004074 | 004075 | 004076 | 004077 | 004078 | 004079 |
| 00FF | 004080 | 004081 | 004082 | 004083 | 004084 | 004085 | 004086 | 004087 | 004088 | 004089 | 004090 | 004091 | 004092 | 004093 | 004094 | 004095 |

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 0100 | 004096 | 004097 | 004098 | 004099 | 004100 | 004101 | 004102 | 004103 | 004104 | 004105 | 004106 | 004107 | 004108 | 004109 | 004110 | 004111 |
| 0101 | 004112 | 004113 | 004114 | 004115 | 004116 | 004117 | 004118 | 004119 | 004120 | 004121 | 004122 | 004123 | 004124 | 004125 | 004126 | 004127 |
| 0102 | 004128 | 004129 | 004130 | 004131 | 004132 | 004133 | 004134 | 004135 | 004136 | 004137 | 004138 | 004139 | 004140 | 004141 | 004142 | 004143 |
| 0103 | 004144 | 004145 | 004146 | 004147 | 004148 | 004149 | 004150 | 004151 | 004152 | 004153 | 004154 | 004155 | 004156 | 004157 | 004158 | 004159 |
| 0104 | 004160 | 004161 | 004162 | 004163 | 004164 | 004165 | 004166 | 004167 | 004168 | 004169 | 004170 | 004171 | 004172 | 004173 | 004174 | 004175 |
| 0105 | 004176 | 004177 | 004178 | 004179 | 004180 | 004181 | 004182 | 004183 | 004184 | 004185 | 004186 | 004187 | 004188 | 004189 | 004190 | 004191 |
| 0106 | 004192 | 004193 | 004194 | 004195 | 004196 | 004197 | 004198 | 004199 | 004200 | 004201 | 004202 | 004203 | 004204 | 004205 | 004206 | 004207 |
| 0107 | 004208 | 004209 | 004210 | 004211 | 004212 | 004213 | 004214 | 004215 | 004216 | 004217 | 004218 | 004219 | 004220 | 004221 | 004222 | 004223 |
| 0108 | 004224 | 004225 | 004226 | 004227 | 004228 | 004229 | 004230 | 004231 | 004232 | 004233 | 004234 | 004235 | 004236 | 004237 | 004238 | 004239 |
| 0109 | 004240 | 004241 | 004242 | 004243 | 004244 | 004245 | 004246 | 004247 | 004248 | 004249 | 004250 | 004251 | 004252 | 004253 | 004254 | 004255 |
| 010A | 004256 | 004257 | 004258 | 004259 | 004260 | 004261 | 004262 | 004263 | 004264 | 004265 | 004266 | 004267 | 004268 | 004269 | 004270 | 004271 |
| 010B | 004272 | 004273 | 004274 | 004275 | 004276 | 004277 | 004278 | 004279 | 004280 | 004281 | 004282 | 004283 | 004284 | 004285 | 004286 | 004287 |
| 010C | 004288 | 004289 | 004290 | 004291 | 004292 | 004293 | 004294 | 004295 | 004296 | 004297 | 004298 | 004299 | 004300 | 004301 | 004302 | 004303 |
| 010D | 004304 | 004305 | 004306 | 004307 | 004308 | 004309 | 004310 | 004311 | 004312 | 004313 | 004314 | 004315 | 004316 | 004317 | 004318 | 004319 |
| 010E | 004320 | 004321 | 004322 | 004323 | 004324 | 004325 | 004326 | 004327 | 004328 | 004329 | 004330 | 004331 | 004332 | 004333 | 004334 | 004335 |
| 010F | 004336 | 004337 | 004338 | 004339 | 004340 | 004341 | 004342 | 004343 | 004344 | 004345 | 004346 | 004347 | 004348 | 004349 | 004350 | 004351 |

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 0110 | 004352 | 004353 | 004354 | 004355 | 004356 | 004357 | 004358 | 004359 | 004360 | 004361 | 004362 | 004363 | 004364 | 004365 | 004366 | 004367 |
| 0111 | 004368 | 004369 | 004370 | 004371 | 004372 | 004373 | 004374 | 004375 | 004376 | 004377 | 004378 | 004379 | 004380 | 004381 | 004382 | 004383 |
| 0112 | 004384 | 004385 | 004386 | 004387 | 004388 | 004389 | 004390 | 004391 | 004392 | 004393 | 004394 | 004395 | 004396 | 004397 | 004398 | 004399 |
| 0113 | 004400 | 004401 | 004402 | 004403 | 004404 | 004405 | 004406 | 004407 | 004408 | 004409 | 004410 | 004411 | 004412 | 004413 | 004414 | 004415 |
| 0114 | 004416 | 004417 | 004418 | 004419 | 004420 | 004421 | 004422 | 004423 | 004424 | 004425 | 004426 | 004427 | 004428 | 004429 | 004430 | 004431 |
| 0115 | 004432 | 004433 | 004434 | 004435 | 004436 | 004437 | 004438 | 004439 | 004440 | 004441 | 004442 | 004443 | 004444 | 004445 | 004446 | 004447 |
| 0116 | 004448 | 004449 | 004450 | 004451 | 004452 | 004453 | 004454 | 004455 | 004456 | 004457 | 004458 | 004459 | 004460 | 004461 | 004462 | 004463 |
| 0117 | 004464 | 004465 | 004466 | 004467 | 004468 | 004469 | 004470 | 004471 | 004472 | 004473 | 004474 | 004475 | 004476 | 004477 | 004478 | 004479 |
| 0118 | 004480 | 004481 | 004482 | 004483 | 004484 | 004485 | 004486 | 004487 | 004488 | 004489 | 004490 | 004491 | 004492 | 004493 | 004494 | 004495 |
| 0119 | 004496 | 004497 | 004498 | 004499 | 004500 | 004501 | 004502 | 004503 | 004504 | 004505 | 004506 | 004507 | 004508 | 004509 | 004510 | 004511 |
| 011A | 004512 | 004513 | 004514 | 004515 | 004516 | 004517 | 004518 | 004519 | 004520 | 004521 | 004522 | 004523 | 004524 | 004525 | 004526 | 004527 |
| 011B | 004528 | 004529 | 004530 | 004531 | 004532 | 004533 | 004534 | 004535 | 004536 | 004537 | 004538 | 004539 | 004540 | 004541 | 004542 | 004543 |
| 011C | 004544 | 004545 | 004546 | 004547 | 004548 | 004549 | 004550 | 004551 | 004552 | 004553 | 004554 | 004555 | 004556 | 004557 | 004558 | 004559 |
| 011D | 004560 | 004561 | 004562 | 004563 | 004564 | 004565 | 004566 | 004567 | 004568 | 004569 | 004570 | 004571 | 004572 | 004573 | 004574 | 004575 |
| 011E | 004576 | 004577 | 004578 | 004579 | 004580 | 004581 | 004582 | 004583 | 004584 | 004585 | 004586 | 004587 | 004588 | 004589 | 004590 | 004591 |
| 011F | 004592 | 004593 | 004594 | 004595 | 004596 | 004597 | 004598 | 004599 | 004600 | 004601 | 004602 | 004603 | 004604 | 004605 | 004606 | 004607 |

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 0120 | 004608 | 004609 | 004610 | 004611 | 004612 | 004613 | 004614 | 004615 | 004616 | 004617 | 004618 | 004619 | 004620 | 004621 | 004622 | 004623 |
| 0121 | 004624 | 004625 | 004626 | 004627 | 004628 | 004629 | 004630 | 004631 | 004632 | 004633 | 004634 | 004635 | 004636 | 004637 | 004638 | 004639 |
| 0122 | 004640 | 004641 | 004642 | 004643 | 004644 | 004645 | 004646 | 004647 | 004648 | 004649 | 004650 | 004651 | 004652 | 004653 | 004654 | 004655 |
| 0123 | 004656 | 004657 | 004658 | 004659 | 004660 | 004661 | 004662 | 004663 | 004664 | 004665 | 004666 | 004667 | 004668 | 004669 | 004670 | 004671 |
| 0124 | 004672 | 004673 | 004674 | 004675 | 004676 | 004677 | 004678 | 004679 | 004680 | 004681 | 004682 | 004683 | 004684 | 004685 | 004686 | 004687 |
| 0125 | 004688 | 004689 | 004690 | 004691 | 004692 | 004693 | 004694 | 004695 | 004696 | 004697 | 004698 | 004699 | 004700 | 004701 | 004702 | 004703 |
| 0126 | 004704 | 004705 | 004706 | 004707 | 004708 | 004709 | 004710 | 004711 | 004712 | 004713 | 004714 | 004715 | 004716 | 004717 | 004718 | 004719 |
| 0127 | 004720 | 004721 | 004722 | 004723 | 004724 | 004725 | 004726 | 004727 | 004728 | 004729 | 004730 | 004731 | 004732 | 004733 | 004734 | 004735 |
| 0128 | 004736 | 004737 | 004738 | 004739 | 004740 | 004741 | 004742 | 004743 | 004744 | 004745 | 004746 | 004747 | 004748 | 004749 | 004750 | 004751 |
| 0129 | 004752 | 004753 | 004754 | 004755 | 004756 | 004757 | 004758 | 004759 | 004760 | 004761 | 004762 | 004763 | 004764 | 004765 | 004766 | 004767 |
| 012A | 004768 | 004769 | 004770 | 004771 | 004772 | 004773 | 004774 | 004775 | 004776 | 004777 | 004778 | 004779 | 004780 | 004781 | 004782 | 004783 |
| 012B | 004784 | 004785 | 004786 | 004787 | 004788 | 004789 | 004790 | 004791 | 004792 | 004793 | 004794 | 004795 | 004796 | 004797 | 004798 | 004799 |
| 012C | 004800 | 004801 | 004802 | 004803 | 004804 | 004805 | 004806 | 004807 | 004808 | 004809 | 004810 | 004811 | 004812 | 004813 | 004814 | 004815 |
| 012D | 004816 | 004817 | 004818 | 004819 | 004820 | 004821 | 004822 | 004823 | 004824 | 004825 | 004826 | 004827 | 004828 | 004829 | 004830 | 004831 |
| 012E | 004832 | 004833 | 004834 | 004835 | 004836 | 004837 | 004838 | 004839 | 004840 | 004841 | 004842 | 004843 | 004844 | 004845 | 004846 | 004847 |
| 012F | 004848 | 004849 | 004850 | 004851 | 004852 | 004853 | 004854 | 004855 | 004856 | 004857 | 004858 | 004859 | 004860 | 004861 | 004862 | 004863 |

Page Missing From Original  
Document

## HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE (Cont'd)

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 0190 | 006400 | 006401 | 006402 | 006403 | 006404 | 006405 | 006406 | 006407 | 006408 | 006409 | 006410 | 006411 | 006412 | 006413 | 006414 | 006415 |
| 0191 | 006416 | 006417 | 006418 | 006419 | 006420 | 006421 | 006422 | 006423 | 006424 | 006425 | 006426 | 006427 | 006428 | 006429 | 006430 | 006431 |
| 0192 | 006432 | 006433 | 006434 | 006435 | 006436 | 006437 | 006438 | 006439 | 006440 | 006441 | 006442 | 006443 | 006444 | 006445 | 006446 | 006447 |
| 0193 | 006448 | 006449 | 006450 | 006451 | 006452 | 006453 | 006454 | 006455 | 006456 | 006457 | 006458 | 006459 | 006460 | 006461 | 006462 | 006463 |
| 0194 | 006464 | 006465 | 006466 | 006467 | 006468 | 006469 | 006470 | 006471 | 006472 | 006473 | 006474 | 006475 | 006476 | 006477 | 006478 | 006479 |
| 0195 | 006480 | 006481 | 006482 | 006483 | 006484 | 006485 | 006486 | 006487 | 006488 | 006489 | 006490 | 006491 | 006492 | 006493 | 006494 | 006495 |
| 0196 | 006496 | 006497 | 006498 | 006499 | 006500 | 006501 | 006502 | 006503 | 006504 | 006505 | 006506 | 006507 | 006508 | 006509 | 006510 | 006511 |
| 0197 | 006512 | 006513 | 006514 | 006515 | 006516 | 006517 | 006518 | 006519 | 006520 | 006521 | 006522 | 006523 | 006524 | 006525 | 006526 | 006527 |
| 0198 | 006528 | 006529 | 006530 | 006531 | 006532 | 006533 | 006534 | 006535 | 006536 | 006537 | 006538 | 006539 | 006540 | 006541 | 006542 | 006543 |
| 0199 | 006544 | 006545 | 006546 | 006547 | 006548 | 006549 | 006550 | 006551 | 006552 | 006553 | 006554 | 006555 | 006556 | 006557 | 006558 | 006559 |
| 019A | 006560 | 006561 | 006562 | 006563 | 006564 | 006565 | 006566 | 006567 | 006568 | 006569 | 006570 | 006571 | 006572 | 006573 | 006574 | 006575 |
| 019B | 006576 | 006577 | 006578 | 006579 | 006580 | 006581 | 006582 | 006583 | 006584 | 006585 | 006586 | 006587 | 006588 | 006589 | 006590 | 006591 |
| 019C | 006592 | 006593 | 006594 | 006595 | 006596 | 006597 | 006598 | 006599 | 006600 | 006601 | 006602 | 006603 | 006604 | 006605 | 006606 | 006607 |
| 019D | 006608 | 006609 | 006610 | 006611 | 006612 | 006613 | 006614 | 006615 | 006616 | 006617 | 006618 | 006619 | 006620 | 006621 | 006622 | 006623 |
| 019E | 006624 | 006625 | 006626 | 006627 | 006628 | 006629 | 006630 | 006631 | 006632 | 006633 | 006634 | 006635 | 006636 | 006637 | 006638 | 006639 |
| 019F | 006640 | 006641 | 006642 | 006643 | 006644 | 006645 | 006646 | 006647 | 006648 | 006649 | 006650 | 006651 | 006652 | 006653 | 006654 | 006655 |

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 01A0 | 006656 | 006657 | 006658 | 006659 | 006660 | 006661 | 006662 | 006663 | 006664 | 006665 | 006666 | 006667 | 006668 | 006669 | 006670 | 006671 |
| 01A1 | 006672 | 006673 | 006674 | 006675 | 006676 | 006677 | 006678 | 006679 | 006680 | 006681 | 006682 | 006683 | 006684 | 006685 | 006686 | 006687 |
| 01A2 | 006688 | 006689 | 006690 | 006691 | 006692 | 006693 | 006694 | 006695 | 006696 | 006697 | 006698 | 006699 | 006700 | 006701 | 006702 | 006703 |
| 01A3 | 006704 | 006705 | 006706 | 006707 | 006708 | 006709 | 006710 | 006711 | 006712 | 006713 | 006714 | 006715 | 006716 | 006717 | 006718 | 006719 |
| 01A4 | 006720 | 006721 | 006722 | 006723 | 006724 | 006725 | 006726 | 006727 | 006728 | 006729 | 006730 | 006731 | 006732 | 006733 | 006734 | 006735 |
| 01A5 | 006736 | 006737 | 006738 | 006739 | 006740 | 006741 | 006742 | 006743 | 006744 | 006745 | 006746 | 006747 | 006748 | 006749 | 006750 | 006751 |
| 01A6 | 006752 | 006753 | 006754 | 006755 | 006756 | 006757 | 006758 | 006759 | 006760 | 006761 | 006762 | 006763 | 006764 | 006765 | 006766 | 006767 |
| 01A7 | 006768 | 006769 | 006770 | 006771 | 006772 | 006773 | 006774 | 006775 | 006776 | 006777 | 006778 | 006779 | 006780 | 006781 | 006782 | 006783 |
| 01A8 | 006784 | 006785 | 006786 | 006787 | 006788 | 006789 | 006790 | 006791 | 006792 | 006793 | 006794 | 006795 | 006796 | 006797 | 006798 | 006799 |
| 01A9 | 006800 | 006801 | 006802 | 006803 | 006804 | 006805 | 006806 | 006807 | 006808 | 006809 | 006810 | 006811 | 006812 | 006813 | 006814 | 006815 |
| 01AA | 006816 | 006817 | 006818 | 006819 | 006820 | 006821 | 006822 | 006823 | 006824 | 006825 | 006826 | 006827 | 006828 | 006829 | 006830 | 006831 |
| 01AB | 006832 | 006833 | 006834 | 006835 | 006836 | 006837 | 006838 | 006839 | 006840 | 006841 | 006842 | 006843 | 006844 | 006845 | 006846 | 006847 |
| 01AC | 006848 | 006849 | 006850 | 006851 | 006852 | 006853 | 006854 | 006855 | 006856 | 006857 | 006858 | 006859 | 006860 | 006861 | 006862 | 006863 |
| 01AD | 006864 | 006865 | 006866 | 006867 | 006868 | 006869 | 006870 | 006871 | 006872 | 006873 | 006874 | 006875 | 006876 | 006877 | 006878 | 006879 |
| 01AE | 006880 | 006881 | 006882 | 006883 | 006884 | 006885 | 006886 | 006887 | 006888 | 006889 | 006890 | 006891 | 006892 | 006893 | 006894 | 006895 |
| 01AF | 006896 | 006897 | 006898 | 006899 | 006900 | 006901 | 006902 | 006903 | 006904 | 006905 | 006906 | 006907 | 006908 | 006909 | 006910 | 006911 |

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 01B0 | 006912 | 006913 | 006914 | 006915 | 006916 | 006917 | 006918 | 006919 | 006920 | 006921 | 006922 | 006923 | 006924 | 006925 | 006926 | 006927 |
| 01B1 | 006928 | 006929 | 006930 | 006931 | 006932 | 006933 | 006934 | 006935 | 006936 | 006937 | 006938 | 006939 | 006940 | 006941 | 006942 | 006943 |
| 01B2 | 006944 | 006945 | 006946 | 006947 | 006948 | 006949 | 006950 | 006951 | 006952 | 006953 | 006954 | 006955 | 006956 | 006957 | 006958 | 006959 |
| 01B3 | 006960 | 006961 | 006962 | 006963 | 006964 | 006965 | 006966 | 006967 | 006968 | 006969 | 006970 | 006971 | 006972 | 006973 | 006974 | 006975 |
| 01B4 | 006976 | 006977 | 006978 | 006979 | 006980 | 006981 | 006982 | 006983 | 006984 | 006985 | 006986 | 006987 | 006988 | 006989 | 006990 | 006991 |
| 01B5 | 006992 | 006993 | 006994 | 006995 | 006996 | 006997 | 006998 | 006999 | 007000 | 007001 | 007002 | 007003 | 007004 | 007005 | 007006 | 007007 |
| 01B6 | 007008 | 007009 | 007010 | 007011 | 007012 | 007013 | 007014 | 007015 | 007016 | 007017 | 007018 | 007019 | 007020 | 007021 | 007022 | 007023 |
| 01B7 | 007024 | 007025 | 007026 | 007027 | 007028 | 007029 | 007030 | 007031 | 007032 | 007033 | 007034 | 007035 | 007036 | 007037 | 007038 | 007039 |
| 01B8 | 007040 | 007041 | 007042 | 007043 | 007044 | 007045 | 007046 | 007047 | 007048 | 007049 | 007050 | 007051 | 007052 | 007053 | 007054 | 007055 |
| 01B9 | 007056 | 007057 | 007058 | 007059 | 007060 | 007061 | 007062 | 007063 | 007064 | 007065 | 007066 | 007067 | 007068 | 007069 | 007070 | 007071 |
| 01BA | 007072 | 007073 | 007074 | 007075 | 007076 | 007077 | 007078 | 007079 | 007080 | 007081 | 007082 | 007083 | 007084 | 007085 | 007086 | 007087 |
| 01BB | 007088 | 007089 | 007090 | 007091 | 007092 | 007093 | 007094 | 007095 | 007096 | 007097 | 007098 | 007099 | 007100 | 007101 | 007102 | 007103 |
| 01BC | 007104 | 007105 | 007106 | 007107 | 007108 | 007109 | 007110 | 007111 | 007112 | 007113 | 007114 | 007115 | 007116 | 007117 | 007118 | 007119 |
| 01BD | 007120 | 007121 | 007122 | 007123 | 007124 | 007125 | 007126 | 007127 | 007128 | 007129 | 007130 | 007131 | 007132 | 007133 | 007134 | 007135 |
| 01BE | 007136 | 007137 | 007138 | 007139 | 007140 | 007141 | 007142 | 007143 | 007144 | 007145 | 007146 | 007147 | 007148 | 007149 | 007150 | 007151 |
| 01BF | 007152 | 007153 | 007154 | 007155 | 007156 | 007157 | 007158 | 007159 | 007160 | 007161 | 007162 | 007163 | 007164 | 007165 | 007166 | 007167 |

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 01C0 | 007168 | 007169 | 007170 | 007171 | 007172 | 007173 | 007174 | 007175 | 007176 | 007177 | 007178 | 007179 | 007180 | 007181 | 007182 | 007183 |
| 01C1 | 007184 | 007185 | 007186 | 007187 | 007188 | 007189 | 007190 | 007191 | 007192 | 007193 | 007194 | 007195 | 007196 | 007197 | 007198 | 007199 |
| 01C2 | 007200 | 007201 | 007202 | 007203 | 007204 | 007205 | 007206 | 007207 | 007208 | 007209 | 007210 | 007211 | 007212 | 007213 | 007214 | 007215 |
| 01C3 | 007216 | 007217 | 007218 | 007219 | 007220 | 007221 | 007222 | 007223 | 007224 | 007225 | 007226 | 007227 | 007228 | 007229 | 007230 | 007231 |
| 01C4 | 007232 | 007233 | 007234 | 007235 | 007236 | 007237 | 007238 | 007239 | 007240 | 007241 | 007242 | 007243 | 007244 | 007245 | 007246 | 007247 |
| 01C5 | 007248 | 007249 | 007250 | 007251 | 007252 | 007253 | 007254 | 007255 | 007256 | 007257 | 007258 | 007259 | 007260 | 007261 | 007262 | 007263 |
| 01C6 | 007264 | 007265 | 007266 | 007267 | 007268 | 007269 | 007270 | 007271 | 007272 | 007273 | 007274 | 007275 | 007276 | 007277 | 007278 | 007279 |
| 01C7 | 007280 | 007281 | 007282 | 007283 | 007284 | 007285 | 007286 | 007287 | 007288 | 007289 | 007290 | 007291 | 007292 | 007293 | 007294 | 007295 |
| 01C8 | 007296 | 007297 | 007298 | 007299 | 007300 | 007301 | 007302 | 007303 | 007304 | 007305 | 007306 | 007307 | 007308 | 007309 | 007310 | 007311 |
| 01C9 | 007312 | 007313 | 007314 | 007315 | 007316 | 007317 | 007318 | 007319 | 007320 | 007321 | 007322 | 007323 | 007324 | 007325 | 007326 | 007327 |
| 01CA | 007328 | 007329 | 007330 | 007331 | 007332 | 007333 | 007334 | 007335 | 007336 | 007337 | 007338 | 007339 | 007340 | 007341 | 007342 | 007343 |
| 01CB | 007344 | 007345 | 007346 | 007347 | 007348 | 007349 | 007350 | 007351 | 007352 | 007353 | 007354 | 007355 | 007356 | 007357 | 007358 | 007359 |
| 01CC | 007360 | 007361 | 007362 | 007363 | 007364 | 007365 | 007366 | 007367 | 007368 | 007369 | 007370 | 007371 | 007372 | 007373 | 007374 | 007375 |
| 01CD | 007376 | 007377 | 007378 | 007379 | 007380 | 007381 | 007382 | 007383 | 007384 | 007385 | 007386 | 007387 | 007388 | 007389 | 007390 | 007391 |

## HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE (Cont'd)

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 01E0 | 007680 | 007681 | 007682 | 007683 | 007684 | 007685 | 007686 | 007687 | 007688 | 007689 | 007690 | 007691 | 007692 | 007693 | 007694 | 007695 |
| 01E1 | 007696 | 007697 | 007698 | 007699 | 007700 | 007701 | 007702 | 007703 | 007704 | 007705 | 007706 | 007707 | 007708 | 007709 | 007710 | 007711 |
| 01E2 | 007712 | 007713 | 007714 | 007715 | 007716 | 007717 | 007718 | 007719 | 007720 | 007721 | 007722 | 007723 | 007724 | 007725 | 007726 | 007727 |
| 01E3 | 007728 | 007729 | 007730 | 007731 | 007732 | 007733 | 007734 | 007735 | 007736 | 007737 | 007738 | 007739 | 007740 | 007741 | 007742 | 007743 |
| 01E4 | 007744 | 007745 | 007746 | 007747 | 007748 | 007749 | 007750 | 007751 | 007752 | 007753 | 007754 | 007755 | 007756 | 007757 | 007758 | 007759 |
| 01E5 | 007760 | 007761 | 007762 | 007763 | 007764 | 007765 | 007766 | 007767 | 007768 | 007769 | 007770 | 007771 | 007772 | 007773 | 007774 | 007775 |
| 01E6 | 007776 | 007777 | 007778 | 007779 | 007780 | 007781 | 007782 | 007783 | 007784 | 007785 | 007786 | 007787 | 007788 | 007789 | 007790 | 007791 |
| 01E7 | 007792 | 007793 | 007794 | 007795 | 007796 | 007797 | 007798 | 007799 | 007800 | 007801 | 007802 | 007803 | 007804 | 007805 | 007806 | 007807 |
| 01E8 | 007808 | 007809 | 007810 | 007811 | 007812 | 007813 | 007814 | 007815 | 007816 | 007817 | 007818 | 007819 | 007820 | 007821 | 007822 | 007823 |
| 01E9 | 007824 | 007825 | 007826 | 007827 | 007828 | 007829 | 007830 | 007831 | 007832 | 007833 | 007834 | 007835 | 007836 | 007837 | 007838 | 007839 |
| 01EA | 007840 | 007841 | 007842 | 007843 | 007844 | 007845 | 007846 | 007847 | 007848 | 007849 | 007850 | 007851 | 007852 | 007853 | 007854 | 007855 |
| 01EB | 007856 | 007857 | 007858 | 007859 | 007860 | 007861 | 007862 | 007863 | 007864 | 007865 | 007866 | 007867 | 007868 | 007869 | 007870 | 007871 |
| 01EC | 007872 | 007873 | 007874 | 007875 | 007876 | 007877 | 007878 | 007879 | 007880 | 007881 | 007882 | 007883 | 007884 | 007885 | 007886 | 007887 |
| 01ED | 007888 | 007889 | 007890 | 007891 | 007892 | 007893 | 007894 | 007895 | 007896 | 007897 | 007898 | 007899 | 007900 | 007901 | 007902 | 007903 |
| 01EE | 007904 | 007905 | 007906 | 007907 | 007908 | 007909 | 007910 | 007911 | 007912 | 007913 | 007914 | 007915 | 007916 | 007917 | 007918 | 007919 |
| 01EF | 007920 | 007921 | 007922 | 007923 | 007924 | 007925 | 007926 | 007927 | 007928 | 007929 | 007930 | 007931 | 007932 | 007933 | 007934 | 007935 |

|      | 0      | 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | A      | B      | C      | D      | E      | F      |
|------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 01F0 | 007936 | 007937 | 007938 | 007939 | 007940 | 007941 | 007942 | 007943 | 007944 | 007945 | 007946 | 007947 | 007948 | 007949 | 007950 | 007951 |
| 01F1 | 007952 | 007953 | 007954 | 007955 | 007956 | 007957 | 007958 | 007959 | 007960 | 007961 | 007962 | 007963 | 007964 | 007965 | 007966 | 007967 |
| 01F2 | 007968 | 007969 | 007970 | 007971 | 007972 | 007973 | 007974 | 007975 | 007976 | 007977 | 007978 | 007979 | 007980 | 007981 | 007982 | 007983 |
| 01F3 | 007984 | 007985 | 007986 | 007987 | 007988 | 007989 | 007990 | 007991 | 007992 | 007993 | 007994 | 007995 | 007996 | 007997 | 007998 | 007999 |
| 01F4 | 008000 | 008001 | 008002 | 008003 | 008004 | 008005 | 008006 | 008007 | 008008 | 008009 | 008010 | 008011 | 008012 | 008013 | 008014 | 008015 |
| 01F5 | 008016 | 008017 | 008018 | 008019 | 008020 | 008021 | 008022 | 008023 | 008024 | 008025 | 008026 | 008027 | 008028 | 008029 | 008030 | 008031 |
| 01F6 | 008032 | 008033 | 008034 | 008035 | 008036 | 008037 | 008038 | 008039 | 008040 | 008041 | 008042 | 008043 | 008044 | 008045 | 008046 | 008047 |
| 01F7 | 008048 | 008049 | 008050 | 008051 | 008052 | 008053 | 008054 | 008055 | 008056 | 008057 | 008058 | 008059 | 008060 | 008061 | 008062 | 008063 |
| 01F8 | 008064 | 008065 | 008066 | 008067 | 008068 | 008069 | 008070 | 008071 | 008072 | 008073 | 008074 | 008075 | 008076 | 008077 | 008078 | 008079 |
| 01F9 | 008080 | 008081 | 008082 | 008083 | 008084 | 008085 | 008086 | 008087 | 008088 | 008089 | 008090 | 008091 | 008092 | 008093 | 008094 | 008095 |
| 01FA | 008096 | 008097 | 008098 | 008099 | 008100 | 008101 | 008102 | 008103 | 008104 | 008105 | 008106 | 008107 | 008108 | 008109 | 008110 | 008111 |
| 01FB | 008112 | 008113 | 008114 | 008115 | 008116 | 008117 | 008118 | 008119 | 008120 | 008121 | 008122 | 008123 | 008124 | 008125 | 008126 | 008127 |
| 01FC | 008128 | 008129 | 008130 | 008131 | 008132 | 008133 | 008134 | 008135 | 008136 | 008137 | 008138 | 008139 | 008140 | 008141 | 008142 | 008143 |
| 01FD | 008144 | 008145 | 008146 | 008147 | 008148 | 008149 | 008150 | 008151 | 008152 | 008153 | 008154 | 008155 | 008156 | 008157 | 008158 | 008159 |
| 01FE | 008160 | 008161 | 008162 | 008163 | 008164 | 008165 | 008166 | 008167 | 008168 | 008169 | 008170 | 008171 | 008172 | 008173 | 008174 | 008175 |
| 01FF | 008176 | 008177 | 008178 | 008179 | 008180 | 008181 | 008182 | 008183 | 008184 | 008185 | 008186 | 008187 | 008188 | 008189 | 008190 | 008191 |

# APPENDIX I

## SCRATCH-PAD MEMORY LAYOUT AND REGISTER ASSIGNMENTS

| Hexadecimal | 0                                 | 1                                      | 2                                 | 3                                 | 4                                 | 5                                 | 6                                 | 7                                 | 8                                      | 9                                      | A                                  | B                                  | C                                  | D                                  | E                                  | F   |
|-------------|-----------------------------------|--|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|--|--|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|---|
| 0           | PROCESSOR UTILITY                 |  |                                   |                                   |                                   |                                   |                                   |                                   | GENERAL PURPOSE REGISTER NO. 8 P4      | GENERAL PURPOSE REGISTER NO. 9 P4      | GENERAL PURPOSE REGISTER NO. 10 P4 | GENERAL PURPOSE REGISTER NO. 11 P4 | INTERRUPT MASK REGISTER P4         | INTERRUPT STATUS REGISTER P4       | PROGRAM COUNTER P4                 | GENERAL PURPOSE REGISTER NO. 15 (WEIGHT) P4 |
| 1           | PROCESSOR UTILITY                 | I/O CHANNEL REGISTERS - MULTIPLEXOR    |                                   |                                   |                                   | PROCESSOR UTILITY                 |                                   |                                   |  | I/O CHANNEL REGISTERS - SELECTOR NO. 1 |                                    |                                    |                                    | PROCESSOR UTILITY                  |                                    |   |
|             |                                   | CHANNEL ADDRESS REGISTER               | CHANNEL COMMAND REGISTER II       | CHANNEL COMMAND REGISTER I        | STATUS REGISTER                   |                                   |                                   |                                   |  | CHANNEL ADDRESS REGISTER               | CHANNEL COMMAND REGISTER II        | CHANNEL COMMAND REGISTER I         | ASSEMBLY STATUS REGISTER           |                                    |                                    |   |
| 2           | INTERRUPT MASK REGISTER P1        | INTERRUPT STATUS REGISTER P1           | PROGRAM COUNTER P1                | INTERRUPT FLAG REGISTER           | INTERRUPT MASK REGISTER P2        | INTERRUPT STATUS REGISTER P2      | PROGRAM COUNTER P2                | GENERAL PURPOSE REGISTER NO. 7 P3 | INTERRUPT MASK REGISTER P3             | INTERRUPT STATUS REGISTER P3           | PROGRAM COUNTER P3                 | GENERAL PURPOSE REGISTER NO. 11 P3 | GENERAL PURPOSE REGISTER NO. 12 P3 | GENERAL PURPOSE REGISTER NO. 13 P3 | GENERAL PURPOSE REGISTER NO. 14 P3 | GENERAL PURPOSE REGISTER NO. 15 (WEIGHT) P3 |
| 3           | PROCESSOR UTILITY                 | I/O CHANNEL REGISTERS - SELECTOR NO. 2 |                                   |                                   |                                   | PROCESSOR UTILITY                 |                                   |                                   |  | I/O CHANNEL REGISTERS - SELECTOR NO. 3 |                                    |                                    |                                    | PROCESSOR UTILITY                  |                                    |   |
|             |                                   | CHANNEL ADDRESS REGISTER               | CHANNEL COMMAND REGISTER II       | CHANNEL COMMAND REGISTER I        | ASSEMBLY STATUS REGISTER          |                                   |                                   |                                   |  | CHANNEL ADDRESS REGISTER               | CHANNEL COMMAND REGISTER II        | CHANNEL COMMAND REGISTER I         | ASSEMBLY STATUS REGISTER           |                                    |                                    |   |
| 4           | GENERAL PURPOSE REGISTER NO. 0 P2 | GENERAL PURPOSE REGISTER NO. 1 P2      | GENERAL PURPOSE REGISTER NO. 2 P2 | GENERAL PURPOSE REGISTER NO. 3 P2 | GENERAL PURPOSE REGISTER NO. 4 P2 | GENERAL PURPOSE REGISTER NO. 5 P2 | GENERAL PURPOSE REGISTER NO. 6 P2 | GENERAL PURPOSE REGISTER NO. 7 P2 | GENERAL PURPOSE REGISTER NO. 8 P2      | GENERAL PURPOSE REGISTER NO. 9 P2      | GENERAL PURPOSE REGISTER NO. 10 P2 | GENERAL PURPOSE REGISTER NO. 11 P2 | GENERAL PURPOSE REGISTER NO. 12 P2 | GENERAL PURPOSE REGISTER NO. 13 P2 | GENERAL PURPOSE REGISTER NO. 14 P2 | GENERAL PURPOSE REGISTER NO. 15 P2          |
| 5           | PROCESSOR UTILITY                 | I/O CHANNEL REGISTERS - SELECTOR NO. 4 |                                   |                                   |                                   | PROCESSOR UTILITY                 |                                   |                                   |  | I/O CHANNEL REGISTERS - SELECTOR NO. 5 |                                    |                                    |                                    | PROCESSOR UTILITY                  |                                    |   |
|             |                                   | CHANNEL ADDRESS REGISTER               | CHANNEL COMMAND REGISTER II       | CHANNEL COMMAND REGISTER I        | ASSEMBLY STATUS REGISTER          |                                   |                                   |                                   |  | CHANNEL ADDRESS REGISTER               | CHANNEL COMMAND REGISTER II        | CHANNEL COMMAND REGISTER I         | ASSEMBLY STATUS REGISTER           |                                    |                                    |   |
| 6           | GENERAL PURPOSE REGISTER NO. 0 P1 | GENERAL PURPOSE REGISTER NO. 1 P1      | GENERAL PURPOSE REGISTER NO. 2 P1 | GENERAL PURPOSE REGISTER NO. 3 P1 | GENERAL PURPOSE REGISTER NO. 4 P1 | GENERAL PURPOSE REGISTER NO. 5 P1 | GENERAL PURPOSE REGISTER NO. 6 P1 | GENERAL PURPOSE REGISTER NO. 7 P1 | GENERAL PURPOSE REGISTER NO. 8 P1      | GENERAL PURPOSE REGISTER NO. 9 P1      | GENERAL PURPOSE REGISTER NO. 10 P1 | GENERAL PURPOSE REGISTER NO. 11 P1 | GENERAL PURPOSE REGISTER NO. 12 P1 | GENERAL PURPOSE REGISTER NO. 13 P1 | GENERAL PURPOSE REGISTER NO. 14 P1 | GENERAL PURPOSE REGISTER NO. 15 P1          |
| 7           | FLOATING-POINT REGISTER NO. 0     | FLOATING-POINT REGISTER NO. 2          | FLOATING-POINT REGISTER NO. 4     | FLOATING-POINT REGISTER NO. 6     | PROCESSOR UTILITY                 |                                   |                                   |                                   | I/O CHANNEL REGISTERS - SELECTOR NO. 6 |  |                                    |                                    | PROCESSOR UTILITY                  |                                    |                                    |   |
|             |                                   |  |                                   |                                   |                                   |                                   |                                   |                                   |  | CHANNEL ADDRESS REGISTER               | CHANNEL COMMAND REGISTER II        | CHANNEL COMMAND REGISTER I         | ASSEMBLY STATUS REGISTER           |                                    |                                    |   |

\* Word Address is in Hexadecimal; e.g., 2A - Program Counter P3.